ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
ΥΠΟΥΡΓΕΙΟ ΕΣΩΤΕΡΙΚΩΝ

εκδδα
ΕΘΝΙΚΟ ΚΕΝΤΡΟ ΔΗΜΟΣΙΑΣ ΔΙΟΙΚΗΣΗΣ ΚΑΙ ΑΥΤΟΔΙΟΙΚΗΣΗΣ

ΥΠΟΕΡΓΟ: ΥΠΟΕΡΓΟ 2 «ΠΡΟΓΡΑΜΜΑΤΑ ΚΑΤΑΡΤΙΣΗΣ, ΑΝΑΠΤΥΞΗΣ ΔΕΞΙΟΤΗΤΩΝ, ΕΝΔΥΝΑΜΩΣΗΣ, ΠΙΣΤΟΠΟΙΗΣΗΣ - ΥΛΟΠΟΙΗΣΗ ΜΕ ΙΔΙΑ ΜΕΣΑ, ΕΠΙΜΟΡΦΩΣΗ ΑΠΟ ΤΟ ΕΚΔΔΑ» του Έργου «SUB4. Αναβάθμιση των δεξιοτήτων του ανθρώπινου δυναμικού του Δημόσιου Τομέα» με κωδικό ΟΠΣ ΤΑ 5150174
της Δράσης 16972 ΤΑΑ

ΤΙΤΛΟΣ ΠΡΟΓΡΑΜΜΑΤΟΣ:
"JAVA - ΕΙΣΑΓΩΓΗ ΣΤΟΝ ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΑΦΗ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟ"

ΕΚΠΑΙΔΕΥΤΙΚΟ ΥΛΙΚΟ

Κωδικός εκπαιδευτικού υλικού:

Κωδικός Πιστοποίησης προγράμματος: 1009

Ελλάδα 2.0
ΕΘΝΙΚΟ ΣΧΕΔΙΟ ΑΝΑΚΑΜΨΗΣ ΚΑΙ ΑΝΘΕΚΤΙΚΟΤΗΤΑΣ

Με τη χρηματοδότηση της Ευρωπαϊκής Ένωσης
NextGenerationEU

ΥΠΟΕΡΓΟ: : ΥΠΟΕΡΓΟ 2 «ΠΡΟΓΡΑΜΜΑΤΑ ΚΑΤΑΡΤΙΣΗΣ, ΑΝΑΠΤΥΞΗΣ ΔΕΞΙΟΤΗΤΩΝ, ΕΝΔΥΝΑΜΩΣΗΣ, ΠΙΣΤΟΠΟΙΗΣΗΣ - ΥΛΟΠΟΙΗΣΗ ΜΕ ΙΔΙΑ ΜΕΣΑ, ΕΠΙΜΟΡΦΩΣΗ ΑΠΟ ΤΟ ΕΚΔΔΑ» του Έργου «SUB4. Αναβάθμιση των δεξιοτήτων του ανθρώπινου δυναμικού του Δημόσιου Τομέα» με κωδικό ΟΠΣ ΤΑ 5150174 της Δράσης 16972 ΤΑΑ

ΤΙΤΛΟΣ ΠΡΟΓΡΑΜΜΑΤΟΣ:

JAVA - ΕΙΣΑΓΩΓΗ ΣΤΟΝ ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΑΦΗ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟ

ΟΜΑΔΑ ΕΡΓΑΣΙΑΣ

Μέλη Ομάδας

Συντονιστής:
Χαραλαμπία Δουλή

Συγγραφείς:
Δημήτριος Τσέλιος
Γεώργιος Χρ. Μακρής
Γεώργιος Μαυρομάτης
Αντώνιος Στεφάνου

Αξιολογητές:
Ματζαβάκης Ιωάννης
Γεώργιος Παπαμιχαήλ

Ελλάδα 2.0
ΕΘΝΙΚΟ ΣΧΕΔΙΟ ΑΝΑΚΑΜΨΗΣ ΚΑΙ ΑΝΘΕΚΤΙΚΟΤΗΤΑΣ

Με τη χρηματοδότηση της Ευρωπαϊκής Ένωσης
NextGenerationEU

Java

Εισαγωγή στον Αντικειμενοστραφή Προγραμματισμό

ΕΚΠΑΙΔΕΥΤΙΚΟ ΥΛΙΚΟ

Περιεχόμενα

Λίγα λόγια για τους συγγραφείς

Δρ. Δημήτριος Τσέλιος

Ο Δρ. Δημήτριος Τσέλιος είναι Αναπληρωτής Καθηγητής του Τμήματος Διοίκησης Επιχειρήσεων της Σχολής Οικονομικών και Διοικητικών Επιστημών του Πανεπιστημίου Θεσσαλίας και διευθυντής του εργαστηρίου Επιχειρηματικής Αναλυτικής. Είναι μέλος ΣΕΠ του ΕΑΠ από το 2017 και συντονιστής της θεματικής ενότητας ΔΗΔ53 από τον Οκτώβριο του 2020. Διετέλεσε μέλος ΕΠ και ΔΕΠ από τον Μάρτιο του 2005 μέχρι τον Ιανουάριο του 2019 στο ΤΕΙ Λάρισας και στο ΤΕΙ Θεσσαλίας. Κατέχει Διδακτορικό δίπλωμα στην Επιστήμη των Υπολογιστών από το Εθνικό Πανεπιστήμιο της Ιρλανδίας (University College Dublin), ΜΔΕ στη Διασφάλιση Ποιότητας και πτυχίο στην Επιστήμη των Υπολογιστών. Έχει συμμετοχή στη σύνταξη πλήθους δημοσιεύσεων σε επιστημονικά περιοδικά και συνέδρια ενώ τα ενδιαφέροντα του σχετίζονται με θέματα Χρονοπρογραμματισμού, Διαχείρισης Έργων, Ευφυών Μεθόδων και χρήσης ΙΤ στην εκπαίδευση. Επίσης συμμετείχε σε διάφορα ερευνητικά έργα του ΤΕΙ Λάρισας και του ΤΕΙ Θεσσαλίας και του Πανεπιστημίου Θεσσαλίας.

Δρ. Γεώργιος Μακρής

Ο κος Γεώργιος Μακρής γεννήθηκε το 1971 στο Χέρφορντ της Γερμανίας μεγάλωσε και ζει στην Κατερίνη. Από τον Ιούλιο του 2022 είναι Διευθυντής της Διεύθυνσης Δευτεροβάθμιας Εκπαίδευσης Πιερίας.

Έχει σπουδάσει Μαθηματικά, Πληροφορική και Οικονομικές Επιστήμες. Έχει μεταπτυχιακές σπουδές: α) στην Εκπαίδευση, β) στην Θεωρητική Πληροφορική και Συστήματα Αυτομάτου Ελέγχου, γ) στα Πληροφοριακά Συστήματα και δ) στην Επιστήμη του Διαδικτύου. Η διδακτορική του διατριβή έχει θέμα: «Κρυπτογραφία με Χάος» στο τμήμα Μαθηματικών του ΑΠΘ. Έχει ολοκληρώσει δύο μεταδιδακτορικές έρευνες α) στο τμήμα Μαθηματικών του ΑΠΘ με θέμα: «Ανάπτυξη Νέων Στατιστικών και Υπολογιστικών Εργαλείων για την Ανάλυση Δικτύων» (Development of new statistical and computational tools for the analysis of networks) και β) στο τμήμα Πληροφορικής του ΑΠΘ με θέμα: «Μοντελοποίηση του Οικοσυστήματος του Ανοιχτού Κώδικα».

Η διδακτική του εμπειρία περιλαμβάνει μαθήματα πληροφορικής και μαθηματικών τόσο στην δευτεροβάθμια όσο και στην τριτοβάθμια εκπαίδευση. Έχει διδάξει επί σειρά ετών μαθήματα προπτυχιακού και μεταπτυχιακού επιπέδου.

Τα ερευνητικά του ενδιαφέροντα σχετίζονται με: «Δίκτυα και Πολύπλοκα συστήματα», «Κρυπτογραφία και Στεγανογραφία», «Δυναμικά Συστήματα και Νευρωνικά Δίκτυα» και «Γλώσσες Προγραμματισμού». Είναι συγγραφέας 7 βιβλίων και ενός συλλογικού τόμου. Έχει πλήθος δημοσιεύσεων τόσο σε ξένα όσο και σε ελληνικά περιοδικά και συνέδρια. Ασχολείται με τον προγραμματισμό από το 1985 και έχει ευχέρεια προγραμματισμού σε πολλές γλώσσες όπως: C, C++, Java, Python, Visual Basic, Assembly κλπ.

Εργάστηκε στον ιδιωτικό τομέα για 10 χρόνια. Από το 2002 εργάζεται ως εκπαιδευτικός στο Δημόσιο Σχολείο, διετέλεσε για 6 χρόνια υποδιευθυντής στο 3ο ΓΕΛ Κατερίνης και για 5 χρόνια Διευθυντής στο 4ο ΓΕΛ.

Δρ. Γεώργιος Μαυρομάτης

Μέλος Ειδικού Επιστημονικού Προσωπικού στο ΕΚΔΔΑ, επιστημονικό προσωπικό στο Κοινωνικό Πολύκεντρο, Ινστιτούτο της ΑΔΕΔΥ.

Πτυχιούχος Μαθηματικών Πανεπιστημίου Αθηνών. Διδάκτορας Τμήματος Πληροφορικής του Πανεπιστημίου Πειραιώς. Μεταδιδακτορικός Ερευνητής του τμήματος Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών του Πανεπιστημίου Θεσσαλίας.

Διδασκαλία προπτυχιακών και μεταπτυχιακών μαθημάτων σε Πανεπιστήμιο Πειραιώς, ΕΚΠΑ, ΣΣΕ, ΠΑΔΑ. Μέλος Συνεργαζόμενου Εκπαιδευτικού Προσωπικού (ΣΕΠ) στο Ελληνικό Ανοικτό Πανεπιστήμιο (Μεταπτυχιακή εξειδίκευση στα Πληροφοριακά Συστήματα). Εισηγητής σε ΕΣΔΔΑ και ΙΝΕΠ.

Συγγραφέας βιβλίων "Γλώσσες Προγραμματισμού" και "Τεχνολογία Λογισμικού" για το ΕΑΠ.

Κύρια επιστημονικά ενδιαφέροντα: Προγραμματισμός, Αλγόριθμοι και δομές δεδομένων, Εκπαιδευτική τεχνολογία & ηλεκτρονική μάθηση, Ηλεκτρονική Διακυβέρνηση, Τεχνολογία λογισμικού & διοίκηση έργων, Επιχειρησιακή έρευνα, Ανάκτηση πληροφορίας, Εξόρυξη δεδομένων, Παράλληλα συστήματα, Μεγάλα δεδομένα, Γεωχωρικά δεδομένα.

MSc Αντώνιος Στεφάνου

Ο κος **Αντώνιος Στεφάνου** είναι κάτοχος πτυχίου «Επιστήμης Υπολογιστών» από σχολή Θετικών Επιστημών τμήμα Ηλεκτρονικών Υπολογιστών του Πανεπιστημίου Κρήτης, κάτοχος Μεταπτυχιακού τίτλου σπουδών στα «Δίκτυα Υπολογιστών και Επικοινωνίες (Data Networks and Communications)», με μεταπτυχιακή εργασία τον σχεδιασμό ενός πακέτου σχεδίασης στη δικτυακή γλώσσα Java, κάτοχος Μεταπτυχιακού τίτλου σπουδών στην «Ανοιχτή Διακυβέρνηση (Open eGov)», από το Πανεπιστήμιο της Στοκχόλμης, καθώς και κάτοχος πολλαπλών βεβαιώσεων και πιστοποιήσεων.

Κατά το χρονικό διάστημα των σπουδών του, εργάστηκε στο Ίδρυμα Τεχνολογίας και Έρευνας Κρήτης (Ι.Τ.Ε.) ως Υπεύθυνος Οργάνωσης Σεμιναρίων και Τεχνικός Συστημάτων στο Εκπαιδευτικό Παράρτημα του Ι.Τ.Ε με στόχο την κατάρτιση Ελευθέρων Επαγγελματιών σε παραθυριακά λογισμικά πακέτα καθώς και σε περιβάλλοντα δικτύου.

Κατά το χρονικό διάστημα 1998-2003, εργάστηκε στην εταιρεία European Dynamicsως Μηχανικός Λογισμικού και Τεχνικός Υπεύθυνος Έργων Πληροφορικής, όπως των έργων CATIF, SEI-JOS (PROJECT NO: 97/S 182-116098), EDIFLOW, STADIUM (PROJECT NO: SOEC No. 9143006 – Lot 1, SOEC No. 91543005 – Lot 5) με κύριους πελάτες το Γραφείο Εκδόσεων (publication office) της Ευρωπαϊκής Κοινότητας (OPOCE) και τη Στατιστική Υπηρεσία της Ευρωπαϊκής Κοινότητας (EUROSTAT).

Κατά το χρονικό διάστημα 2003–2006 εργάστηκε αρχικά ως Υπεύθυνος Έργων (Project Manager) και αρμόδιος για την Ανάλυση και Σχεδίαση Πληροφοριακών Συστημάτων για έργα όπως το e-MATE (electronic multilingual tourism environment, PROJECT NO: 22018Y1C1DMAL2), «Χαρτογράφηση επιλεγμένων λειτουργιών της Δημόσιας Διοίκησης και Στρατηγική Θεσμικής Διαλειτουργικότητας» (προκήρυξη ΥΠΕΣΔΔΑ Νο ΔΙΟΙΚ/Φ.370Γ/1/19810/15.10.2003 του ΥΠΕΣΔΔΑ, καθώς και ως Presales Manager, υπεύθυνος για την αξιολόγηση, την οργάνωση και την συγγραφή Τεχνικών Προσφορών και Τευχών έργων Ελληνικού Δημοσίου, R&D και Εμπορικών Έργων Ευρωπαϊκής Κοινότητας.

Κατά τα έτη 2006-2008 και μετά από γραπτό διαγωνισμό του ΑΣΕΠ, εργάστηκε ως εκπαιδευτικός Δευτεροβάθμιας Εκπαίδευσης για το Υπουργείο Παιδείας και Θρησκευμάτων.

Εν συνεχεία κατά τα έτη 2008-2021 εργάστηκε ως Υπεύθυνος Διαχείρισης Έργων, με αρμοδιότητα τη μεθοδολογική παρακολούθησης έως και 12 ταυτόχρονων έργων με ετήσια –ανανεώσιμη- διάρκεια στα πλαίσια του framework έργου «Υποστήριξη Υπουργείου Παιδείας, Έρευνας &

Θρησκευμάτων σε Επιστημονικά και Τεχνολογικά Θέματα 2016» καθώς και Υπεύθυνος Ποιότητας για το έργο της ανάπτυξης του Ηλεκτρονικού Μηχανογραφικού και γενικότερα για έργα που σχετίζονται με την διεξαγωγή των Πανελλαδικών Εξετάσεων.

Από το 2021 και εν συνεχεία, εργάζεται στο Τμήμα Εφαρμογών Πληροφορικής του Εθνικού Κέντρου Δημόσιας Διοίκησης και Αυτοδιοίκησης (ΕΚΔΔΑ).

Εισαγωγή

Καλώς ήρθατε στο μάθημα ηλεκτρονικής διδασκαλίας «Java-Εισαγωγή στον Αντικειμενοστραφή Προγραμματισμό» του Εθνικού Κέντρου Δημόσιας Διοίκησης και Αυτοδιοίκησης (Ε.Κ.Δ.Δ.Α). Το μάθημα αυτό απευθύνεται σε όσους επιθυμούν να ξεκινήσουν τον προγραμματισμό εφαρμογών στη γλώσσα προγραμματισμού Java αλλά δεν έχουν προγενέστερη εμπειρία στη γλώσσα αυτή ή γενικότερα στον αντικειμενοστραφή προγραμματισμό. Σκοπός του είναι η εκμάθηση των βασικών δυνατοτήτων της Java μέσα από μία σειρά 10 εκπαιδευτικών ενοτήτων οι οποίες καλύπτουν όλα όσα θα πρέπει να γνωρίζει κάποιος για να ξεκινήσει τον προγραμματισμό εφαρμογών σε αυτή τη γλώσσα.

Ιδιαίτερη έμφαση δίνεται στη σωστή εκμάθηση της γλώσσας και για το λόγο αυτό τόσο ο κώδικας που θα συναντήσετε στις σημειώσεις και τις παρουσιάσεις ακολουθεί πιστά τους κανόνες σωστής πρακτικής που έχουν θεσπιστεί από την προγραμματιστική κοινότητα για τη δημιουργία «σωστών» και ευανάγνωστων προγραμμάτων. Οι κανόνες αυτοί θα σας παρουσιαστούν αναλυτικά κατά τη διάρκεια του μαθήματος και σας προτρέπουμε να τους υιοθετήσετε ώστε να μάθετε να γράφετε σωστά δομημένο κώδικα.

Στο πλαίσιο του μαθήματος, οι εκπαιδευόμενοι θα μάθουν πώς να αναπτύσσουν εφαρμογές με χρήση αντικειμενοστραφή προγραμματισμού στη γλώσσα προγραμματισμού Java, να χειρίζονται δεδομένα και να μπορούν να εφαρμόσουν τις γνώσεις τους για να σχεδιάζουν από απλά προγράμματα έως σύνθετα βάσει της αρχιτεκτονικής κώδικα του MVC (Model-View-Controller). Θα εξεταστούν θέματα όπως η ανάπτυξη κλάσεων και αντικειμένων, βασικές έννοιες αντικειμενοστραφούς προγραμματισμού όπως η κληρονομικότητα, η ενθυλάκωση, ο πολυμορφισμός, οι Αφαιρετικοί Τύποι Δεδομένων σε Java, η διαχείριση εξαιρέσεων, ο σχεδιασμός γραφικών Διεπαφών Χρήσης καθώς και η διαχείριση αρχείων.

Η ανάγκη για το μάθημα αυτό προκύπτει από την “Εθνική Στρατηγική για την Ανάπτυξη Ψηφιακής Πολιτικής” και την Αναλυτική Δεδομένων, καθώς και τις σύγχρονες ψηφιακές εξελίξεις όπως αυτές περιγράφονται στη Βίβλο Ψηφιακού Μετασχηματισμού 2020-2025. Η ορθά εστιασμένη εφαρμογή του προγραμματισμού στη γλώσσα Java μπορεί να βοηθήσει την Ελληνική Δημόσια Διοίκηση να γίνει ένας αποδοτικός και εξατομικευμένος πάροχος υπηρεσιών προς τους πολίτες και τις επιχειρήσεις.

ΚΕΦΑΛΑΙΟ 1: Εισαγωγή στην Java

1.1. Εκπαιδευτικοί στόχοι του παρόντος κεφαλαίου

Οι εκπαιδευτικοί στόχοι του παρόντος κεφαλαίου, συνοψίζονται στα κάτωθι σημεία. Οι εκπαιδευόμενοι:

- θα γνωρίσουν βασικά στοιχεία για την Επίσημη Εξέταση Πιστοποίησης της Oracle,
- θα μάθουν για την Ιστορία της Java,
- θα γνωρίσουν τα χαρακτηριστικά της Java,
- θα γνωρίσουν την αρχιτεκτονική της γλώσσας,
- θα γνωρίσουν το Java SE Development Kit (JDK) και Java Runtime Environment (JRE) και τον τρόπο χρήσης τους,
- θα γνωρίζουν το Java Virtual Machine (JVM),
- θα κατανοήσουν τον Κύκλο Υλοποίησης προγράμματος,
- θα γνωρίσουν τα Ολοκληρωμένα Περιβάλλοντα Ανάπτυξης - Integrated Development Environments (IDEs) και θα εγκαταστήσουν κάποιο από αυτά.

1.2. Η Επίσημη Εξέταση Πιστοποίησης

Η τέως Sun Microsystems (νυν Oracle μετά την εξαγορά) προσφέρει τη δυνατότητα επίσημης πιστοποίησης γνώσεων στη γλώσσα Java, όπως άλλωστε και οι περισσότερες εταιρείες που παράγουν τεχνολογία (Cisco, Microsoft κλπ). Συγκεκριμένα, παρέχονται οκτώ διαφορετικά διαγωνίσματα τα οποία οδηγούν στην αντίστοιχη πιστοποίηση και διαφοροποιούνται ως προς το επίπεδο δυσκολίας αλλά και το αντικείμενο εξέτασης.

Οι πιστοποιήσεις Java που παρέχονται από την Oracle είναι σχεδιασμένες για να επιβεβαιώνουν τις δεξιότητες και τις γνώσεις σας στη γλώσσα προγραμματισμού Java. Ακολουθούν οι κύριες πιστοποιήσεις που προσφέρονται:

1. Oracle Certified Associate (OCA)

Αυτή η πιστοποίηση είναι για αρχάριους και καλύπτει τις βασικές αρχές της Java. Είναι το πρώτο βήμα για να γίνετε επαγγελματίας προγραμματιστής Java.

2. Oracle Certified Professional (OCP)

Το αμέσως επόμενο επίπεδο πιστοποίησης, το διαγώνισμα της οποίας εξετάζει τις βασικές γνώσεις του υποψηφίου στη Java και την ικανότητά του να χρησιμοποιεί σωστά τα εργαλεία της γλώσσας. Είναι η πιστοποίηση την οποία θα είστε σε θέση να διεκδικήσετε ολοκληρώνοντας επιτυχώς το μάθημα αυτό και έχοντας κατανοήσει το σύνολο της διδακτέας ύλης. Η συγκεκριμένη πιστοποίηση χαίρει μεγάλης αποδοχής στην αγορά εργασίας και αποτελεί σίγουρα ένα πρόσθετο εφόδιο σε όποιον την κατέχει.

3. Oracle Certified Expert (OCE)

Αυτή η πιστοποίηση επικεντρώνεται σε εξειδικευμένα θέματα, όπως η ασφάλεια Java, η ανάπτυξη web εφαρμογών και η χρήση συγκεκριμένων API.

4. Oracle Certified Master (OCM)

Αυτή είναι η πιο προχωρημένη πιστοποίηση και απαιτεί εκτεταμένη εμπειρία και γνώση. Περιλαμβάνει πρακτικές εξετάσεις και απαιτεί την επίλυση σύνθετων προβλημάτων.

- **Oracle Certified Associate (OCA), Java SE 6:** Εισαγωγική πιστοποίηση, το διαγώνισμα της οποίας εξετάζει τις στοιχειώδεις γνώσεις στη γλώσσα Java. Δεν έχει μεγάλη πρακτική αξία.
- **Oracle Certified Professional (OCP), Java SE 6 Programmer (τέως SCJP):** Το αμέσως επόμενο επίπεδο πιστοποίησης, το διαγώνισμα της οποίας εξετάζει τις βασικές γνώσεις του υποψηφίου στη Java SE 6 και την ικανότητά του να χρησιμοποιεί σωστά τα εργαλεία της γλώσσας. Είναι η πιστοποίηση την οποία θα είστε σε θέση να διεκδικήσετε ολοκληρώνοντας επιτυχώς το μάθημα αυτό και έχοντας κατανοήσει το σύνολο της διδακτέας ύλης. Η συγκεκριμένη πιστοποίηση χαίρει μεγάλης αποδοχής στην αγορά εργασίας και αποτελεί σίγουρα ένα πρόσθετο εφόδιο σε όποιον την κατέχει.
- **Oracle Certified Master (OCM), Java SE 6 Developer:** Ένα βήμα ανώτερο από την OCP, η συγκεκριμένη πιστοποίηση απευθύνεται σε όσους επιθυμούν να έχουν επίσημη απόδειξη για την ικανότητά τους να υλοποιούν προχωρημένες εφαρμογές στη Java SE 6. Για την επίτευξη της πιστοποίησης αυτής εκτός από το ηλεκτρονικό διαγώνισμα απαιτείται και η εκπόνηση μιας εργασίας. Επίσης, ως προαπαιτούμενο, ο κάθε υποψήφιος θα πρέπει να κατέχει πιστοποίηση SCJP για την έκδοση 6 ή κάποια άλλη προγενέστερη έκδοση της Java.

Οι ερωτήσεις που περιέχονται στο διαγώνισμα είναι όλων των ειδών, υπάρχουν εύκολες ερωτήσεις, υπάρχουν ερωτήσεις παγίδες (trick questions) και τέλος υπάρχουν και δύσκολες ερωτήσεις που για να απαντηθούν σωστά θα πρέπει ο υποψήφιος να έχει εμβαθύνει αρκετά στη γνώση της γλώσσας. Ως αποτέλεσμα, το τελικό σκορ του κάθε υποψηφίου αντικατοπτρίζει πλήρως τις γνώσεις του και τις δεξιότητές του στη Java.

Από τις παραπάνω πιστοποιήσεις, εκείνη που καλύπτεται από τη διδακτέα ύλη του μαθήματος και που μας ενδιαφέρει είναι η OCP. Για την επίτευξη της συγκεκριμένης πιστοποίησης ο υποψήφιος καλείται να περάσει επιτυχώς ένα ηλεκτρονικό διαγώνισμα με ερωτήσεις πολλαπλής επιλογής. Το διαγώνισμα λαμβάνει χώρα σε κάποιο εγκεκριμένο εξεταστικό κέντρο (VUE ή Prometric), συγκεκριμένη μέρα και ώρα που έχει συμφωνηθεί μεταξύ του υποψηφίου και των εξεταστών. Για όσους δεν έχουν προηγούμενη πιστοποίηση SCJP σε προγενέστερη έκδοση της Java ο κωδικός του διαγωνίσματος είναι ο CX-310-065. Η διάρκεια του συγκεκριμένου διαγωνίσματος είναι 180 λεπτά, μέσα στα οποία ο υποψήφιος καλείται να απαντήσει 60 ερωτήσεις πολλαπλής επιλογής με συνήθως 5 πιθανές απαντήσεις. Το σκορ βάσης είναι το 58,33%, δηλαδή θα πρέπει να απαντηθούν σωστά 35 από τις 60 ερωτήσεις.

Αντίστοιχα, αν κάποιος κατέχει πιστοποίηση SCJP σε προγενέστερη έκδοση της Java, μπορεί να εξεταστεί στο διαγώνισμα με κωδικό CX-310-066, το οποίο είναι διαγώνισμα αναβάθμισης και περιλαμβάνει λιγότερες ερωτήσεις αλλά και μικρότερη χρονική διάρκεια (48 ερωτήσεις σε 150

λεπτά). Το σκορ βάσης σε αυτήν την περίπτωση είναι το 66% δηλαδή θα πρέπει να απαντηθούν σωστά 32 από τις 48 ερωτήσεις.

Το αποτέλεσμα της εξέτασης γίνεται άμεσα γνωστό στον υποψήφιο με το πέρας της διαδικασίας. Θα πρέπει να τονιστεί σε αυτό το σημείο πως τα διαγωνίσματα διατίθενται στην Αγγλική γλώσσα. Επίσης, κατά τη διάρκεια του διαγωνίσματος όπως είναι φυσικό, ο υποψήφιος δε μπορεί να κάνει χρήση ηλεκτρονικών βοηθημάτων (PC, κινητό κλπ) και το μόνο που μπορεί να χρησιμοποιήσει είναι κόλλες αναφοράς και στυλό που του παρέχονται από το εξεταστικό κέντρο.

1.3. Η ιστορία της Java

Η Java είναι μια σύγχρονη αντικειμενοστραφής (object oriented) γλώσσα προγραμματισμού με αρκετά χαρακτηριστικά που δεν συναντώνται σε άλλες γλώσσες προγραμματισμού. Αναπτύχθηκε το 1991 από τον James Gosling (ο πατέρας της Java) και την Sun Microsystems. Εκείνη την εποχή η Sun Microsystems αναζητούσε ένα κατάλληλο εργαλείο που θα μπορούσε να χρησιμοποιηθεί ως πλατφόρμα ανάπτυξης λογισμικού για «έξυπνες» μικροσυσκευές. Ο Gosling αποφάσισε να πειραματιστεί με τη δημιουργία μιας νέας γλώσσας η οποία θα είχε αρκετά από τα χαρακτηριστικά της C++ και θα πρόσθετε τα χαρακτηριστικά εκείνα που χρειάζονταν για τον προγραμματισμό μικροσυσκευών. Ύστερα από μία σειρά ενδιάμεσων γλώσσων κατέληξε στην Oak (βελανιδιά). Η γλώσσα ονομάστηκε έτσι από μία βελανιδιά που βρισκόταν έξω από το γραφείο του και που έβλεπε καθημερινά. Η συγκεκριμένη γλώσσα διατηρούσε μεγάλη συγγένεια με τη C++ αλλά είχε πιο έντονο αντικειμενοστρεφή χαρακτήρα και χαρακτηριζόταν από την απλότητά της. Λίγο αργότερα η ομάδα ανάπτυξης της Oak ενημερώθηκε πως το συγκεκριμένο όνομα ήταν ήδη κατοχυρωμένο, ονόμασε την νέα γλώσσα Java (καφές). Το 2007 η Sun Microsystems αποφασίζει να κάνει όλον τον πηγαίο κώδικα της Java open source με μοναδική εξαίρεση ένα μικρό κομμάτι, του οποίου τα πνευματικά δικαιώματα δεν κατείχε η ίδια. Στις 27 Απριλίου 2010 η εταιρία λογισμικού Oracle Corporation ανακοίνωσε την εξαγορά της Sun Microsystems και των τεχνολογιών (πνευματικά δικαιώματα/ πατέντες) που η δεύτερη είχε στην κατοχή της ή δημιουργήσει. Η Java και το γενικότερο οικοσύστημα τεχνολογιών γύρω από αυτή ήρθε στον έλεγχο της Oracle Corporation.

1.4. Χαρακτηριστικά της Java

Τα βασικότερα χαρακτηριστικά της Java που την κάνουν να υπερέχει σε σύγκριση με άλλες υψηλού επιπέδου γλώσσες προγραμματισμού είναι:

- Η ανεξαρτησία της από την πλατφόρμα εκτέλεσης: Ένα πρόγραμμα Java εκτελείται σε οποιαδήποτε πλατφόρμα είτε σε Windows, ή Linux, ή Unix ή και Macintosh χωρίς να χρειάζεται να ξαναγίνει μεταγλώττιση (compiling) ή να αλλάξει ο πηγαίος κώδικας για κάθε διαφορετικό λειτουργικό σύστημα.
- Είναι αντικειμενοστραφής γλώσσα προγραμματισμού.

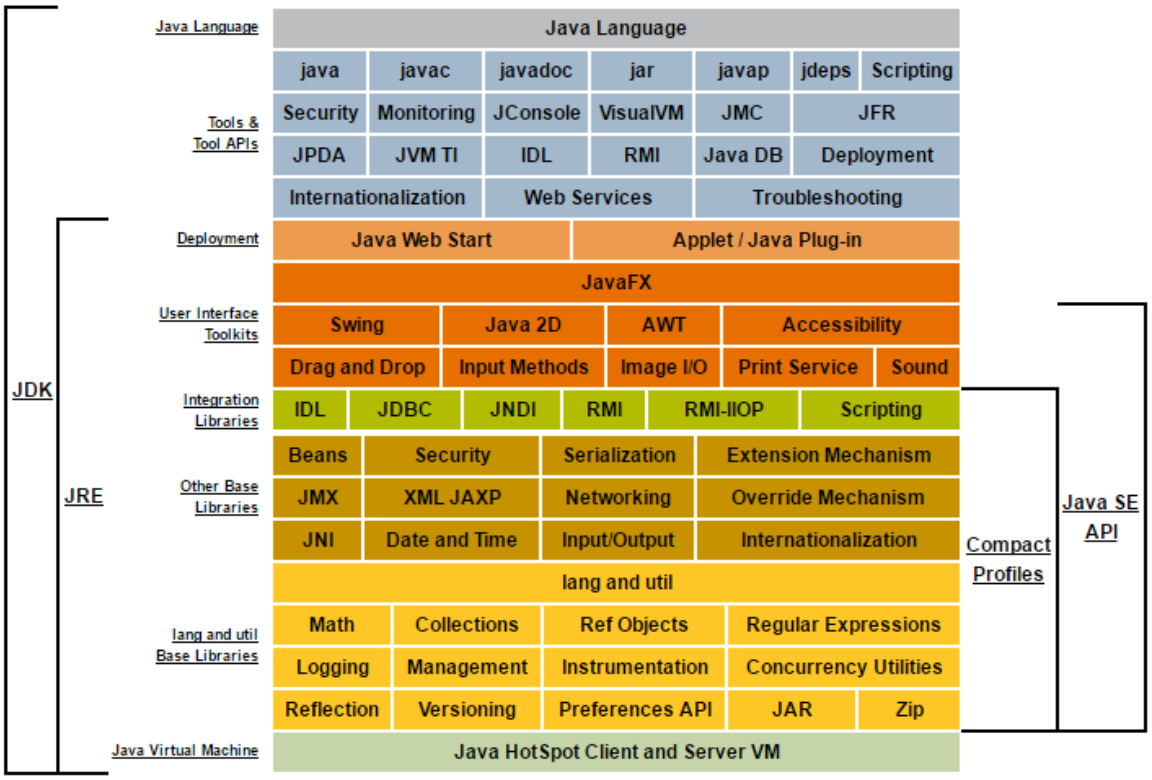
Άλλα χαρακτηριστικά της γλώσσας είναι:

- Είναι υψηλού επιπέδου γλώσσα προγραμματισμού: Η εκμάθηση της Java και η σύνταξη κώδικα είναι σχετικά απλή μιας και η γλώσσα κάνει χρήση λέξεων που βρίσκονται πιο κοντά στη φυσική γλώσσα.

- Παρέχει υψηλό επίπεδο ασφάλειας: Η εκτέλεση προγραμμάτων ελέγχεται από μηχανισμούς ασφαλείας που αποτρέπουν την κακόβουλου κώδικα.
- Είναι κατάλληλη για προγραμματισμό διαδικτυακών εφαρμογών.
- Υποστηρίζει πολυνηματική επεξεργασία (multi-threaded processing): η Java είναι μία από τις ελάχιστες γλώσσες της κατηγορίας της που παρέχει εγγενή υποστήριξη για την ανάπτυξη multi-threaded εφαρμογών.
- Κάνει αυτόματη διαχείριση μνήμης: Στη Java, η διαχείριση της μνήμης ελέγχεται αποκλειστικά από αυτήν μέσω ενός υποπρογράμματος που ονομάζεται garbage collector (αποκομιστής απορριμάτων) και ο προγραμματιστής δεν εμπλέκεται ποτέ στη διαδικασία δέσμευσης και απελευθέρωσης μνήμης.
- Είναι δυναμική: Η Java ενημερώνεται συνεχώς ενσωματώνοντας και υποστηρίζοντας τις τελευταίες τεχνολογικές εξελίξεις.

1.5. Αρχιτεκτονική της Γλώσσας – JDK & JRE

Για να εκτελέσουμε ένα πρόγραμμα γραμμένο σε Java σε κάποιον υπολογιστή, είναι απαραίτητο στον υπολογιστή αυτόν να είναι εγκατεστημένο τουλάχιστον το αντίστοιχο JRE (Java Runtime Environment) για τον επεξεργαστή που διαθέτει και το λειτουργικό σύστημα που χρησιμοποιεί. Στην εικόνα που ακολουθεί, διακρίνονται τα συστατικά της αρχιτεκτονικής που χρησιμοποιούν τόσο το JRE της Java, όσο και το JDK (Java Development Kit).

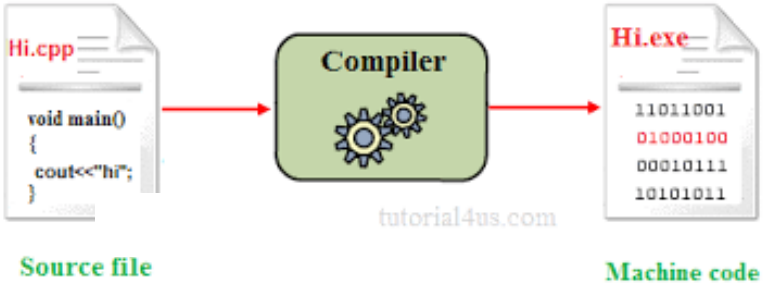


Εικόνα 1 Αρχιτεκτονική γλώσσα προγραμματισμού Java (πηγή https://untangle.data.blog/2016/10/05/java-architecture-basics/)

Το JRE αποτελεί υπερσύνολο του JDK όπως φαίνεται και στην εικόνα, και η κύρια διαφορά τους είναι ότι το JRE το χρησιμοποιεί κανείς για να **τρέξει** προγράμματα Java, ενώ το JDK για να μπορέσει να προγραμματίσει (και να τρέξει φυσικά) προγράμματα Java.

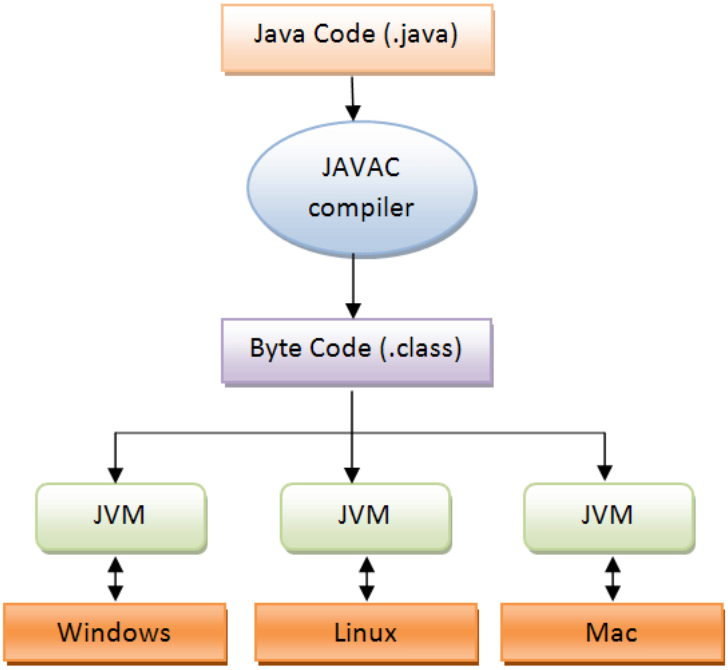
1.6. Το Java Virtual Machine (JVM)

Οι περισσότερες γλώσσες προγραμματισμού που χρησιμοποιούνται σήμερα (τρίτης γενιάς) π.χ. C++, C, κ.α. ανήκουν στην κατηγορία των υψηλών επιπέδου γλωσσών προγραμματισμού. Οι γλώσσες αυτές μεταγλωττίζουν τον πηγαίο κώδικά τους σε γλώσσα μηχανής που αποτελεί την εκτελέσιμη μορφή του κώδικα που είναι αντιληπτή από τον επεξεργαστή και το λειτουργικό σύστημα ώστε να την εκτελέσουν. Αυτή η εκτελέσιμη μορφή του πηγαίου κώδικα έχει μεταγλωττιστεί σε μια σειρά από εντολές σε δυαδική μορφή (1,0). Η εκτελέσιμη μορφή του κώδικα είναι ρητά συνδεδεμένη με την αρχιτεκτονική του επεξεργαστή και το λειτουργικό σύστημα του μηχανήματος στο οποίο πραγματοποιήθηκε η μεταγλώττιση. Η εκτελέσιμη μορφή του πηγαίου κώδικα που μεταγλωττίστηκε σε πλατφόρμα Windows δεν μπορεί να χρησιμοποιηθεί ως εκτελέσιμος κώδικας σε μια άλλη πλατφόρμα Solaris ή Linux. Κατά δημιουργία της εκτελέσιμης μορφής του κώδικα ο μεταγλωττισμένος κώδικας σε γλώσσα μηχανής συνδέεται με ένα σύνολο βιβλιοθηκών ή άλλον κώδικα που εξαρτάται αποκλειστικά από την πλατφόρμα του μηχανήματος στο οποίο πραγματοποιείται η μεταγλώττιση. Για να χρησιμοποιηθεί ο πηγαίος κώδικας και να εκτελεστεί σε άλλη πλατφόρμα απαραίτητη είναι η μεταγλώττιση του στην νέα πλατφόρμα.



Εικόνα 2 Μεταγλωτιστής C++ (Πηγή tutorial4us.com)

Αντίθετα ένα από τα βασικότερα πλεονεκτήματα της Java αποτελεί η ανεξαρτησία της από την πλατφόρμα που θα εκτελεστεί το πρόγραμμα Java. Κάθε πρόγραμμα Java μπορεί να εκτελεστεί σε οποιαδήποτε πλατφόρμα χωρίς καμία επιπλέον μεταγλώττιση. Το ίδιο πρόγραμμα Java δηλαδή μπορεί να εκτελεστεί είτε σε Windows με Intel επεξεργαστή, ή σε Solaris με Sparc επεξεργαστή είτε σε Mac OS X με Intel επεξεργαστή. Το πλεονέκτημα οφείλεται αποκλειστικά στην ύπαρξη της Java Virtual Machine (JVM). Ο πηγαίος κώδικας σε Java μεταγλωττίζεται από τον Java Compiler (μεταγλωττιστή) σε μια εξειδικευμένη κωδικοποιημένη μορφή το bytecode αντίθετα από τις άλλες γλώσσες προγραμματισμού που μεταγλωττίζονται σε γλώσσα μηχανής. Η μορφή αυτή (bytecode) αποτελεί έναν ενδιάμεσο κώδικα ο οποίος εκτελείται στην εικονική μηχανή Java Virtual Machine (JVM). Η εικονική μηχανή είναι διαφορετική για κάθε πλατφόρμα και είναι υπεύθυνη για την εκτέλεση του ενδιάμεσου κώδικα (bytecode) και τη μετατροπή του στη γλώσσα που 'καταλαβαίνει' το λειτουργικό σύστημα και ο επεξεργαστής του συγκεκριμένου μηχανήματος. Η Java ακολουθεί τον κανόνα ***"Compile once, run everywhere".***



Εικόνα 3 Java ανεξάρτητη πλατφόρμας

Προκειμένου να εκτελέσουμε ένα πρόγραμμα Java, απαραίτητη είναι η εγκατάσταση του Java Runtime Environment (JRE) που περιλαμβάνει το JVM καθώς και οι βιβλιοθήκες της Java που εξαρτώνται από την πλατφόρμα εκτέλεσης. Το JRE παρέχεται σε όλες τις σύγχρονες πλατφόρμες. Για να μεταγλωττίσουμε ένα πρόγραμμα Java θα πρέπει να διαθέτουμε τον Java Development Kit (JDK) το οποίο είναι επίσης διαφορετικό για κάθε πλατφόρμα.

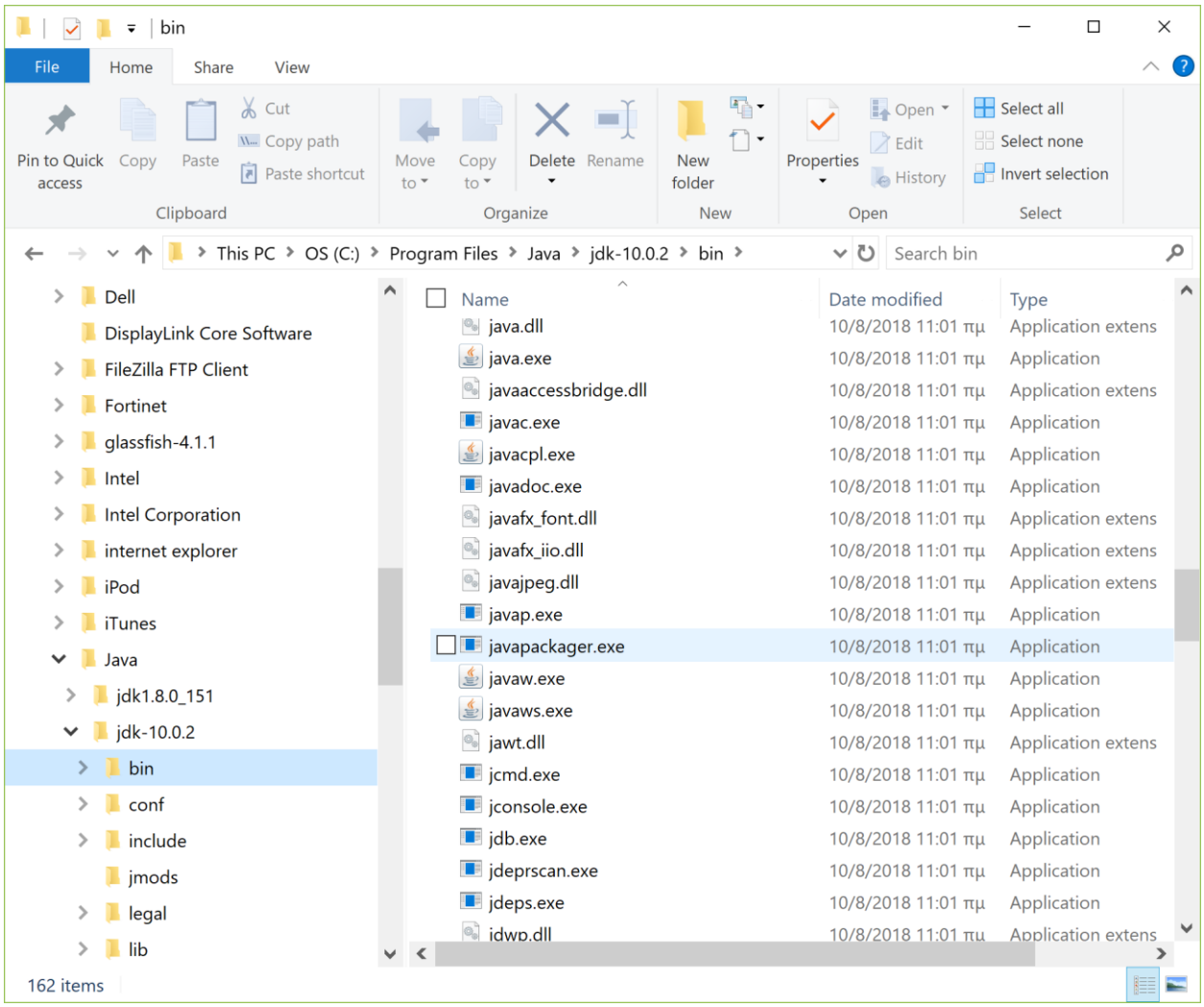
- Ο Διαδικαστικός προγραμματισμός δεν μοντελοποιεί σωστά τον πραγματικό κόσμο. Στον φυσικό κόσμο έχουμε οντότητες (αντικείμενα, ανθρώπους) που έχουν χαρακτηριστικά, και μπορούν να εκτελέσουν κάποιες δράσεις αντίθετα ο διαδικαστικός προγραμματισμός θεωρεί μόνο την ύπαρξη σταδίων.
- Είναι δύσκολο να συντηρούμε προγράμματα. Μία αλλαγή σε λογική μπορεί να απαιτεί αλλαγή σε περισσότερα βήματα της διαδικασίας
- Έχει αποδειχθεί επίσης δύσκολη η βελτίωση και αναβάθμιση των προγραμμάτων μας.

1.7. Εγκατάσταση Απαραίτητου Λογισμικού

Για την δημιουργία του περιβάλλοντος ανάπτυξης της JAVA κατεβάζουμε το Java Development Kit (JDK) από την επίσημη ιστοσελίδα της Oracle (https://www.oracle.com/technetwork/java/javase/downloads/index.html) το οποίο είναι διαφορετικό για κάθε πλατφόρμα και το εγκαθιστούμε. Μετά την εγκατάσταση δημιουργείται ο κατάλογος jdk10.0.1, όπου περιέχει:

- τον κατάλογο bin που περιλαμβάνει το σύνολο των executables. Τα πιο σημαντικά είναι:
- Java (Runtime Runtime Environment που περιλαμβάνει το JVM για την εκτέλεση των προγραμμάτων java (εκτέλεση των .class αρχείων)

- Javac (Compiler) για την μεταγλώττιση των προγραμμάτων JAVA και την δημιουργία των .class αρχείων
- Jar για την δημιουργία Java archives αρχείων (.jar files)



Εικόνα 4 Κατάλογος εγκατάστασης Java Development Kit (JDK)

- Τον κατάλογο lib που περιλαμβάνει βιβλιοθήκες της Java.

Όπως προαναφέραμε, για τους χρήστες οι οποίοι επιθυμούν μόνο να εκτελούν προγράμματα Java υπάρχει η έκδοση JRE που αντιστοιχεί στο μόνο Java Runtime Environment και περιλαμβάνει το JVM καθώς και οι βιβλιοθήκες της Java που εξαρτώνται από την πλατφόρμα εκτέλεσης. Η έκδοση jre δεν περιλαμβάνει compiler. Επίσης το jre είναι διαφορετικό για κάθε πλατφόρμα και το εγκαθιστούμε κατεβάζοντάς το από την επίσημη ιστοσελίδα της Oracle (https://java.com/en/download/).
Το σετ από τα πιο σημαντικά εργαλεία για τον προγραμματιστή (μόνο στο JDK) είναι:

- **javac**: Ο compiler της Java
- **java**: Ο διερμηνέας της Java
- **javadoc**: Το εργαλείο που δημιουργεί HTML τεκμηρίωση σε στυλ API

- **jar**: Το εργαλείο που δημιουργεί και διαχειρίζεται αρχεία τύπου JAR
- **jdb**: Ο Java debugger
- **javap**: Disassembler αρχείων τύπου .class

1.8. Κύκλος Υλοποίησης προγράμματος

Ο κύκλος υλοποίησης ενός προγράμματος Java αρχίζει με τη σύνταξη του κώδικα από τον προγραμματιστή και την αποθήκευσή του σε ένα αρχείο πηγαίου κώδικα. Για να γράψουμε ένα απλό πρόγραμμα σε Java αρκεί να έχουμε έναν οποιοδήποτε κειμενογράφο (π.χ. notepad++) και να έχουμε εγκαταστήσει ένα πρόσφατο JDK. Το ακόλουθο είναι το διάσημο παράδειγμα HelloWorld.

HelloWorld.java

```
1  public class HelloWorld {
2      public static void main(String[] args){
3      System.out.println("Hello World!");
4      }
5  }
```

java

Εικόνα 5 HelloWorld.java

Στην Εικόνα 5 HelloWorld.java γίνεται χρήση του notepad++ για τη **σύνταξη του κώδικα** ενός απλού προγράμματος που εμφανίζει «Hello World!». Για μεγαλύτερες εφαρμογές που περιλαμβάνουν δεκάδες κλάσεις και μερικές χιλιάδες γραμμές κώδικα, θα ήταν αδύνατο να υλοποιήσουμε την εφαρμογή χωρίς την χρήση ενός Integrated Development Environment (IDE) όπως θα δούμε παρακάτω, αλλά για το παρόν παράδειγμα μας αρκεί και ένας απλός κειμενογράφος.

Το επόμενο βήμα είναι η **μεταγλώττιση** του προγράμματος Java. Για την μεταγλώττιση ενός απλού προγράμματος Java ακολουθούμε τα παρακάτω βήματα:

- Στα windows ανοίγουμε ένα command prompt παράθυρο
- Πηγαίνουμε στον κατάλογο που περιέχει τον πηγαίο κώδικα (source code) (τα .java αρχεία)
- Εκτελούμε javac και το όνομα του αρχείου .java. Π.χ.

```
javac HelloWorld.java
```

- Δημιουργείται το αρχείο HelloWorld.class που αποτελεί το bytecode αρχείο – ενδιάμεσο αρχείο εκτέλεσης που αναγνωρίζει το Java Virtual Machine.
- Εάν υπάρχουν σφάλματα αυτά εμφανίζονται στην οθόνη και ο χρήστης καλείται να προβεί σε διορθώσεις των σφαλμάτων πριν εκτελέσει το javac και πάλι.

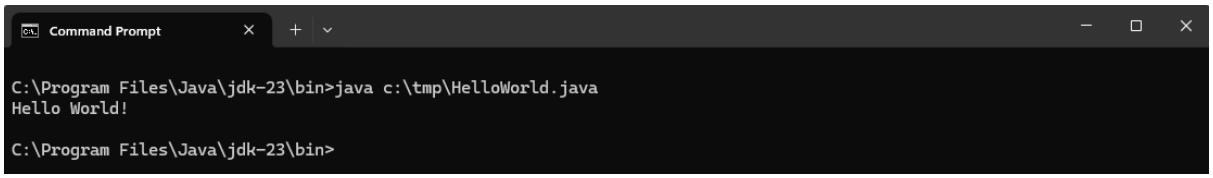
```
C:\Program Files\Java\jdk-23\bin>javac c:\tmp\HelloWorld.java
c:\tmp\HelloWorld.java:2: error: invalid method declaration; return type required
        public static main(String[] args){
                      ^
c:\tmp\HelloWorld.java:3: error: ';' expected
        System.out.println("Hello World!")
                                          ^
2 errors

C:\Program Files\Java\jdk-23\bin>
```

Το τελευταίο βήμα αφού πετύχουμε το compilation του προγράμματος χωρίς σφάλματα, είναι η εκτέλεση του Java Προγράμματος όπου και ακολουθούμε τα παρακάτω βήματα:

- Στα windows ανοίγουμε ένα command prompt παράθυρο
- Πηγαίνουμε στον κατάλογο που περιέχει το .class αρχείο
- Εκτελούμε java και το όνομα του αρχείου .class. Π.χ

```
java HelloWorld
```

- Εμφανίζεται το αποτέλεσμα «Hello World!»



1.9. Πρακτική εξάσκηση: Συγγραφή, μεταγλώττιση και εκτέλεση του πρώτου σας προγράμματος Java command line

Δοκιμάστε να υλοποιήσετε το παράδειγμα του HelloWord το οποίο περιγράφεται στην προηγούμενη ενότητα, αφού πρώτα εγκαταστήσετε όλα τα απαραίτητα λογισμικά που απαιτείται για την εκτέλεση του προγράμματος, όπως αυτά περιγράφονται στο παρόν κεφάλαιο.

1.10. Ολοκληρωμένα Περιβάλλοντα Ανάπτυξης - Integrated Development Environments (IDEs)

Φανταστείτε τώρα πως δουλεύετε σε ένα project που περιλαμβάνει δεκάδες κλάσεις και μερικές χιλιάδες γραμμές κώδικα. Είναι προφανές πως θα ήταν αδύνατο να υλοποιήσετε μια εφαρμογή ακολουθώντας τη διαδικασία που περιγράφηκε στην προηγούμενη υποενότητα, δηλαδή χρησιμοποιώντας έναν απλό κειμενογράφο και τη γραμμή εντολών. Ο απλός κειμενογράφος δε 'γνωρίζει' τη σύνταξη της γλώσσας και δε μπορεί να κάνει διαχωρισμό μεταξύ λέξεων που έχουν ειδική σημασία για τη γλώσσα και του λοιπού κώδικα.

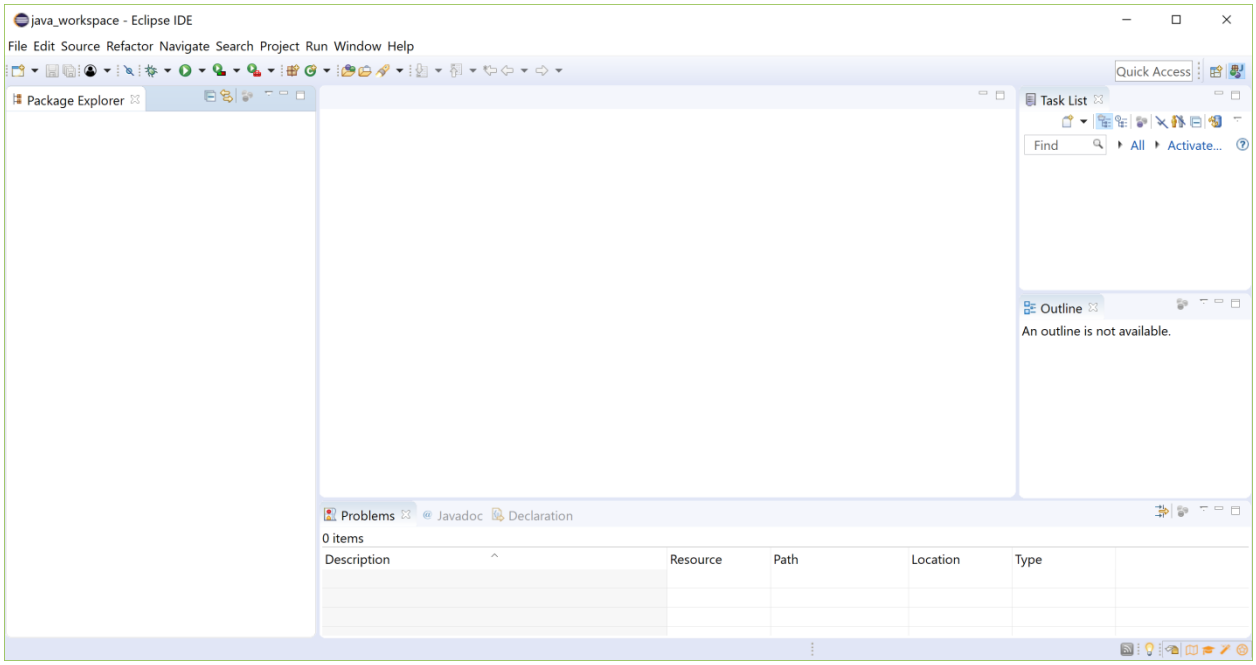
Ο προγραμματιστής θα έπρεπε να συντάξει τον κώδικα χωρίς κανένα ουσιαστικό βοήθημα από τον κειμενογράφο, στη συνέχεια να τον μεταγλωττίσει φεύγοντας από τον κειμενογράφο και

πηγαίνοντας σε κάποιο άλλο παράθυρο (αυτό του command), ενώ στην περίπτωση που υπήρχαν λάθη, θα έπρεπε να τα εντοπίσει και να τα διορθώσει ελέγχοντας μία μία τις λέξεις του κώδικα που βρίσκονται κοντά στη γραμμή που του υπέδειξε ο compiler. Κάτι τέτοιο είναι εντελώς αντιπαραγωγικό και φυσικά, για αρκετές δεκαετίες τώρα η ανάπτυξη εφαρμογών γίνεται με τη βοήθεια εξειδικευμένων εφαρμογών που ονομάζονται Ενιαία Περιβάλλοντα Υλοποίησης (Integrated Development Environments) ή απλά IDEs.

Τα IDEs διευκολύνουν τους προγραμματιστές αφού προσφέρουν μια μεγάλη ποικιλία βοηθημάτων όσο σε βασικό όσο και σε πιο προχωρημένο επίπεδο. Προσφέρουν κειμενογράφο που 'γνωρίζει' τις λέξεις που έχουν ειδική σημασία για τη γλώσσα προγραμματισμού, προβάλλει με ξεχωριστό χρώμα τα διάφορα στοιχεία του κώδικα (δεσμευμένες λέξεις, σχόλια, αλφαριθμητικά), μαρκάρει τις γραμμές που του υποδεικνύει ο compiler πως υπάρχουν σφάλματα, και προσφέρει αυτόματη υποβοήθηση κατά την σύνταξη του κώδικα όπως παρέχοντας λίστες με την σύνταξη μεθόδων της γλώσσας. Το πιο ισχυρό χαρακτηριστικό τους όμως είναι η δυνατότητα που δίνουν στον προγραμματιστή να εκτελεί όλα τα βήματα του κύκλου υλοποίησης μέσα από ένα και μόνο περιβάλλον, αυτό του IDE. Η σύνταξη, η μεταγλώττιση η εκτέλεση και αποσφαλματοποίηση του κώδικα (debugging) πραγματοποιούνται μέσα από το ίδιο περιβάλλον και όπως είναι φυσικό αυτό αυξάνει την παραγωγικότητα των ομάδων υλοποίησης.

Τα πιο δημοφιλή και ποιοτικά IDEs για την ανάπτυξη Java εφαρμογών είναι το Eclipse (http://www.eclipse.org/) και το NetBeans (https://netbeans.org/). Και τα δύο αυτά IDEs ανήκουν στη κατηγορία open source και αποτελούν και τα δύο αξιόλογα εργαλεία ανάπτυξης. Στα πλαίσια αυτού του μαθήματος θα χρησιμοποιηθεί το IDE Eclipse. Διατίθεται τόσο για την πλατφόρμα των Windows όσο και για άλλες πλατφόρμες όπως το Linux και το Mac. Το eclipse είναι διαθέσιμο στον ιστότοπο http://www.eclipse.org/downloads/ .

Το eclipse διαθέτει ένα σύνολο από διαφορετικά πακέτα. Στα πλαίσιο αυτού του κεφαλαίου θα χρησιμοποιήσουμε το «Eclipse IDE for Java Developers».

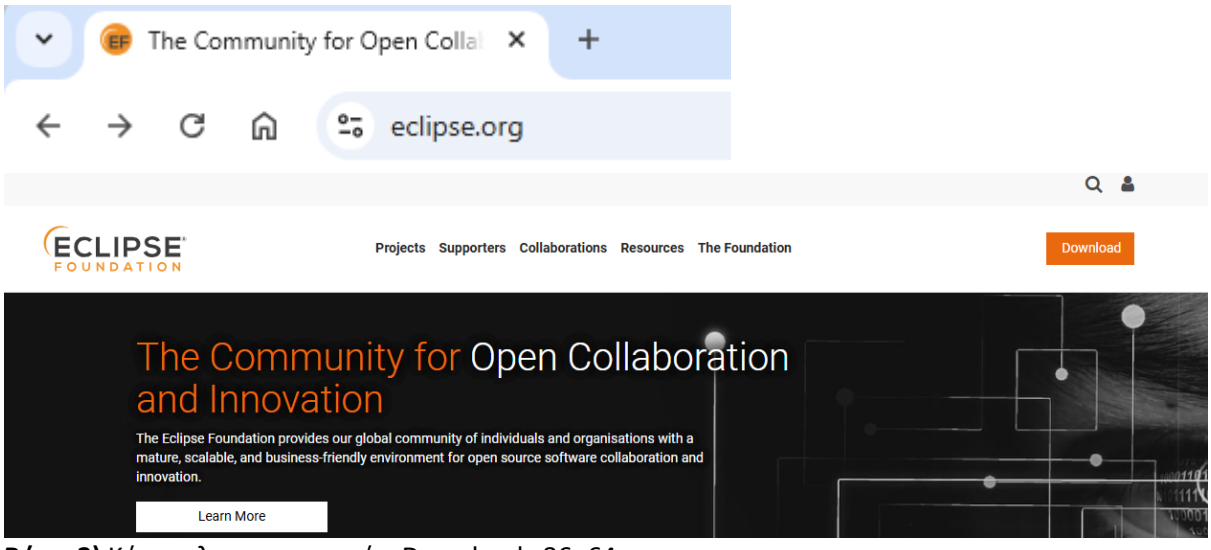


Εικόνα 6 Eclipse IDE for Java Developers

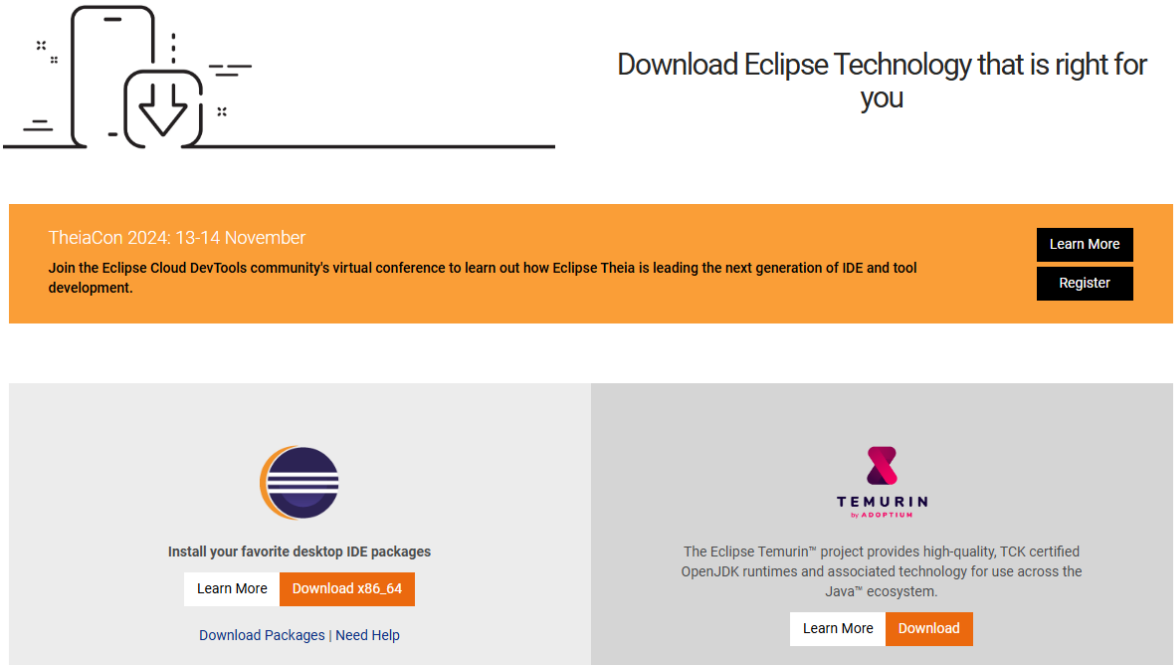
Συμβουλή: Παρόλο που θα χρησιμοποιείτε κάποιο από τα δύο IDEs που σας προτάθηκαν για την ανάπτυξη των εφαρμογών σας και που αυτοματοποιούν τη διαδικασία μεταγλώττισης, θα πρέπει οπωσδήποτε να γνωρίζετε για το διαγώνισμα της Oracle τη διαδικασία χρήσης των javac και java.

1.11. Εγκατάσταση Eclipse

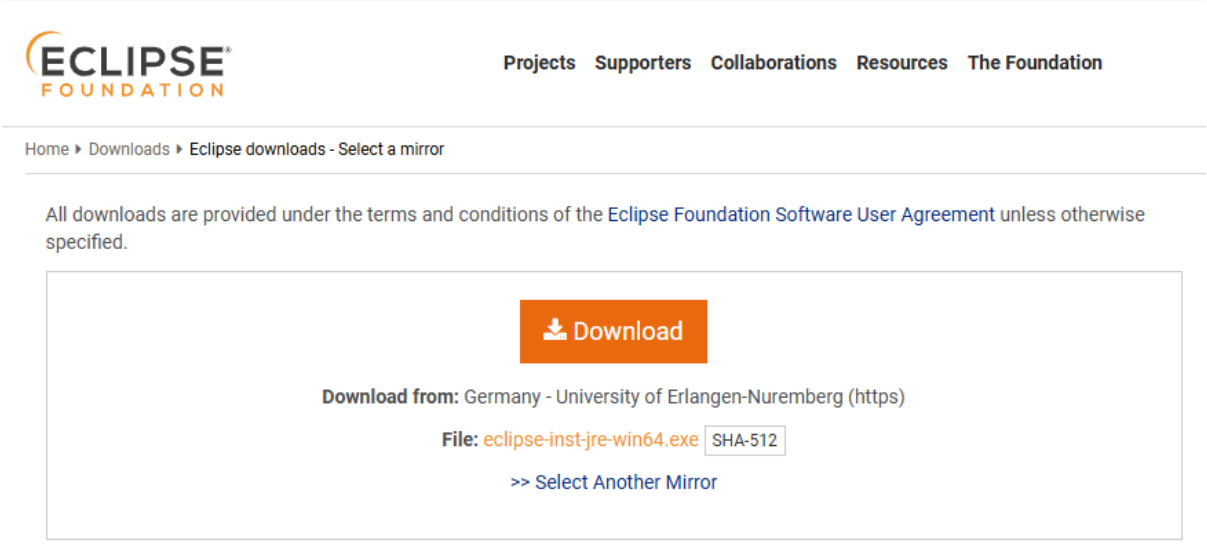
Βήμα 1) Ανοίξτε το πρόγραμμα περιήγησής σας και πληκτρολογήστε https://www.eclipse.org/



Βήμα 2) Κάντε κλικ στο κουμπί « Download x86_64».



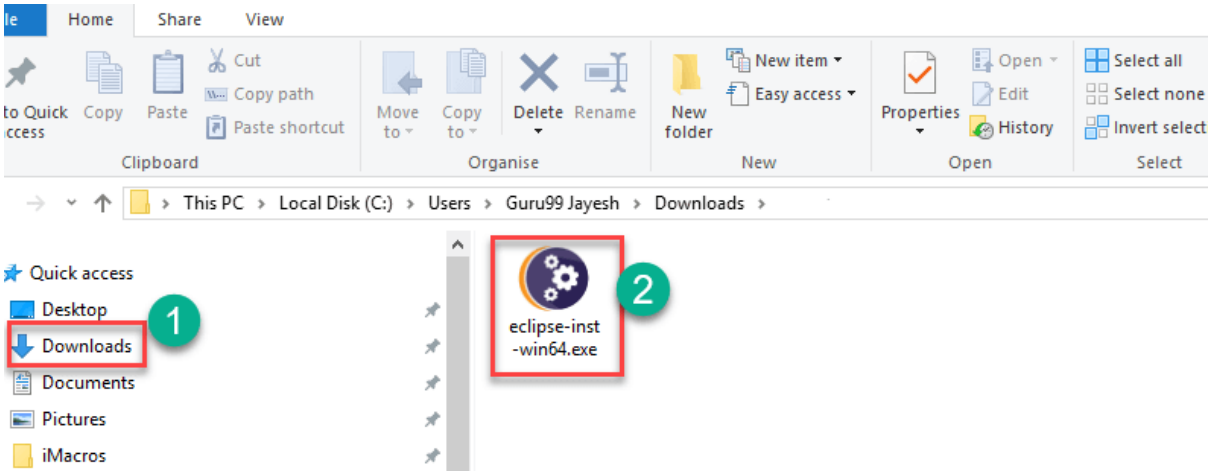
Βήμα 3) Κάντε κλικ στο κουμπί "Download".



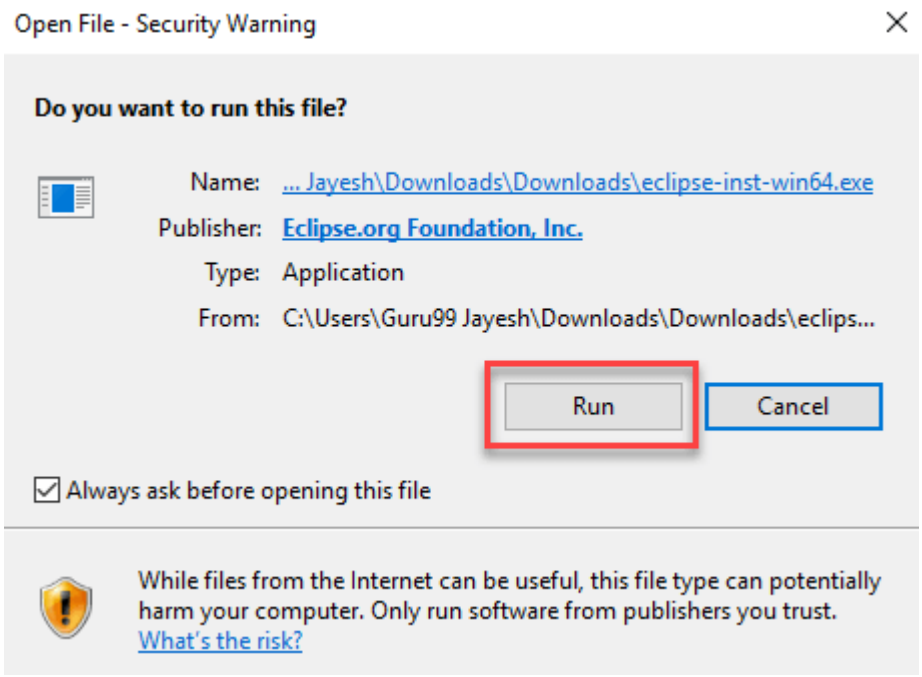
Βήμα 4) εγκαταστήστε Eclipse.

Κάντε κλικ στο «λήψεις» στο Windows εξερευνητής αρχείων.

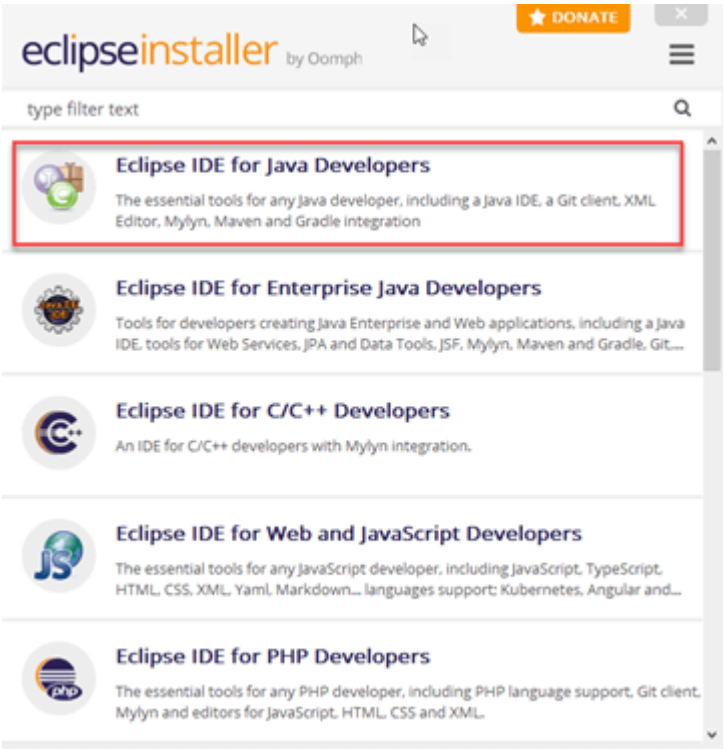
Κάντε κλικ στο αρχείο "eclipse-inst-jre-win64.exe".



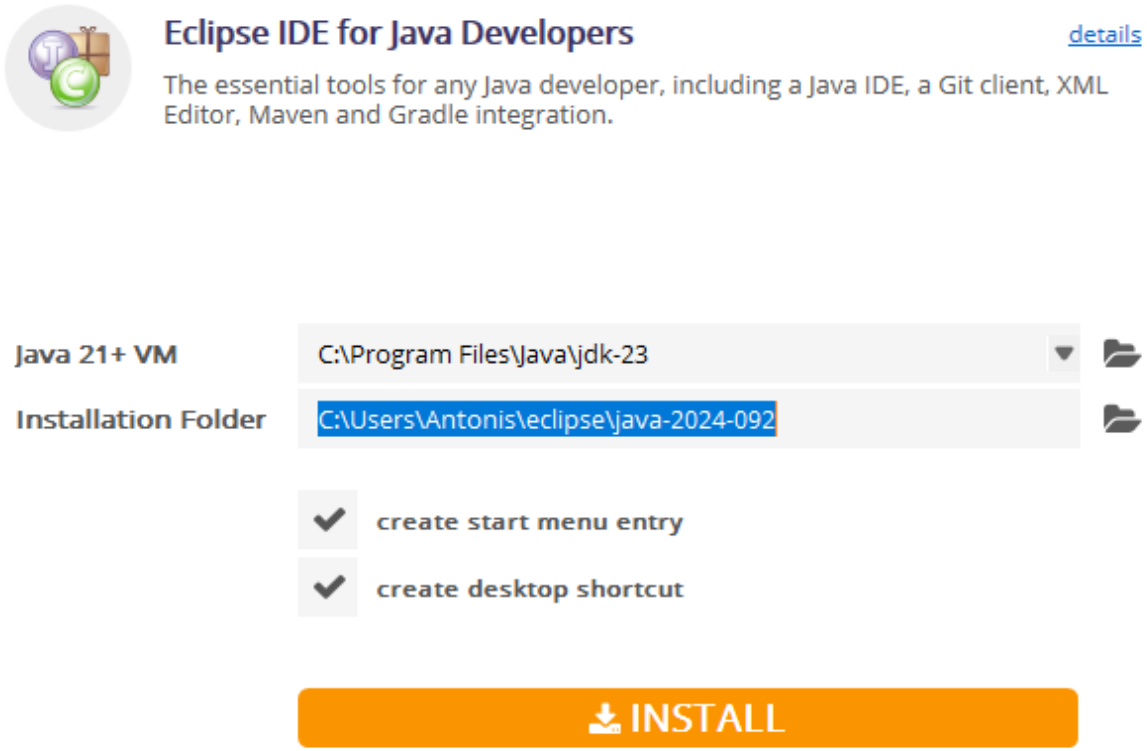
Βήμα 5) Κάντε κλικ στο κουμπί Εκτέλεση



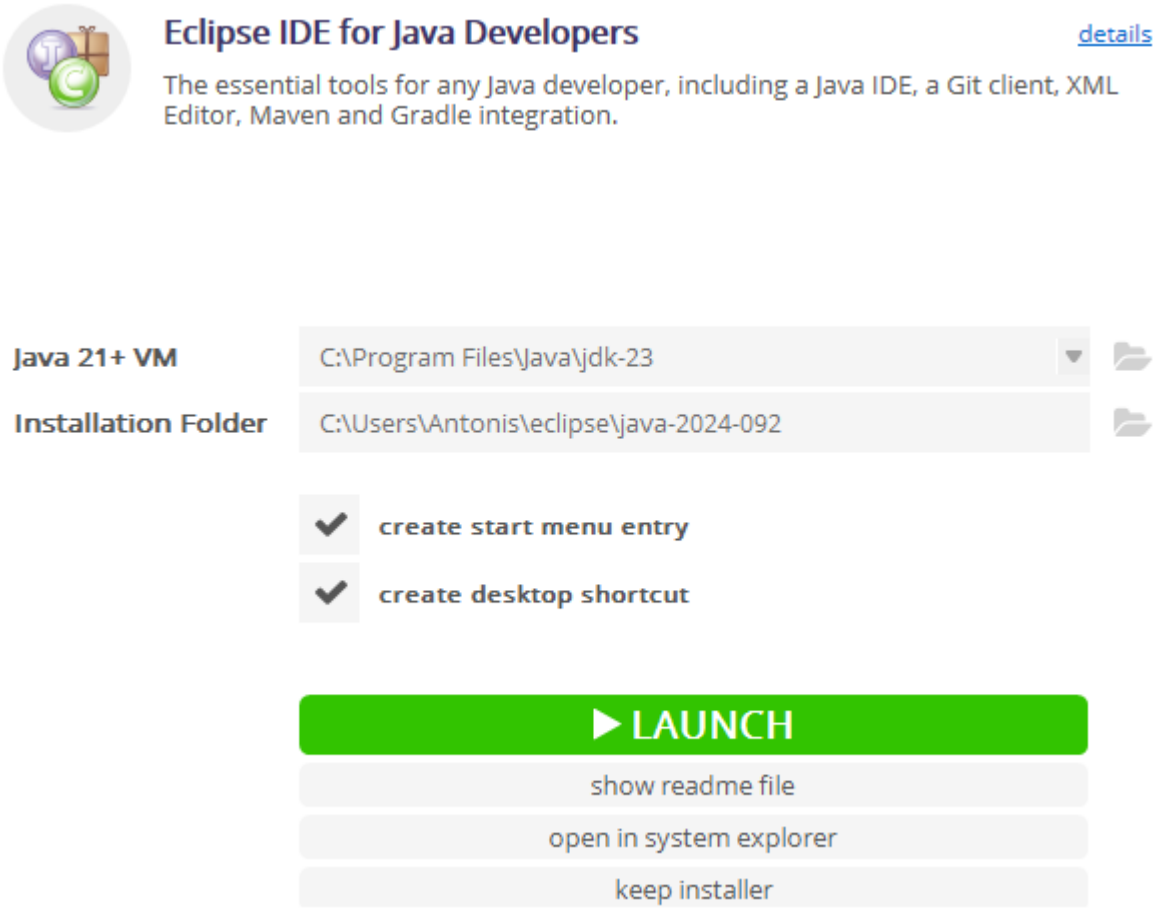
Βήμα 6) Κάντε κλικ στο "Eclipse IDE for Java Developers»



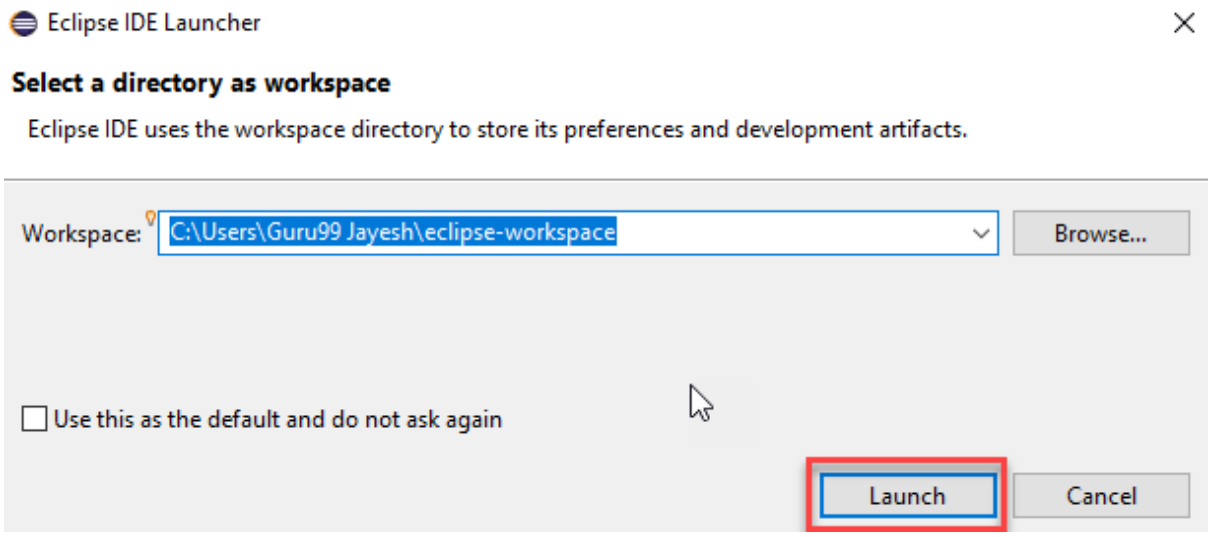
Βήμα 7) Κάντε κλικ στο κουμπί «Install».



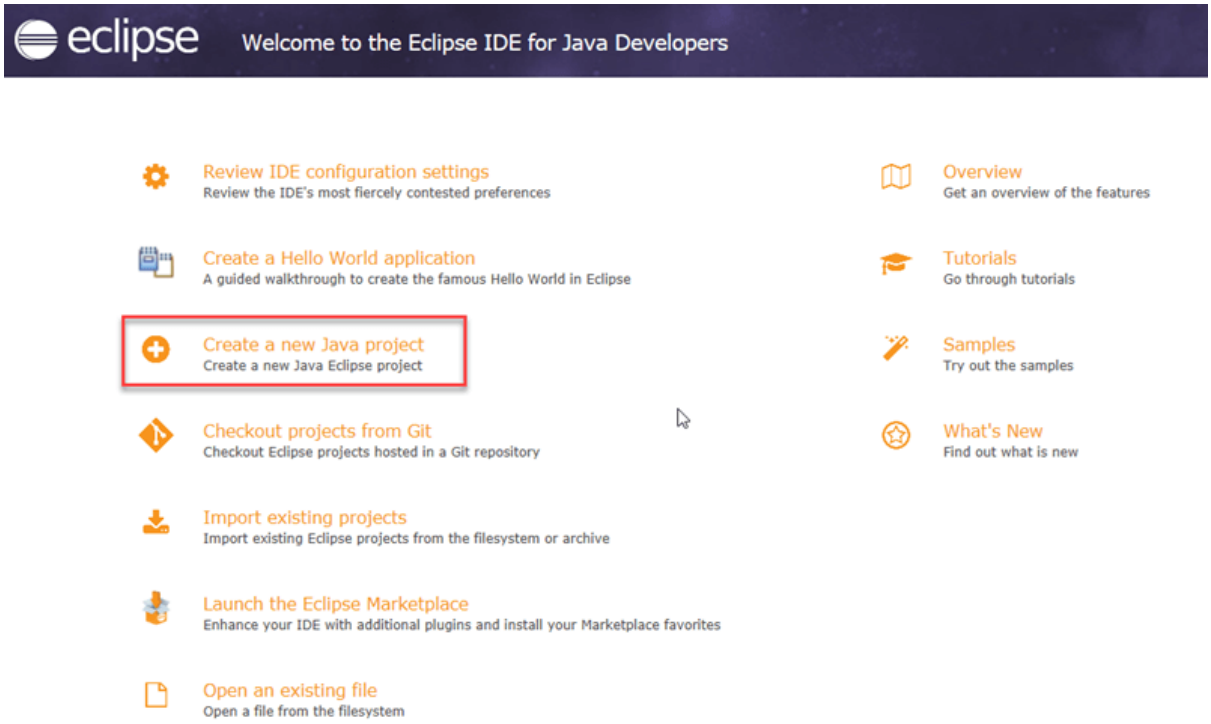
Βήμα 8) Κάντε κλικ στο κουμπί «Launch».



Βήμα 9) Κάντε κλικ στο κουμπί "Εκκίνηση".



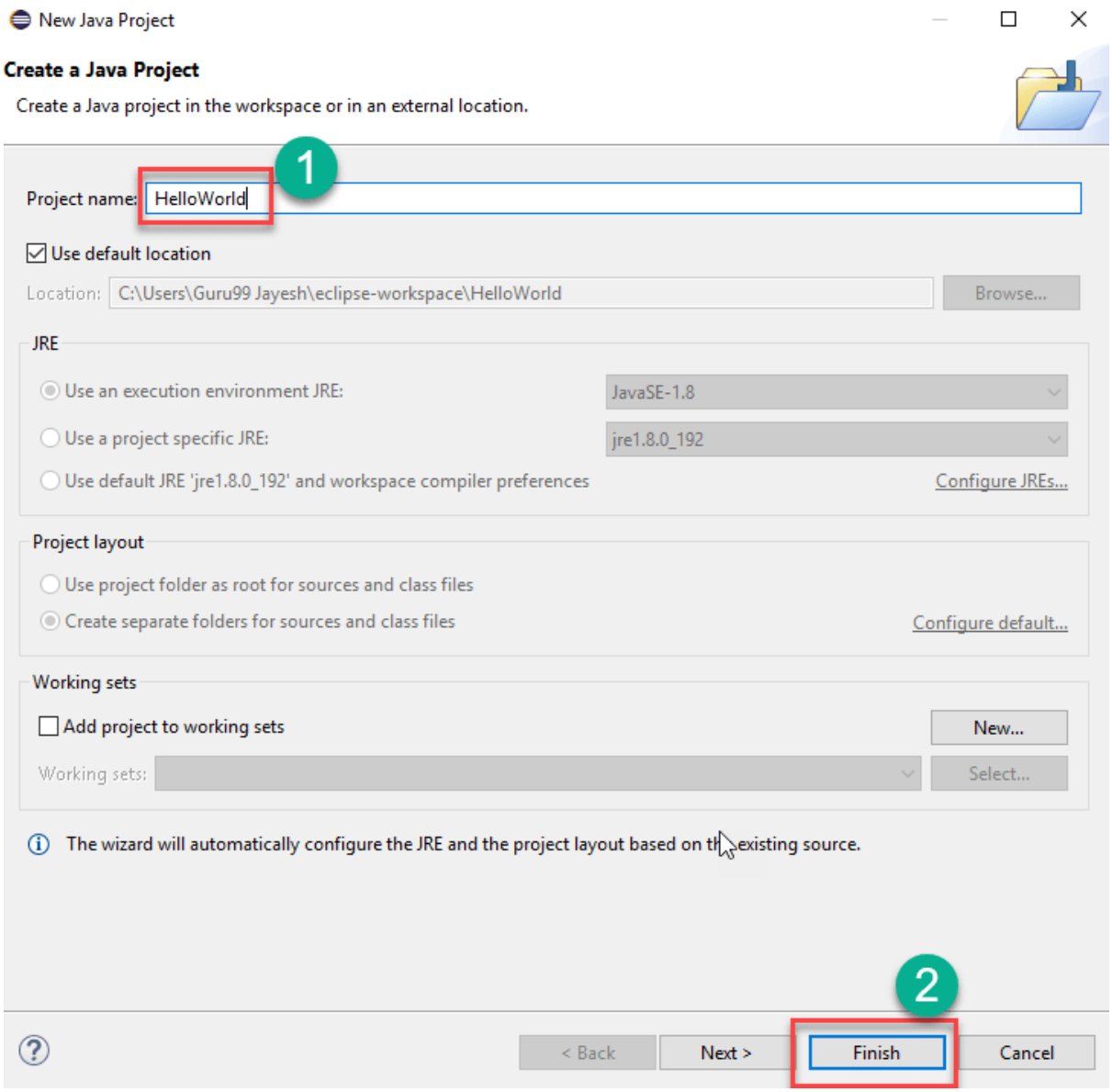
Βήμα 10) Κάντε κλικ στο «Δημιουργία νέου Java σύνδεσμος έργου».



Βήμα 11) Δημιουργήστε ένα νέο Java Σχέδιο

Γράψτε το όνομα του έργου.

Κάντε κλικ στο κουμπί "Τέλος".

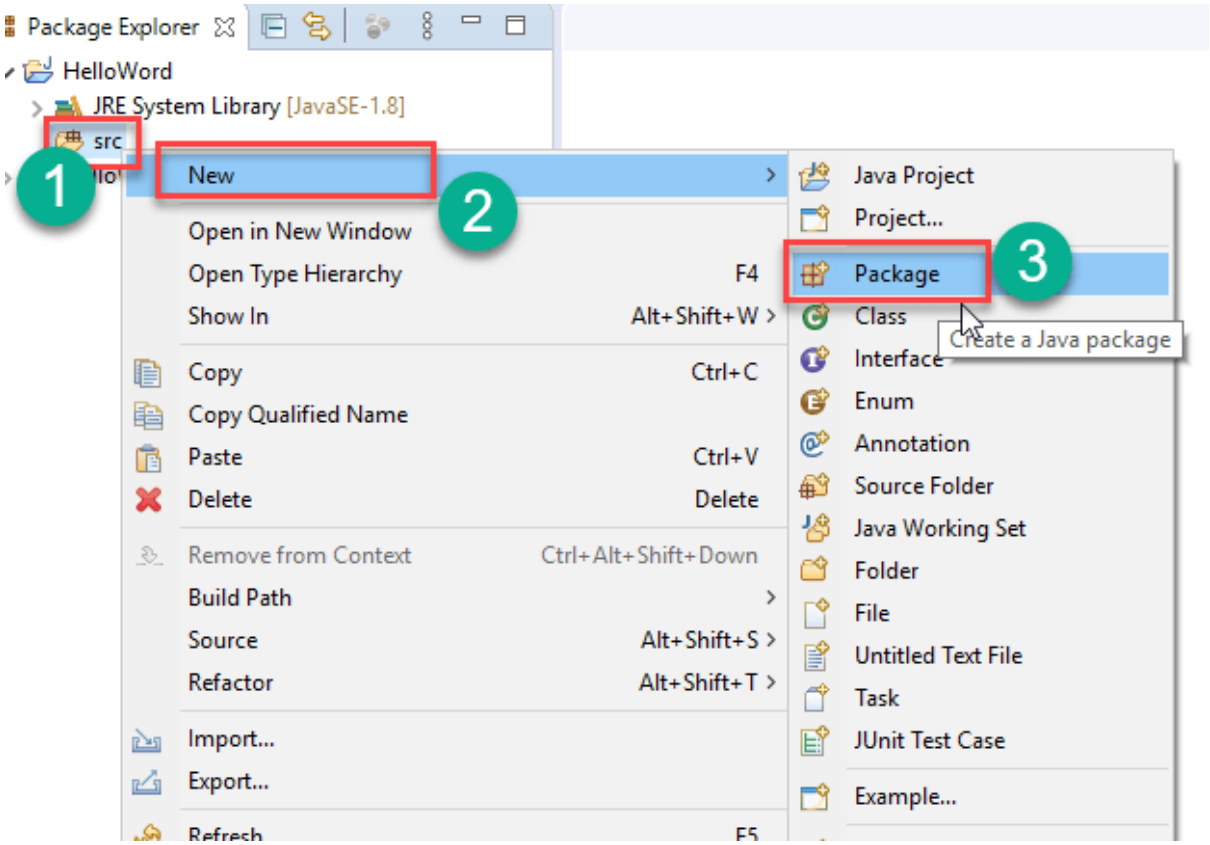


Βήμα 12) Δημιουργία Java Πακέτο.

Μεταβείτε στο "src".

Κάντε κλικ στο «Νέο».

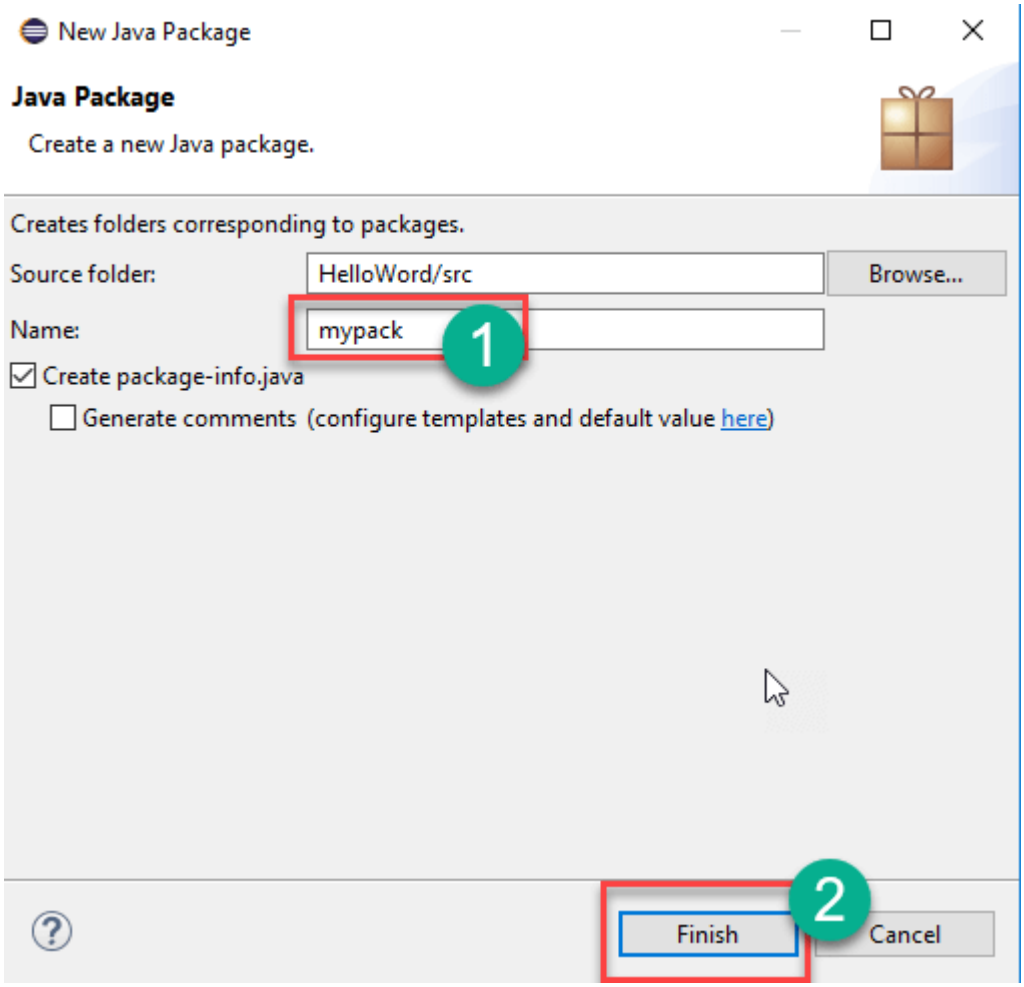
Κάντε κλικ στο «Πακέτο».



Βήμα 13) Γράψιμο ονόματος πακέτου.

Γράψτε το όνομα του πακέτου

Κάντε κλικ στο κουμπί Τέλος.

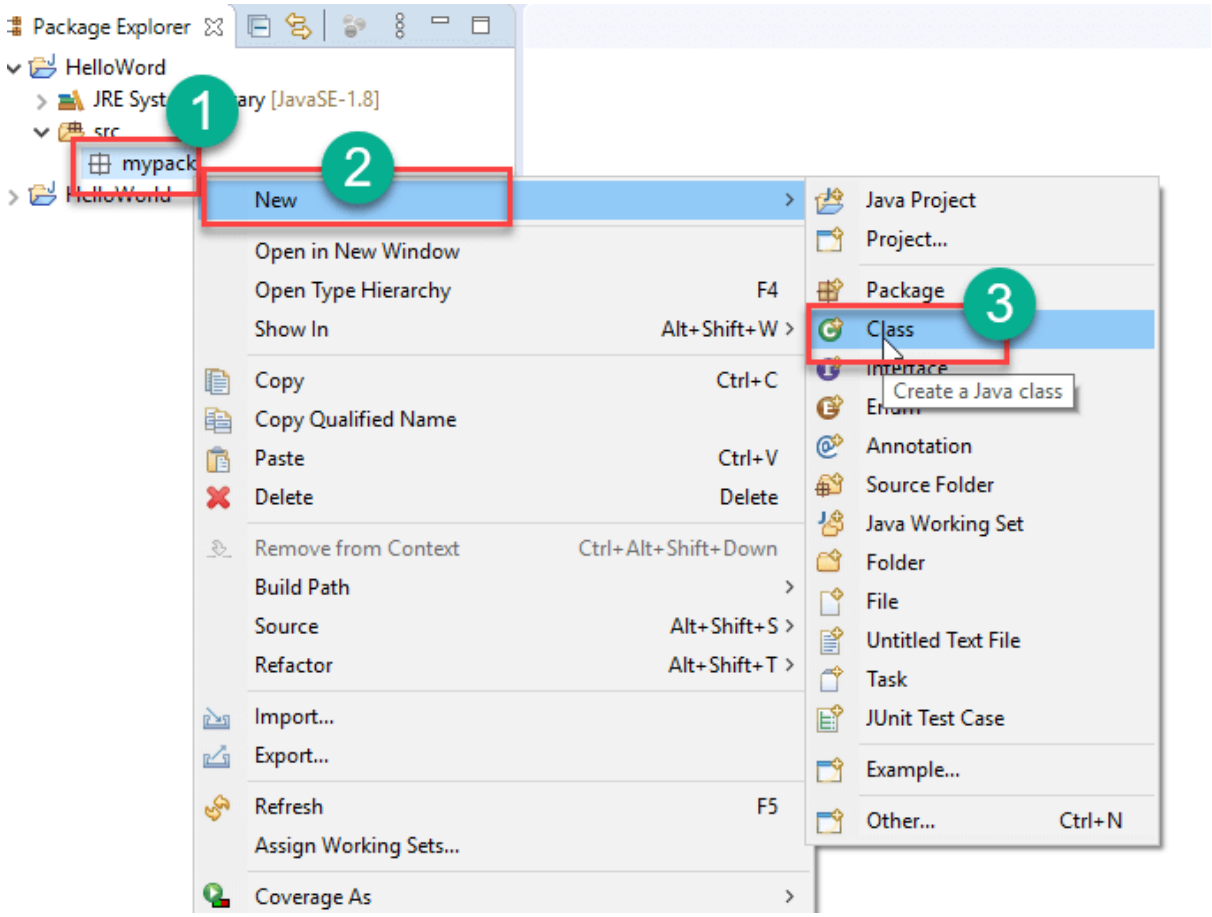


Βήμα 14) δημιουργία Java Τάξη

Κάντε κλικ στο πακέτο που δημιουργήσατε.

Κάντε κλικ στο «Νέο».

Κάντε κλικ στην «Τάξη».

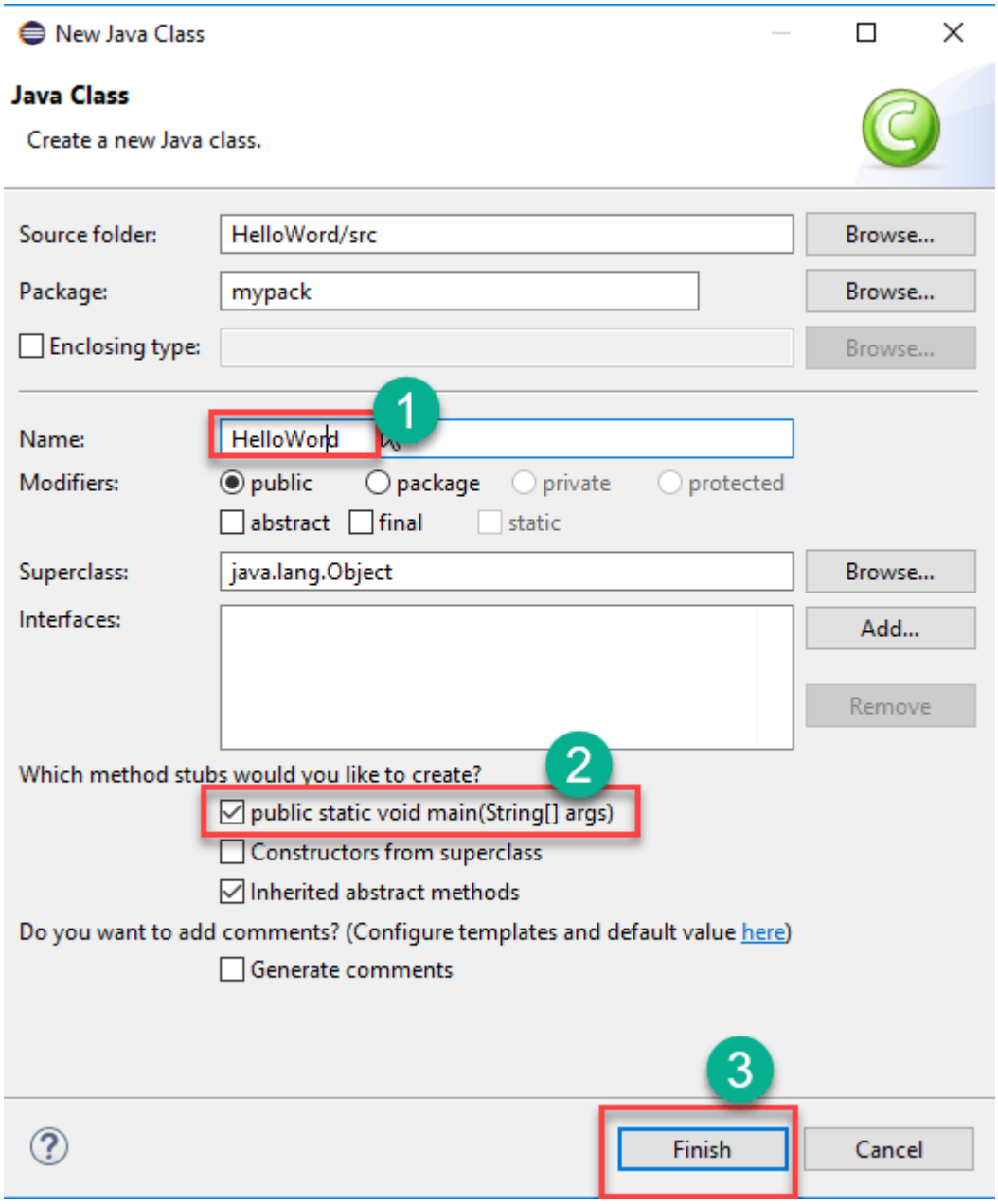


Βήμα 15) Ορισμός Java Κατηγορίας

Γράψτε το όνομα της τάξης

Κάντε κλικ στο πλαίσιο ελέγχου "δημόσιο στατικό κενό κύριο (String[] args)".

Κάντε κλικ στο κουμπί "Τέλος".

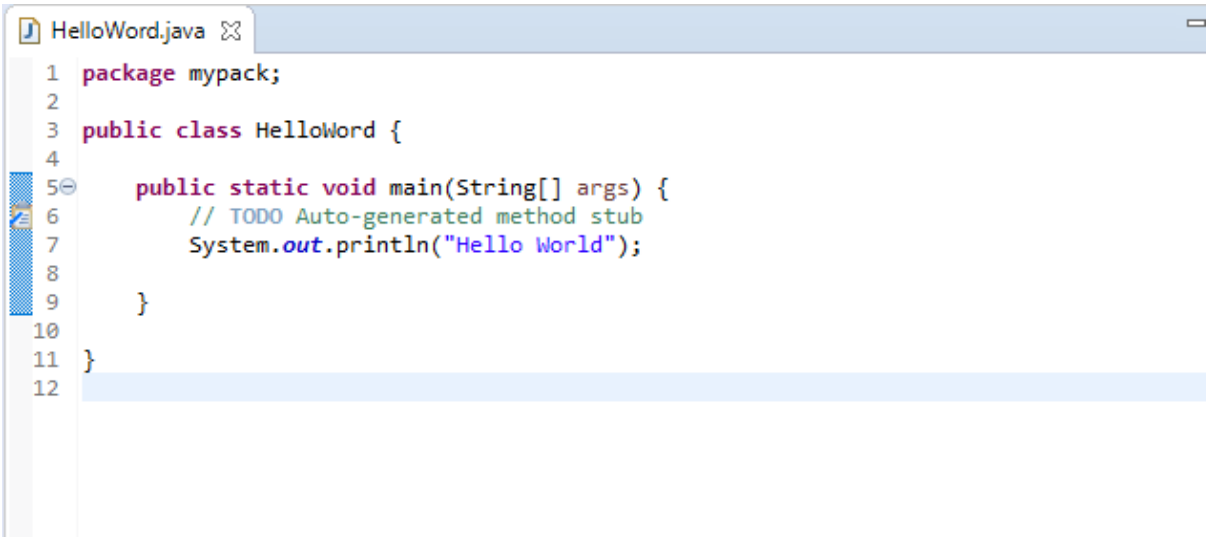


Το αρχείο Helloword.java θα δημιουργηθεί όπως φαίνεται παρακάτω:

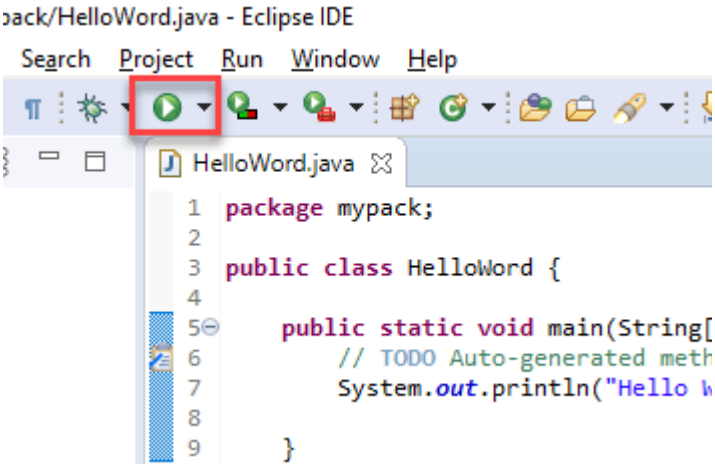
ΣΧΕΤΙΚΑ ΑΡΘΡΑ

Τι είναι το JVM; Java Εικονική μηχανή Archiδομή

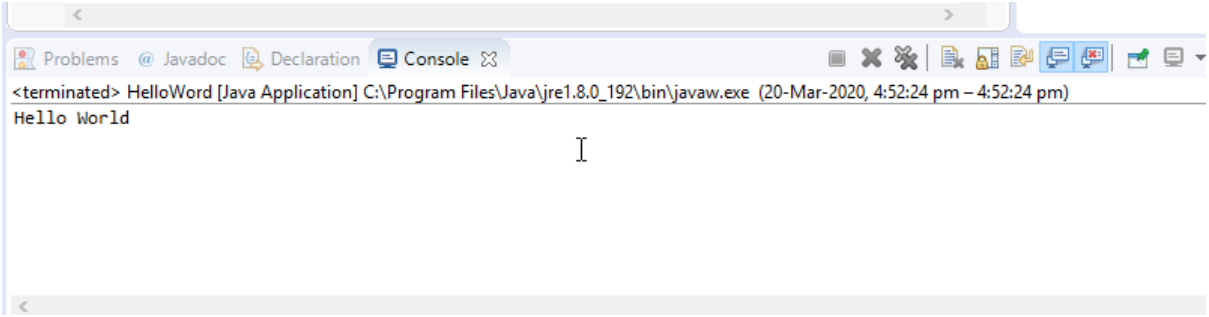
Spring Tutorial: Τι είναι το Spring Framework και πώς να εγκαταστήσετε;



Βήμα 16) Κάντε κλικ στο κουμπί "Εκτέλεση".



Η έξοδος θα εμφανιστεί όπως φαίνεται παρακάτω.



ΚΕΦΑΛΑΙΟ 2: Βασικές έννοιες προγραμματισμού

2.1. Εκπαιδευτικοί στόχοι του παρόντος κεφαλαίου

Οι εκπαιδευτικοί στόχοι του παρόντος κεφαλαίου, συνοψίζονται στα κάτωθι σημεία. Οι εκπαιδευόμενοι:

- θα διδαχθούν τη δομή της Java κλάσης,
- θα γνωρίσουν τη Java Main Κλάση,
- θα μάθουν τη κεντρική μέθοδος main,
- θα μάθουν την απλή Έξοδος στην Κονσόλα,
- θα μάθουν τον τρόπο χρήσης των μεταβλητών, και των τύπων Δεδομένων,
- θα γνωρίσουν τις κυριολεκτικές τιμές (LITERALS),
- θα μάθουν πως γίνεται η δήλωση και Αρχικοποίηση Μεταβλητών,
- θα μάθουν πως γίνεται η ανάθεση (Assignment,
- θα γνωρίσουν τους τύπους δεδομένων της Java,
- θα μάθουν τους τελεστές στην Java,
- θα γνωρίσουν τις μετατροπές τύπων (Promotion και Casting),
- θα μάθουν πως υλοποιείται η δομή ελέγχου (Conditional statement) if/else και οι εμφωλευμένες if,
- θα μάθουν πως υλοποιείται η δομή ελέγχου Switch και οι σύνθετες συνθήκες,
- θα μάθουν πως υλοποιούνται οι εντολές επανάληψης While Loop, For Loop, do/while Loop,
- θα γνωρίσουν πως υλοποιούνται οι εντολές break και continue keywords,
- θα μάθουν πως υλοποιούνται οι εμφωλευμένες εντολές επανάληψης.

2.2. Η Java Κλάση

2.2.1. Δομή Αρχείων στη Java

Στη γλώσσα προγραμματισμού Java, υπάρχουν μόνο αρχεία πηγαίου κώδικα. Τα αρχεία αυτά έχουν πάντοτε κατάληξη .java, π.χ.

`MyClass.java.`

Κάθε αρχείο πηγαίου κώδικα καλό είναι να περιέχει μία μόνο κλάση. Το αρχείο θα πρέπει να έχει ακριβώς το ίδιο όνομα με την κλάση. Σημειώνεται ότι η Java είναι case sensitive δηλαδή κάνει διαχωρισμό μεταξύ πεζών και κεφαλαίων χαρακτήρων. Αυτό σημαίνει πως για τη Java οι λέξεις myClass και MyClass αναφέρονται σε ξεχωριστές οντότητες. Βάσει αυτού και του παραπάνω κανόνα σωστής πρακτικής, αν δημιουργήσετε ένα αρχείο πηγαίου κώδικα στο IDE που χρησιμοποιείτε και το ονομάσετε MyClass.java, θα πρέπει το όνομα της κλάσης που περιέχει να είναι το MyClass.

Γενικότερα, για την ονομασία αρχείων πηγαίου κώδικα στη Java ισχύουν οι εξής κανόνες σωστής πρακτικής:

- Τα ονόματα μπορούν να περιέχουν μόνο αλφαριθμητικούς χαρακτήρες. Περίεργα σύμβολα όπως τα * - & (^ ~ ! κλπ απαγορεύονται. Το μόνο σύμβολο που επιτρέπεται είναι το underscore (_)
- Τα ονόματα θα πρέπει πάντα να ξεκινούν με γράμμα και όχι με αριθμό.
- Τα ονόματα δε θα πρέπει να περιέχουν κενά (spaces). Αν κάποιο όνομα αποτελείται από δύο ή περισσότερες λέξεις, μπορείτε να τις διαχωρίσετε είτε με τη χρήση underscores είτε χρησιμοποιώντας κεφαλαίο χαρακτήρα στο πρώτο γράμμα κάθε λέξης.
- Τα ονόματα θα πρέπει να χρησιμοποιούν αποκλειστικά το Αγγλικό αλφάβητο.

Ένα Java project μπορεί να περιέχει πολλά αρχεία πηγαίου κώδικα. Για είναι εκτελέσιμο, θα πρέπει σε τουλάχιστον ένα από τα αρχεία του project να περιέχεται μία κεντρική μέθοδος, η οποία αναλύεται διεξοδικά σε μεταγενέστερο κεφάλαιο.

Ένα τυπικό αρχείο πηγαίου κώδικα Java λοιπόν, μπορεί να περιέχει τα εξής:

- Δήλωση πακέτου
- Εντολές import
- Ορισμό κλάσης ή interface
- Σχόλια
- Μία κεντρική μέθοδο

2.2.2. Δημιουργία Java κλάσης

Η Java κλάση αποτελεί το δομικό στοιχείο μιας Java Εφαρμογής. Μια Java εφαρμογή αποτελείται από δεκάδες κλάσεις. Για να δηλώσουμε μια κλάση χρησιμοποιούμε το keyword class και ακολουθεί το όνομα της κλάσης το οποίο γράφεται πάντα με λατινικούς χαρακτήρες και σύμφωνα με τις συμβάσεις ξεκινά πάντα με κεφαλαίο γράμμα. Επίσης αν το όνομα της κλάσης αποτελείται από 2 λέξεις όπως για παράδειγμα HelloWorld σύμφωνα με τις συμβάσεις και η δεύτερη λέξη θα ξεκινάει με κεφαλαίο γράμμα. Τα ονόματα των κλάσεων περιέχουν μόνο αλφαριθμητικούς χαρακτήρες και από ειδικά σύμβολά είναι επιτρεπτός μόνο ο χαρακτήρας underscore (_). Επιπλέον τα ονόματα θα πρέπει πάντα να ξεκινούν με γράμμα και όχι με αριθμό και δεν περιέχουν κενά (spaces).

Παράδειγμα σύνταξης κλάσης:

```
public class HelloWorld {
	//χαρακτηριστικά (fields) της κλάσης

	//μέθοδοι (methods) της κλάσης
}
```

Το keyword public ονομάζεται *modifier* και δηλώνει ότι η κλάση HelloWorld μπορεί να χρησιμοποιηθεί από οποιονδήποτε. Η Java είναι case-sensitive δηλαδή το HelloWorld είναι διαφορετικό από το helloworld. Μια κλάση περιέχει fields που αποτελούν τα χαρακτηριστικά της κλάσης και μεθόδους (methods) που ορίζουν την συμπεριφορά της κλάσης. Το σώμα της κλάσης περιλαμβάνεται εντός των αγκίστρων **{ }.** Κάθε κλάση αποθηκεύεται σε ένα αρχείο με όνομα το όνομα της κλάσης και την κατάληξη .java. Στο παραπάνω παράδειγμα το όνομα του αρχείου που περιλαμβάνει την κλάση είναι HelloWorld.java.

2.2.3. Δημιουργία Java Packages

Το package (πακέτο) αποτελεί έναν τρόπο οργάνωσης του κώδικά μας. Πρόκειται για ένα ιδιαίτερα σημαντικό κομμάτι της γλώσσας μιας και παρέχει στους προγραμματιστές έναν μηχανισμό ιεραρχικής οργάνωσης των κλάσεων. Ένα package περιλαμβάνει ένα σύνολο κλάσεων οι οποίες με την χρήση του package ομαδοποιούνται. Το package με Κλάσεις έχει την έννοια του Καταλόγου με τα Αρχεία του. Όταν δημιουργούμε μια κλάση δεν είναι υποχρεωτική η χρήση του package όμως συστήνεται η χρήση του. Για να δηλώσουμε ένα package γράφουμε το keyword package και κατόπιν το όνομα του package που ξεκινά πάντα με μικρό γράμμα. Σύμφωνα με τις συμβάσεις το όνομα του package πρέπει να είναι με μικρά γράμματα. Στο τέλος εισάγουμε το semicolon **;** που αποτελεί τον χαρακτήρα τέλους κάθε εντολής στην java. Για παράδειγμα:

```
package hello;
public class HelloWorld {
      //χαρακτηριστικά και μέθοδοι της κλάσης
}
```

Η κλάση HelloWorld ανήκει στο package hello και είναι η μόνη με το όνομα HelloWorld στο package hello. Απευθυνόμαστε στην κλάση HelloWorld ως **hello.HelloWorld.**

Καθώς ακόμη και η ίδια η Java έχει τις κλάσεις τις οργανωμένες σε πακέτα, αυτά περιέχονται στο πακέτα με το όνομα java και ως εκ τούτου είναι προσπελάσιμες με την ονοματολογία.

```
java.<package_name>
```

Αν θέλουμε να χρησιμοποιήσουμε στον κώδικά μας μία κλάση που περιέχεται σε κάποιο πακέτο, θα πρέπει να πληκτρολογήσουμε το πλήρες όνομά της (fully qualified name) ώστε ο compiler να μπορέσει να την εντοπίσει. Για παράδειγμα η εντολή

```
java.<package_name>.<class_name>
```

υποδεικνύει στον compiler να δημιουργήσει ένα αντικείμενο της κλάσης `class_name`που περιέχεται στο πακέτο `package_name`, το οποίο με τη σειρά του περιέχεται στο πακέτο java που βρίσκεται στην κορυφή της. ιεραρχίας. Η αναφορά χρησιμοποιώντας το πλήρες όνομα δεν είναι απαραίτητη μόνο στην περίπτωση που οι δύο κλάσεις βρίσκονται στο ίδιο πακέτο και άρα ‘βλέπουν’ η μία την άλλη.

Είναι προφανές πως η χρήση του πλήρους ονόματος απαιτεί αρκετή πληκτρολόγηση από πλευράς προγραμματιστή και για την καταπολέμηση του συγκεκριμένου προβλήματος η ομάδα της Java δημιούργησε τη λύση του import. Κάνοντας χρήση της δεσμευμένης λέξης import σε συνδυασμό με το πλήρες όνομα της κλάσης που θέλουμε να χρησιμοποιήσουμε στον κώδικά μας δε χρειάζεται

πλέον να αναφερόμαστε σε αυτήν με το πλήρες όνομα και αρκεί να χρησιμοποιήσουμε το απλό όνομά της. Το προηγούμενο παράδειγμα δηλαδή, κάνοντας χρήση του import θα μετατρεπόταν ως εξής:

```
import java.util.Stack;
...
Stack s = new Stack();
```

Ένας άλλος τρόπος χρήσης του import είναι σε συνδυασμό με το χαρακτήρα μπαλαντέρ (wildcard) όπως φαίνεται στην έκφραση που ακολουθεί, υποδεικνύοντας στον διερμηνέα να συμπεριλάβει τα ονόματα όλων των κλάσεων και interfaces που περιέχονται στο πακέτο `java.<package_name>` (όχι όμως τα ονόματα των πακέτων που μπορεί να περιέχονται σε αυτό):

```
import java.util.*;
```

Η Java μας δίνει τη δυνατότητα να φτιάξουμε δικά μας πακέτα χρησιμοποιώντας τη δεσμευμένη λέξη package σε συνδυασμό με ένα όνομα της επιλογής μας. Για παράδειγμα η εντολή

```
package myPackage;
```

υποδεικνύει στον compiler να τοποθετήσει την κλάση που περιέχεται στο συγκεκριμένο αρχείο πηγαίου κώδικα μέσα στο πακέτο `myPackage`. Αντίστοιχα, η εντολή

```
package myPackage.mySubPackage;
```

τοποθετεί την κλάση που περιέχεται στο συγκεκριμένο αρχείο μέσα στο πακέτο `mySubPackage`, το οποίο με τη σειρά του περιέχεται στο πακέτο `myPackage` που βρίσκεται στην κορυφή της ιεραρχίας.

Σε ένα αρχείο πηγαίου κώδικα, η εντολή ορισμού πακέτου (package) τοποθετείται πάντοτε στην πρώτη γραμμή και στη συνέχεια ακολουθούν τα import statements. Τέλος, ακολουθεί η δήλωση της κλάσης, όπως φαίνεται στον κώδικα που ακολουθεί:

```
package myPackage.mySubPackage;
import java.util.*;
public class myClass {
...
}
```

2.2.4. Java Main Κλάση

Η main είναι μια ειδική μέθοδος η οποία αναγνωρίζεται από την Java Virtual Machine (JVM) και αποτελεί το αρχικό σημείο εκτέλεσης ενός Java προγράμματος. Επομένως για να είναι ένα πρόγραμμα εκτελέσιμο, θα πρέπει οπωσδήποτε σε κάποιο από τα αρχεία πηγαίου κώδικα (.java) από τα οποία αποτελείται να υπάρχει τουλάχιστον μία κεντρική μέθοδος - main method. Δηλαδή θα πρέπει κάποια κλάση να περιέχει την μέθοδο main. Η κλάση αυτή ονομάζεται Main Class. Με τον τρόπο αυτό όταν δώσουμε στο διερμηνέα της Java την εντολή να εκτελέσει το πρόγραμμά μας, ο διερμηνέα θα εκκινήσει να εκτελεί πρώτα από όλα τις εντολές που υπάρχουν στην κλάση main.

Το συντακτικό της main είναι πάντα το ίδιο και είναι αυτό που φαίνεται στο παράδειγμα που ακολουθεί:

```
public static void main(String[] args)  {
        //οι εντολές που περιλαμβάνονται στο σώμα της main

}
```

Το JVM θα ξεκινήσει να εκτελεί τις εντολές που περιέχονται στη main αρχίζοντας από αυτήν που βρίσκεται αμέσως μετά το άγκιστρο έναρξης και καταλήγει αφού εκτελέσει και αυτήν που βρίσκεται αμέσως πριν το άγκιστρο τερματισμού. Οι εντολές εκτελούνται η μία μετά την άλλη με τη σειρά που είναι γραμμένες (σειριακά).

Για παράδειγμα:

```
package hello;
public class HelloWorld {
    public static void main(String[] args)  {
      System.out.println("Hello World!");
       System.out.println("-The End-");
    }
}
```

Ως **εντολή** ορίζουμε στη Java μία 'έκφραση' που ξεκινάει από την αρχή μιας γραμμής και τερματίζει με το χαρακτήρα **;** (semicolon). Ακολουθώντας τις ισχύουσες καλές πρακτικές προγραμματισμού, θα πρέπει στα προγράμματά μας να έχουμε **μία εντολή ανά γραμμή**. Μία εντολή θα μπορούσε να είναι η δήλωση μιας μεταβλητής, η κλήση μιας μεθόδου, μία αριθμητική πράξη κλπ.

Μία ομάδα εντολών που περικλείονται σε ένα ζεύγος αγκίστρων ({ }) ονομάζεται **μπλοκ εντολών**. Στο παραπάνω παράδειγμα στην main περιλαμβάνονται δύο εντολές οι οποίες τυπώνουν στην οθόνη εξόδου. Η πρώτη εντολή System.out.println("Hello World!") τυπώνει ***Hello World!*** και η δεύτερη System.out.println("-The End-"); τυπώνει ***-The End-*** .

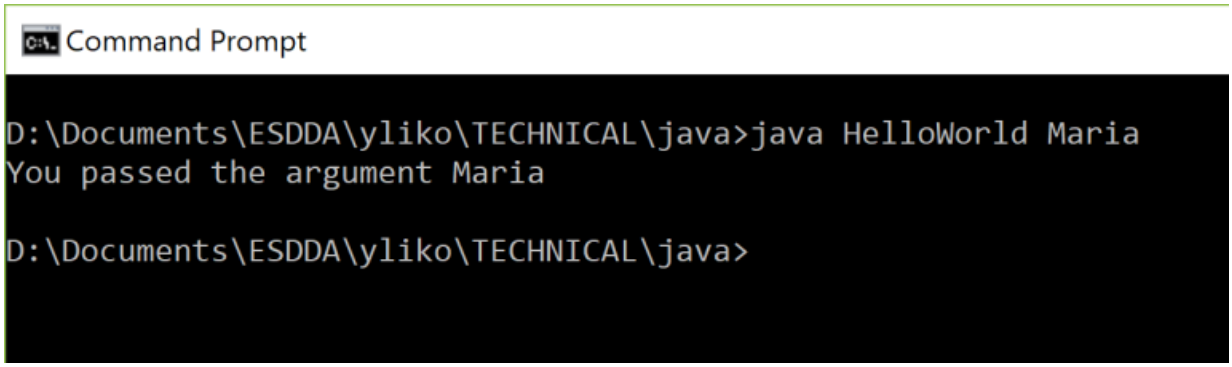
Ένα ακόμη χαρακτηριστικό της main μεθόδου είναι η δυνατότητα που παρέχει να περνάμε παραμέτρους (command line arguments) κατά την εντολή εκτέλεσης του προγράμματος, οι οποίες μπορεί να χρησιμοποιηθούν από τον κώδικα για την επίτευξη συγκεκριμένης λειτουργικότητας. Για παράδειγμα:

```
public class HelloWorld {
    public static void main(String[] args)  {
            System.out.println("You passed the argument "+args[0]);
     }
}
```

Στον παραπάνω κώδικα, για να εκτελεστεί σωστά το πρόγραμμα θα πρέπει κατά την εκτέλεση να περάσουμε μία παράμετρο την οποία στη συνέχεια θα χρησιμοποιηθεί στην

```
System.out.println("You passed the argument "+args[0]);
```

και θα μας προβάλλει το μήνυμα You passed the argument [παράμετρος], όπως εμφανίζεται στην παρακάτω εικόνα:



Εικόνα 7 Εκτέλεση προγράμματος με παράμετρο

2.2.5. Απλή Έξοδος στην Κονσόλα

Μία από τις πιο κοινές λειτουργίες των προγραμμάτων είναι η προβολή μηνυμάτων ή αποτελεσμάτων στο χρήστη. Για εφαρμογές γραμμής εντολών, η συγκεκριμένη λειτουργία επιτυγχάνεται με τη χρήση του αντικειμένου System.out που είναι το βασικό αντικείμενο εξόδου για τη Java. Η χρήση του είναι πολύ απλή μιας και δεν απαιτείται η εισαγωγή κάποιου πακέτου και στην πιο απλή μορφή της εμφανίζει μέσω της κλήσης μιας εκ των println ή print κάποιο αλφαριθμητικό στην κονσόλα.

Η println εμφανίζει το αλφαριθμητικό που της περνάμε ως παράμετρο στην κονσόλα και κατεβαίνει στην επόμενη γραμμή ενώ η print έχει ακριβώς την ίδια λειτουργία χωρίς όμως να κατέβει στην επόμενη γραμμή. Διαδοχικές κλήσεις της print δηλαδή εξακολουθούν να εμφανίζουν τα μηνύματα στην ίδια γραμμή μέχρις ότου ο προγραμματιστής εισάγει έναν ειδικό χαρακτήρα που θα πληροφορήσει τον compiler να κατέβει στην επόμενη γραμμή (\n) ή καλέσει την println.

Χρησιμοποιώντας τις μεθόδους αυτές σε συνδυασμό με τον τελεστή + μπορούμε εκτός από αλφαριθμητικά να εμφανίζουμε στην κονσόλα και τιμές μεταβλητών. Ο ακόλουθος κώδικας για παράδειγμα θα εμφανίσει στην κονσόλα το μήνυμα x = 5.

```
int z = 13;
System.out.println("Η τιμή είναι ή " + z);
```

2.2.6. Σχόλια (Comments)

Η χρήση σχολίων στον κώδικα προγραμματισμού είναι ζωτικής σημασίας για διάφορους λόγους. Πρώτον, τα σχόλια βοηθούν στην κατανόηση του κώδικα τόσο από τον ίδιο τον προγραμματιστή όσο και από άλλους που μπορεί να χρειαστεί να διαβάσουν ή να συντηρήσουν τον κώδικα στο μέλλον. Όταν γράφουμε κώδικα, μπορεί να φαίνεται ξεκάθαρος εκείνη τη στιγμή, αλλά με την πάροδο του χρόνου, οι λεπτομέρειες μπορεί να ξεχαστούν. Τα σχόλια παρέχουν μια εξήγηση για το τι κάνει κάθε τμήμα του κώδικα, διευκολύνοντας την κατανόηση και την αναθεώρηση. Επιπλέον, σε ομάδες ανάπτυξης, τα σχόλια είναι απαραίτητα για τη συνεργασία, καθώς επιτρέπουν σε όλους τους προγραμματιστές να κατανοήσουν τη λογική και τη λειτουργικότητα του κώδικα, μειώνοντας τον χρόνο που απαιτείται για την εκμάθηση και την προσαρμογή. Είναι απαραίτητο στον κώδικα που γράφουμε να εισάγουμε σχόλια αφού παρέχουν σημαντική βοήθεια σε οποιονδήποτε τρίτο διαβάσει τον κώδικα μας ώστε ο κώδικας να είναι κατανοητός

Δεύτερον, τα σχόλια βοηθούν στην ανίχνευση και την επίλυση σφαλμάτων. Όταν ο κώδικας δεν λειτουργεί όπως αναμένεται, τα σχόλια μπορούν να παρέχουν ενδείξεις για το πού μπορεί να βρίσκεται το πρόβλημα. Περιγράφοντας τη λογική πίσω από συγκεκριμένα τμήματα του κώδικα, τα σχόλια επιτρέπουν στους προγραμματιστές να εντοπίζουν πιο εύκολα τις αποκλίσεις από την αναμενόμενη λειτουργία. Επιπλέον, τα σχόλια μπορούν να χρησιμοποιηθούν για να σημειώσουν περιοχές του κώδικα που χρειάζονται βελτιώσεις ή περαιτέρω ανάπτυξη, λειτουργώντας ως υπενθυμίσεις για μελλοντικές εργασίες. Συνολικά, τα σχόλια βελτιώνουν την ποιότητα του κώδικα, καθιστώντας τον πιο ευανάγνωστο, συντηρήσιμο και λιγότερο επιρρεπή σε σφάλματα.

Σε ένα πρόγραμμα Java για να δηλώσουμε σχόλια χρησιμοποιούμε δύο τρόπους. Ο πρώτος τρόπος που ονομάζεται σχόλια γραμμής ξεκινά με την ακολουθία χαρακτήρων **//** που σημαίνει οτιδήποτε μετά από αυτούς τους χαρακτήρες μεχρι το τέλος της γραμμής είναι σχόλιο. Παράδειγμα:

```
// αυτό είναι ένα σχόλιο μέχρι το τέλος της γραμμής
```

Στον δεύτερο τρόπο τα σχόλια ξεκινούν με τους χαρακτήρες **/*** και τερματίζουν με την ακολουθία χαρακτήρων ***/** Για παράδειγμα

```
/*αυτό είναι ένα σχόλιο */
είτε

/* αυτό είναι ένα σχόλιο που
περιλαμβάνει
περισσότερες γραμμές
*/

είτε
/* αυτό είναι ένα σχόλιο που
 * περιλαμβάνει
 * περισσότερες γραμμές
 */
```

Οτιδήποτε βρίσκεται μεταξύ των χαρακτήρων έναρξης /* και τερματισμού των σχολίων */ ή του σχολίου γραμμής // αγνοείται παντελώς από τον compiler, πράγμα που σημαίνει πως μπορούμε να γράψουμε οτιδήποτε θελήσουμε στα σχόλια στα Αγγλικά, στα Ελληνικά ή στα Greeklish κλπ.

Τέλος, με τη χρήση ειδικών σχολίων (/** */) και ειδικών tags που ονομάζονται annotations και σε συνδυασμό με το εργαλείο javadoc του JDK είναι δυνατόν να παράξουμε ένα HTML reference manual για τον κώδικά μας, στο στυλ του Java API reference manual. Η συγκεκριμένη λειτουργία δεν συμπεριλαμάνεται στην ύλη του διαγωνίσματος της Oracle και σας παρουσιάζεται απλά ενημερωτικά. Στο παράδειγμα που ακολουθεί γίνεται χρήση τέτοιων annotations εντός των σχολίων.

```
/**
* This class represents a jpg image
*
```

```
* @author User User
* @version 3.0, February 19, 2024
* @see server
*/
```

2.2.7. Πρακτική εξάσκηση έξοδος στην Κονσόλα

Δημιουργήστε μία κλάση που αθροίζει τις τιμές 14 και 2 και εν συνεχεία τυπώνει το άθροισμα αυτό στην κονσόλα.

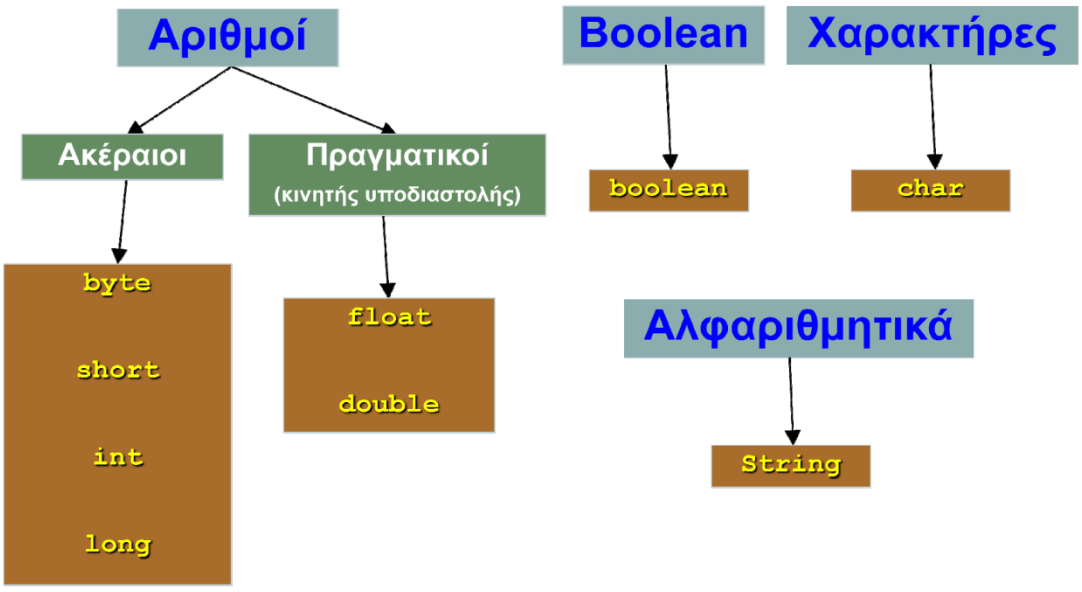
2.3. Μεταβλητές, Τύποι Δεδομένων

2.3.1. Τύποι δεδομένων

Στην Java υπάρχουν 8 primitive τύποι δεδομένων (data types) τέσσερα (4) υποσύνολα των ακεραίων (integers), δύο (2) υποσύνολα αριθμών κινητής υποδιαστολής, ο χαρακτήρας – character και ο boolean τύπος δεδομένων. Οι υπόλοιποι τύποι δεδομένων αναπαριστώνται χρησιμοποιώντας αντικείμενα.

Τύπος	**Περιγραφή**	**Μέγεθος**	**Εύρος**
byte	Ακέραιο τύπου byte	1 byte	-128 .. 127
short	Μικρός ακέραιος αριθμός	2 bytes	-32,768...32,767
int	Ακέραιος αριθμός	4 bytes	-2,147,483,648 ... 2,147,483,647
long	Μεγάλος ακέραιος αριθμός	8 bytes	-9,223,372,036,854,775,808.. 9,223,372,036,854,775,807
float	Κινητής υποδιαστολής απλής ακρίβειας	4 bytes	$1.4 * e^{-45}$ .. $3.4 * e^{38}$
double	Κινητής υποδιαστολής διπλής ακρίβειας	8 bytes	$4.9 * e^{-324}$ .. $1.8 * e^{308}$
boolean	Λογική τιμή	1 byte	true/false
char	Χαρακτήρας	2 bytes	-

Οι τύποι δεδομένων byte, short, int, long προορίζονται για χρήση με ακέραιους αριθμούς, ενώ οι τύποι float, double για χρήση αριθμών κινητής υποδιαστολής (δηλ. δεκαδικών αριθμών). Ο τύπος boolean μπορεί να λαμβάνει μόνο δύο τιμές true και false. Ο τύπος char έχει μέγεθος 2 bytes γιατί έχει σχεδιαστεί να περιέχει χαρακτήρες Unicode (UTF-16 για την ακρίβεια).



Εικόνα 8 Τύποι Δεδομένων (Πηγή: Εργαστήριο Τεχνολογίας Πολυμέσων ΕΜΠ)

```
int  num = 10;       Σωστό
int num = 10.20;  Compilation Error

double weight = 50.75;      Σωστό
double weight = 20;            Σωστό, θα μεταφραστεί ως 20.0
```

Οι χαρακτήρες δηλώνονται με τον τύπο δεδομένων char. Μπορούν να περιλαμβάνουν οποιοδήποτε χαρακτήρα γράμμα, ψηφίο, σημείο στίξης κλπ. Επιπλέον των χαρακτήρων υπάρχουν και οι ειδικοί χαρακτήρες που συνήθως χρησιμοποιούνται στις print και println για να προβάλουμε στην έξοδο κενές γραμμές ή τον χαρακτήρα tab ή άλλους ειδικούς χαρακτήρες. Αυτοί οι ειδικοί χαρακτήρες λέγονται ακολουθίες διαφυγής (escape sequences) και συνοψίζονται στον παρακάτω πίνακα:

Ακολουθία Διαφυγής	**Χαρακτήρας**
\n	Newline - Νέα γραμμή
\t	Tab
\b	backspace
\"	" Διπλά εισαγωγικά
\'	‘ Μονό εισαγωγικό
****	\ back slash

Επομένως αν θέλουμε να εμφανίσουμε στην έξοδο την λέξη "**Hello**" θα πρέπει να γράψουμε:

```
System.out.println("\"Hello\"");
Έξοδος:
"Hello"
```

```
public class Flowers {
        //-----------------------------------------------------------------
        // Prints a poem (of sorts) on multiple lines.
        //-----------------------------------------------------------------
        public static void main (String[] args) {
                System.out.println ("Τα τριαντάφυλλα είναι κόκκινα,\n\tΟι
Βιολέτες είναι μπλε,\n" +
                     "Η ζαχαρη είναι γλυκιά!");
        }
}
```

Έξοδος:

```
Τα τριαντάφυλλα είναι κόκκινα,
                Οι Βιολέτες είναι μπλε,
Η ζαχαρη είναι γλυκιά!
```

2.3.2. Κυριολεκτικές τιμές (LITERALS)

Με τον όρο κυριολεκτική τιμή (literal) αναφερόμαστε στην αναπαράσταση μιας τιμής κάποιου από τους βασικούς τύπους ή αλφαριθμητικού, όπως αυτή είναι γραμμένη μέσα στο πρόγραμμα.

Για παράδειγμα στην παρακάτω γραμμή,

```
x = 5;
```

το 5 είναι μία ακέραια κυριολεκτική τιμή. Άλλα παραδείγματα κυριολεκτικών τιμών είναι:

- `214 // τύπου int`
- `"I love Java" // τύπου αλφαριθμιτικού String`
- `-934.235 // τύπου double`
- `'z' // χαρακτήρας - char`
- `true // τύπου boolean`
- `5.2545F //τύπου float`
- `4834782463L // τύπου long`

2.3.3. Μεταβλητές

Μία μεταβλητή είναι το μέσο που μας επιτρέπει να αποθηκεύουμε προσωρινά στη μνήμη του υπολογιστή δεδομένα. Ουσιαστικά η μεταβλητή αντιστοιχεί σε μία θέση μνήμης η οποία καταλαμβάνει μια τιμή. Αυτή η τιμή μπορεί να αλλάξει. Μια μεταβλητή Μπορεί να παίρνει διαφορετικές τιμές κατά τη διάρκεια του προγράμματος. Μια μεταβλητή κρατά πάντα ένα συγκεκριμένο τύπο δεδομένων. Κάθε μεταβλητή έχει τέσσερα βασικά χαρακτηριστικά:

- τον τύπο (type),
- το όνομα (name ή identifier),
- την τιμή (value)
- και την διάρκεια ζωής (scope).

Και τα τέσσερα αυτά χαρακτηριστικά παίζουν σημαντικό ρόλο στη λειτουργία της κάθε μεταβλητής και καθορίζονται από τον προγραμματιστή.

Για να ορίσουμε μια μεταβλητή ακολουθούμε την σύνταξη:

```
τύπος_μεταβλητής όνομα_μεταβλητής;
```

Ο **τύπος_μεταβλητής** αντιστοιχεί στον τύπο δεδομένων που ανήκει η μεταβλητή και **όνομα_μεταβλητής** είναι ακριβώς το όνομα της μεταβλητής μας. Στην επόμενη ενότητα θα αναφέρουμε τις κοινές συμβάσεις σχετικά με την ονοματοδοσία της μεταβλητής.

Για παράδειγμα:

```
String lastname; // δήλωση μεταβλητής τύπου αλφαριθμητικού
int x;         // δήλωση μεταβλητής τύπου int
double y;  // δήλωση μεταβλητής τύπου double
boolean a; // δήλωση μεταβλητής τύπου boolean
```

Επίσης κατά τον ορισμό της μεταβλητής έχουμε την δυνατότητα να αρχικοποιήσουμε την μεταβλητή. Η σύνταξη είναι:

```
τύπος_μεταβλητής όνομα_μεταβλητής = αρχική_τιμή;
```

Η αρχική τιμή πρέπει να ακολουθεί τον τύπο της μεταβλητής. Παραδείγματα ορισμών μεταβλητών με αρχικοποίηση είναι:

```
String lastname = "ΑΝΔΡΕΟΥ"; // δήλωση μεταβλητής τύπου αλφαριθμητικού
int x = 5;         // δήλωση μεταβλητής τύπου int
double y = 10.5;  // δήλωση μεταβλητής τύπου double
boolean a = true; // δήλωση μεταβλητής τύπου Boolean
```

Στην περίπτωση που δεν αρχικοποιήσουμε τις μεταβλητές, οι τιμές που θα έχουν είναι:

```
String lastname = null; // δήλωση μεταβλητής τύπου αλφαριθμητικού
int x = 0;         // δήλωση μεταβλητής τύπου int
double y = 0.0;  // δήλωση μεταβλητής τύπου double
boolean a = false; // δήλωση μεταβλητής τύπου Boolean
```

Μια μεταβλητή μπορεί να δηλωθεί σε ένα σημείο και να πάρει ή να ξαναπάρει τιμή σε άλλο σημείο του προγράμματος.

```
int numberOfObjects = 30;         // δήλωση μεταβλητής
……
numberOfObjects = 50;
```

Επίσης έχουμε και τις παρακάτω συντάξεις για δήλωση περισσότερων μεταβλητών του ιδίου τύπου:

```
String lastname, firstname; // δήλωση μεταβλητής τύπου αλφαριθμητικού
int x = 0, y=1;         // δήλωση μεταβλητής τύπου int
final int MAX_VALUE = 10;  // δήλωση σταθερής τιμής μεταβλητή τύπου int
```

Η λέξη **final** ορίζει ότι η τιμή της μεταβλητής είναι σταθερή (constant) και δεν μπορεί να αλλάξει. Ακολουθώντας τις συμβάσεις το όνομα των constant μεταβλητών είναι πάντα με κεφαλαία

γράμματα όπως φαίνεται και στο παράδειγμα και μπορεί να χρησιμοποιηθεί και το σύμβολο _. Οι constant μεταβλητές δηλώνονται σε ένα σημείο του προγράμματος.

2.3.4. Ονομασία Μεταβλητών

Γενικότερα οι καλές πρακτικές συμβάσεις προγραμματισμού ορίζουν ότι κάθε μεταβλητή να ξεκινά με μικρό γράμμα. Επίσης αν το όνομα μιας μεταβλητής αποτελείται από περισσότερες λέξεις τότε κάθε λέξη να ξεκινά με κεφαλαίο γράμμα για παράδειγμα myVariable. Τα ονόματα μεταβλητών είναι case-sensitive δηλαδή η μεταβλητή με όνομα myCar είναι διαφορετική από την myCAR. Τα ονόματα των μεταβλητών αντίστοιχα με τα ονόματα των κλάσεων μπορούν να περιλαμβάνουν λατινικούς αλφαριθμητικούς χαρακτήρες και το σύμβολο _ . Επίσης τα ονόματα των μεταβλητών δεν μπορούν να περιέχουν κενούς χαρακτήρες.

2.3.5. Δήλωση και Αρχικοποίηση Μεταβλητών

Κατά τη δήλωση μεταβλητών και αναφορών και ανάλογα με το σημείο του κώδικα που γίνεται η δήλωση αυτές μπορεί να αρχικοποιηθούν με default τιμές ή όχι. Έτσι, έχουμε τις εξής περιπτώσεις:

Μεταβλητές μέλη κλάσης (instance variables)

Μεταβλητές **βασικού τύπου**. Οι μεταβλητές μέλη μιας κλάσης που είναι βασικοί τύποι, αρχικοποιούνται πάντοτε αυτόματα από τον compiler με default τιμές, οι οποίες συνοψίζονται στον πίνακα που ακολουθεί.

Τύπος μεταβλητής	Αρχική Τιμή
byte, short, int, long	0
float, double	0.0
boolean	False
char	‘\u0000’

Αναφορές. Οι αναφορές που είναι δηλωμένες ως μεταβλητές μέλη μιας κλάσης, αρχικοποιούνται πάντοτε κατά τη δημιουργία ενός αντικειμένου με την τιμή null.

Τοπικές μεταβλητές (local – automatic)

Μεταβλητές **βασικού τύπου**. Δεν αρχικοποιούνται αυτόματα με καμία τιμή. Μάλιστα, θα πρέπει για να μας επιτρέψει ο compiler να μεταγλωττίσουμε το πρόγραμμα σωστά να τις έχουμε αρχικοποιήσει με κάποια τιμή πριν τις χρησιμοποιήσουμε (π.χ. σε μία σύγκριση). Σε αντίθετη περίπτωση θα παραχθεί compiler error.

Αναφορές. Επίσης δεν αρχικοποιούνται αυτόματα με καμία τιμή, ούτε καν με την τιμή null. Απλά θεωρείται πως δεν περιέχουν τίποτα. Και αυτές θα πρέπει να αρχικοποιηθούν με κάποια τιμή (πραγματικό αντικείμενο ή null) πριν τη χρήση τους, αλλιώς θα παραχθεί compiler error.

Σε γενικές γραμμές, ο κανόνας που ισχύει είναι ο εξής. Οι μεταβλητές μέλη μιας κλάσης (είτε βασικού τύπου είτε αναφορές) αρχικοποιούνται πάντοτε με κάποια default τιμή, ενώ οι τοπικές μεταβλητές θα πρέπει να αρχικοποιηθούν από τον προγραμματιστή.

2.3.6. Ανάθεση (Assignment)

Για να εξηγήσουμε τον ορισμό της Ανάθεσης (Assignment) ας δούμε το παρακάτω παράδειγμα:

```
public class Poligon {
	public static void main(String[] args)  {
		int angle = 4; // 1. δήλωση και αρχικοποίηση μεταβλητής int
		angle = 6;     // 2. ανάθεση – assignment statement
		int sides = 8; // 3. δήλωση και αρχικοποίηση μεταβλητής int
		angle = sides; // 4. ανάθεση – assignment statement
	}
}
```

Οι προτάσεις 2 και 4 ονομάζονται αναθέσεις γιατί αναθέτουν μια τιμή σε μια μεταβλητή. Στην περίπτωση 4 που αποτελεί επίσης πρόταση ανάθεσης αναθέτουμε την τιμή της μεταβλητής sides στην μεταβλητή angle. Δηλαδή μετά από αυτήν την ανάθεση η μεταβλητή angle γίνεται 8.

2.3.7. Αλφαριθμητικά

Ο τύπος δεδομένων αλφαριθμητικό αποτελεί αντικείμενο της Java που ορίζεται από τις κλάσεις String και StringBuider για το λόγο αυτό το όνομα του τύπου ξεκινά με κεφαλαίο γράμμα. Ένα String περικλείεται μέσα σε διπλά εισαγωγικά: "Hello" **.** Για να δηλώσουμε μια μεταβλητή τύπου String η σύνταξη είναι:

```
String lastname; // δήλωση μεταβλητής τύπου αλφαριθμητικού
```

Για να συνενώσουμε μεταβλητές τύπου String χρησιμοποιούμε τον τελεστή + . Μπορούμε να συνενώσουμε μεταβλητές τύπου String, μεταβλητές String με Ακεραίους αριθμούς κα. Για παράδειγμα:

```
String greeting1 = "Hello";
String greeting2 = "World";
String  grmessage1 = greeting1 +" "+greeting2+" !";

String  grmessage2 = greeting1 +" "+greeting2+" "+2018+"!";
```

Συνενώσεις μεταβλητών τύπου String μπορούμε να έχουμε κατά την κλήση της print και println.

Για παράδειγμα:

```
String greeting1 = "Hello";
String greeting2 = "World";
System.out.println(greeting1 +" "+greeting2+" !");
```

Έξοδος:

```
Hello World!
```

2.3.8. Σταθερές (constants)

Οι μεταβλητές ονομάζονται έτσι, μιας και η τιμή τους μπορεί να αλλάξει πολλές φορές κατά τη διάρκεια εκτέλεσης ενός προγράμματος. Υπάρχουν όμως και περιπτώσεις που στο πρόγραμμά μας θέλουμε να αναπαραστήσουμε μία τιμή που παραμένει σταθερή καθ' όλη τη διάρκεια του. Χαρακτηριστικό παράδειγμα τέτοιας περίπτωσης είναι το γνωστό μας π (3,14159...), το e ή οποιαδήποτε άλλη σταθερά κάποιου φυσικού τύπου.

Για να ορίσουμε μία σταθερά στη Java, απλά προσθέτουμε τη δεσμευμένη λέξη final στη σύνταξη που γνωρίζουμε για τη δήλωση μιας απλής μεταβλητής, π.χ.

```
final double PI = 3.14159;
```

2.3.9. Παραδείγματα χρήσης αλφαριθμητικών και αριθμών

```
public class DataTypesExamples {
    public static void main(String[] args) {
        // Ακέραιοι τύποι δεδομένων
        byte byteVar = 127;
        short shortVar = 32767;
        int intVar = 2147483647;
        long longVar = 9223372036854775807L;

        // Τύποι δεδομένων κινητής υποδιαστολής
        float floatVar = 3.14f;
        double doubleVar = 3.141592653589793;

        // Χαρακτήρες
        char charVar = 'A';

        // Αλφαριθμητικά
        String stringVar = "Hello, World!";

        // Λογικοί τύποι δεδομένων
        boolean booleanVar = true;

        // Δηλώσεις και αρχικοποιήσεις
        byte anotherByte = -128;
        short anotherShort = -32768;
        int anotherInt = -2147483648;
        long anotherLong = -9223372036854775808L;

        float anotherFloat = 1.23f;
        double anotherDouble = 1.234567890123456;

        char anotherChar = 'B';
        String anotherString = "Java Programming";

        boolean anotherBoolean = false;

        // Εκτύπωση των τιμών
```

```
        System.out.println("byteVar: " + byteVar);
        System.out.println("shortVar: " + shortVar);
        System.out.println("intVar: " + intVar);
        System.out.println("longVar: " + longVar);
        System.out.println("floatVar: " + floatVar);
        System.out.println("doubleVar: " + doubleVar);
        System.out.println("charVar: " + charVar);
        System.out.println("stringVar: " + stringVar);
        System.out.println("booleanVar: " + booleanVar);

        System.out.println("anotherByte: " + anotherByte);
        System.out.println("anotherShort: " + anotherShort);
        System.out.println("anotherInt: " + anotherInt);
        System.out.println("anotherLong: " + anotherLong);
        System.out.println("anotherFloat: " + anotherFloat);
        System.out.println("anotherDouble: " + anotherDouble);
        System.out.println("anotherChar: " + anotherChar);
        System.out.println("anotherString: " + anotherString);
        System.out.println("anotherBoolean: " + anotherBoolean);
    }
}
```

2.4. Τελεστές

2.4.1. Αριθμητικοί Τελεστές

Στη γλώσσα προγραμματισμού Java χρησιμοποιούμε τους ακόλουθπυς τελεστές για την υλοποίηση αριθμητικών πράξεων

Πράξη	Τελεστής	Παράδειγμα
Πρόσθεση	+	sum = num1 + num2
Αφαίρεση	-	diff = num1 – num2
Πολλαλπασιασμός	*	prod = num1 * num2
Διαίρεση	/	div = num1 / num2
Υπόλοιπο διαίρεσης	%	rest = num1 % num2

2.4.2. Τελεστές Μοναδιαίας Αύξησης και Μείωσης

Η Java εκτός από τους τελεστές πρόσθεσης + και αφαίρεσης – έχει και τους τελεστές Αύξησης ++ και Μείωσης – –

Τελεστής	Περιγραφή
++	Τελεστής Αύξησης κατά 1
--	Τελεστής Μείωσης κατά 1

Για παράδειγμα:

```
num = num + 1;
//γράφεται και
num++;
num = num – 1;
//γράφεται και
num- -;
```

Ας υποθέσουμε ότι:

```
int b;
int a = 5;
b = a++;
```

θα έχει ως αποτέλεσμα το **b** να λάβει την τιμή 5 και η τιμή του **a** να αυξηθεί σε 6.

```
//αλλιώς μπορεί να γραφεί
b = a;
a = a + 1;
```

Δηλαδή, όταν οι συγκεκριμένοι τελεστές (++) ακολουθούν την μεταβλητή τους, εδώ **a,** σε μία έκφραση, πρώτα δίνουν την τρέχουσα τιμή της μεταβλητής τους (**a**) και στη συνέχεια γίνεται η αύξηση στην μεταβλητή τους.

```
int b;
int a = 5;
b = ++a;
```

Πρώτα θα αυξηθεί το a κατά 1 και θα γίνει 6 και μετά θα ανατεθεί η τιμή 6 στο **b**.

```
//αλλιώς μπορεί να γραφεί
a = a + 1;
b = a;
```

Όταν οι συγκεκριμένοι τελεστές (++) βρίσκονται πριν την μεταβλητή τους, εδώ **a,** σε μία έκφραση, πρώτα πραγματοποιείται η αύξηση στην τρέχουσα τιμή της μεταβλητής τους (**a**) και κατόπιν δίνουν την τιμή της μεταβλητής τους (**a**).

2.4.3. Τελεστές Αντικατάστασης (Compound Assignment)

Για κάθε τελεστή από αυτούς που έχουμε εξετάσει μέχρι τώρα, υπάρχει και ο αντίστοιχος τελεστής αντικατάστασης. Οι τελεστές αντικατάστασης δεν προσφέρουν κάποια ιδιαίτερη λειτουργία, απλά προσφέρουν μία εναλλακτική σύνταξη που έχει το ίδιο αποτέλεσμα με τη γραμμή π.χ.,

```
a = a + 5;
```

Χρησιμοποιώντας τον αντίστοιχο τελεστή αντικατάστασης (+=), η ίδια γραμμή θα μπορούσε να γραφτεί:

```
a += 5;
```

Στον πίνακα που ακολουθεί, συνοψίζονται οι τελεστές αντικατάστασης της Java μαζί με παραδείγματα χρήσης τους.

Τελεστής αντικατάστασης	Έκφραση	Ισοδύναμη	Αναθέτει
+=	c += 7	c = c + 7	10 στο c
-=	d -= 4	d = d - 4	1 στο d
*=	e *= 5	e = e * 5	20 στο e
/=	f /= 3	f = f / 3	2 στο f
%=	g %= 9	g = g % 9	3 στο g
&=	c &= 2	c = c & 2	2 στο c
^=	d ^= 4	d = d ^ 4	1 στο d
|=	e |= 3	e = e | 3	7 στο e
<<=	f <<= 2	f = f << 2	24 στο f
>>=	g >>= 2	g = g >> 2	3 στο g
>>>=	c >>>= 1	c = c >>> 1	1 στο c

2.4.4. Σχεσιακοί τελεστές

Οι σχεσιακοί τελεστές αναφέρονται στην σχέση μεταξύ δύο αντικειμένων. Ο παρακάτω πίνακας παρουσιάζει το σύνολο των σχεσιακών τελεστών.

Συνθήκη	Τελεστής	Παράδειγμα
Ίσο	==	int i=1; (i == 1)
Διαφορετικό	!=	int i=2; (i != 2)
μικρότερο	<	int i=1; (i < 0)
Μικρότερο ίσο	<=	int i=1; (i <= 2)
Μεγαλύτερο	>	int i=3; (i > 1)
Μεγαλύτερο ίσο	>=	int i=1; (i >= 1)

Το αποτέλεσμα όλων των σχεσιακών τελεστών είναι μια Boolean τιμή true/false. Κατά την διαδικασία δημιουργίας του προγράμματος μεγάλη προσοχή πρέπει να δίνουμε στην έκφραση της ισότητας η οποία απεικονίζεται με == (δύο ίσον) και της ανάθεσης που απεικονίζεται = (ένα ίσον) .

Για παράδειγμα:

```
int numOfStudents = 10; //δήλωση μεταβλητής και αρχικοποίηση
boolean largeroom; //δήλωση μεταβλητής

if (numOfStudents > 8) {
      largeroom  =  true;
}else {
      largeroom = false;
}
```

2.4.5. Λογικοί Τελεστές

Οι λογικοί τελεστές είναι τρεις, η λογική σύζευξη AND (&&), η λογική διάζευξη OR (||) και η άρνηση NOT (!). Οι λογικοί τελεστές χρησιμοποιούνται για τη σύνθεση ολοκληρωμένων και πιο σύνθετων λογικών παραστάσεων. Οι λογικοί τελεστές παρουσιάζονται στον παρακάτω πίνακα.

Λογικοί τελεστές	Παράδειγμα	Επεξήγηση
&&	(x == y) && (x != z)	**x** ίσο με το **y** και το **x** διάφορο του **z**
||	(x == y) || (x != z)	**x** ίσο με το **y** ή το **x** διάφορο του **z**
!	! (x != z)	**x** όχι διάφορο του **z**

Το αποτέλεσμα των λογικών τελεστών αποτελεί boolean έκφραση και έχει τιμές true/false. Για παράδειγμα:

```
int x = 5;
boolean b = (x++ > 5) && (x == 3);
```

Ο compiler θα αποτιμήσει πρώτα την παράσταση (x++ > 5) η οποία έχει τιμή **false**. Ο λόγος που είναι false είναι διότι πρώτα θα γίνει η σύγκριση και κατόπιν θα αυξηθεί το x κατά 1. Μόλις ο compiler αποτιμήσει την πρώτη παράσταση σε false δεν θα προχωρήσει στο δεξί τμήμα της έκφρασης αφού ο λογικός τελεστής && δύο εκφράσεων είναι true μόνο αν και οι δύο εκφράσεις είναι true αλλιώς είναι false.

2.4.6. Προτεραιότητα Τελεστών

Ισχύουν οι γνωστές προτεραιότητες των πράξεων. Πρώτα πραγματοποιούνται οι πράξεις εντός παρενθέσεων. Ο παρακάτω πίνακας δηλώνει την προτεραιότητα των πράξεων.

Τελεστές	Περιγραφή
! ++ --	NOT , Τελεστές Αύξησης / Μείωσης
/ * %	πολλαπλασιασμοί, οι διαιρέσεις και το υπόλοιπο της διαίρεσης από αριστερά προς τα δεξιά
+ -	προσθεσεις και αφαιρέσεις από αριστερά προς τα δεξιά
< <= >= >	Σχεσιακοί τελεστές
== !=	Ισότητα και Διάφορο
&&	Λογικός τελεστής σύζευξης AND
||	Λογικός τελεστής διάζευξης OR
=	Τελεστής Ανάθεσης

Μεγαλύτερη προτεραιότητα έχουν οι μοναδιαίοι τελεστές NOT και οι Τελεστές Αύξησης και Μείωσης, ενώ αμέσως ακολουθούν οι πράξεις της διαίρεσης, του πολλαπλασιασμού και του υπολοίπου. Ακολουθούν μετά οι λογικοί τελεστές. Στην τελευταία θέση βρίσκονται οι τελεστές ανάθεσης, που αναθέτουν την τελική τιμή μιας έκφρασης σε κάποια μεταβλητή.

Ας δούμε ένα παράδειγμα που δείχνει την προτεραιότητα των πράξεων:

```
public class OperatorPrecedence {
    public static void main(String[] args) {
        int a = 10;
        int b = 5;
        int c = 2;
        int result;

        // Παράδειγμα με πολλαπλές πράξεις
        result = a + b * c; // Πρώτα εκτελείται το b * c, μετά το a + (b *
c)
        System.out.println("Result: " + result); // Εκτυπώνει: Result: 20

        // Χρήση παρενθέσεων για αλλαγή προτεραιότητας
        result = (a + b) * c; // Πρώτα εκτελείται το a + b, μετά το (a + b)
* c
        System.out.println("Result with parentheses: " + result); //
Εκτυπώνει: Result with parentheses: 30
    }
}
```

Στο παραπάνω παράδειγμα, η πράξη b * c εκτελείται πριν από την πράξη a +, λόγω της υψηλότερης προτεραιότητας του πολλαπλασιασμού. Χρησιμοποιώντας παρενθέσεις, μπορούμε να αλλάξουμε την προτεραιότητα των πράξεων.

2.5. Μετατροπές Τύπων (Promotion και Casting)

Η Java υποστηρίζει τη διαδικασία explicit type casting, μέσω της οποίας μπορεί ένας προγραμματιστής να μετατρέψει ένα τύπο μίας μεταβλητής σε κάποιον άλλο. Είναι σημαντικό να τονίσουμε ότι κατά την μετατροπή, πάντα ελλοχεύει ο κίνδυνος να έχουμε κάποια απώλεια ακριβείας (ειδικά στην μετατροπή αριθμών από έναν τύπο σε κάποιον άλλο) και για το λόγο αυτό η συγκεκριμένη λειτουργικότητα θα πρέπει να χρησιμοποιείται με φειδώ.

Για να μετατρέψουμε έναν τύπο σε κάποιον άλλον, αρκεί να τοποθετήσουμε το όνομα του τύπου στον οποίο θέλουμε να μετατρέψουμε μια μεταβλητή ή μία κυριολεκτική τιμή μπροστά από την μεταβλητή ή την τιμή αυτή και μέσα σε ένα ζεύγος από παρενθέσεις, όπως φαίνεται και στο παράδειγμα στα παραδείγματα που ακολουθούν.

```
//Από int σε double:
int num = 10;
```

```
double d = (double) num;
//Από double σε int:
double d = 9.78;
int num = (int) d;
//Από char σε int:
char c = 'A';
int num = (int) c;
//Από int σε char:
int num = 65;
char c = (char) num;
//Από long σε float:
long l = 100000L;
float f = (float) l;
//Από float σε long:
float f = 10.5f;
long l = (long) f;
//Από byte σε short:
byte b = 10;
short s = (short) b;
//Από short σε byte:
short s = 100;
byte b = (byte) s;
//Από int σε String:
int num = 123;
String str = Integer.toString(num);
//Από String σε int:
String str = "123";
int num = Integer.parseInt(str);
```

Στην Java, το “promotion” αναφέρεται στη διαδικασία αυτόματης μετατροπής (ή προαγωγής) μιας /μεταβλητής από έναν τύπο δεδομένων σε έναν άλλο, συνήθως μεγαλύτερο ή πιο ακριβή τύπο. Αυτό συμβαίνει κυρίως σε αριθμητικές πράξεις για να διασφαλιστεί η ακρίβεια των αποτελεσμάτων. Εδώ είναι μερικά παραδείγματα:

```
//Από byte σε int:
byte b = 10;
int i = b; // Αυτόματα προάγεται σε int
//Από short σε int:
//short s = 20;
int i = s; // Αυτόματα προάγεται σε int
//Από char σε int:
```

```
char c = 'A';
int i = c; // Αυτόματα προάγεται σε int
//Από int σε long:
int i = 100;
long l = i; // Αυτόματα προάγεται σε long
//Από float σε double:
float f = 10.5f;
double d = f; // Αυτόματα προάγεται σε double
```

Αυτές οι προαγωγές βοηθούν στην αποφυγή απώλειας δεδομένων και διασφαλίζουν ότι οι πράξεις εκτελούνται με τη μέγιστη δυνατή ακρίβεια.

2.6. Δομές Ελέγχου Ροής

Η σειρά με την οποία εκτελούνται οι εντολές σε ένα πρόγραμμα ονομάζεται ροή ελέγχου. Εκτός εάν ορίζεται διαφορετικά, η βασική εκτέλεση ενός προγράμματος προχωρά με γραμμικό – σειριακό τρόπο. Δηλαδή, η εκτέλεση ενός προγράμματος αρχίζει από την πρώτη εντολή μετακινείται στην επόμενη μέχρι να ολοκληρωθούν όλες οι εντολές του προγράμματος πλήρως. Μια εφαρμογή Java αρχίζει να εκτελείται από την πρώτη γραμμή της μεθόδου main και συνεχίζει βήμα προς βήμα μέχρι να φτάσει στο τέλος της μεθόδου. Η κλήση μιας μεθόδου μεταβάλλει τη ροή του ελέγχου. Όταν καλείται μια μέθοδος, η ροή εκτέλεσης μεταπηδά στον κώδικα που ορίζεται για τη συγκεκριμένη μέθοδο και αρχίζει να εκτελείται. Όταν ολοκληρωθεί η μέθοδος, ο έλεγχος επιστρέφει στη θέση της μεθόδου κλήσης όπου έγινε η επίκληση και η επεξεργασία συνεχίζεται από εκεί. Οι μέθοδοι καθώς και η κλήση των μεθόδων αναλύονται σε επόμενη ενότητα.

Μέσα σε μια μέθοδο, μπορούμε να αλλάξουμε τη ροή εκτέλεσης χρησιμοποιώντας ορισμένους τύπους δηλώσεων προγραμματισμού. Οι δηλώσεις αυτές που ελέγχουν τη ροή εκτέλεσης χωρίζονται σε δύο κατηγορίες: τις συνθήκες (conditional statements) και τους βρόχους (loops).

2.6.1. Η δομή ελέγχου (Conditional statement) if/else

Οι συνθήκες - conditional statements μας επιτρέπουν να επιλέξουμε ποια εντολή (statement) θα εκτελεστεί σε επόμενο βήμα. Η Java περιλαμβάνει τρεις conditional statements τις:

- If … else
- If … else if
- Switch

```
if (<κάποια συνθήκη αληθής>) {
      //A. κάνει κάτι
}else {
     //B. κάνει κάτι διαφορετικό
}
```

```
if (<κάποια συνθήκη αληθής>) {
```

```
     //Α. κάνει κάτι
}else if (<κάποια άλλη συνθήκη αληθής>) {
    //Β. κάνει κάτι άλλο
}…
else {
    //Γ. κάνει κάτι άλλο διαφορετικό
}
```

Η switch λόγω της πολύπλοκότητάς της θα αναλυθεί σε επόμενη ενότητα.

Αν η συνθήκη μετά το if είναι αληθής τότε εκτελούνται οι εντολές που περικλείονται στο μπλοκ Α. Αλλιώς αν η συνθήκη δεν είναι αληθής στην πρώτη περίπτωση του if/else κατευθύνομαστε στο else και εκτελούνται οι εντολές εντός του μπλοκ Β. Στην δεύτερη περίπτωση κατευθυνόμαστε στο else if και ελέγχεται η άλλη συνθήκη αν αυτή είναι αληθής εκτελούνται οι εντολές του μπλοκ Β. Διαφορετικά αν υπάρχει επόμενη else if ελέγχεται η συνθήκη της αν είναι αληθής ή ψευδής και προχωράμε ανάλογα. Αν σε κάθε περίπτωση όλες οι συνθήκες των else if είναι ψευδείς κατευθυνόμαστε στο else και εκτελούνται οι εντολές του μπλοκ Γ. Η συνθήκες του if και των else if αποτελούν **Boolean εκφράσεις** οι οποίες έχουν τιμές true/false. Ο τύπος δεδομένων boolean έχει δύο τιμές την true και την false. Παράδειγμα συνθηκών αποτελούν οι παρακάτω εκφράσεις:

```
lengthx > 20 // μεγαλύτερος
price <= 1000 //μικρότερο ίσο
total == (price * 10) //ίσο
```

2.6.2. Η δομή ελέγχου (Conditional statement) if/else if

Η δομή ελέγχου if..else if μας επιτρέπει να γράφουμε κώδικα για διακλαδωμένες αποφάσεις

και εκτελείται με διαφορετικό τρόπο όπως θα δούμε από ότι η απλή if..else. Η σύνταξή της είναι

η ακόλουθη:

```
if (συνθήκη){
εντολή1;
...
} else if (συνθήκη){
εντολή2;
...
} else {
εντολήΝ;
...
}
```

Όπως βλέπουμε από τη σύνταξη, η δομή ξεκινάει με τη χρήση της δεσμευμένης λέξης if, ακολουθεί η συνθήκη για την πρώτη περίπτωση και το μπλοκ εντολών που περιέχεται σ' αυτήν. Ακολουθούν οι λέξεις else if με τη συνθήκη της δεύτερης περίπτωσης κ.ο.κ. Τέλος, η δομή μπορεί να περιέχει τη

δεσμευμένη λέξη else (είναι προαιρετική, γι αυτό είναι γραμμένη με italics) η οποία όμως δεν ακολουθείται από συνθήκη.

Όταν ο κώδικας συναντήσει μία δομή if..else if, ξεκινάει η αποτίμηση των συνθηκών, μίας μίας ξεχωριστά με τη σειρά που είναι γραμμένες. Αν μία συνθήκη επαληθεύεται (true), εκτελείται το μπλοκ των εντολών που συνδέεται με αυτήν. Αυτό που θα πρέπει να προσέξετε είναι πως μετά την εκτέλεση του μπλοκ των εντολών της συνθήκης που επαληθεύτηκε, η αλυσίδα τερματίζεται και η ροή μεταφέρεται εκτός της δομής. Αυτό σημαίνει πως οι επόμενες συνθήκες που μπορεί να υπάρχουν αγνοούνται και δεν ελέγχονται καν από τον compiler. Γι αυτό, όπως είπαμε και πριν, κάθε φορά μόνο ένα κλαδί της δομής θα εκτελεστεί.

Το τελευταίο τμήμα της δομής (else) χειρίζεται την περίπτωση 'τίποτα από τα παραπάνω' δηλαδή θα τρέξει κάθε φορά που καμία από τις προηγούμενες συνθήκες δεν έχει επαληθευτεί. Το τμήμα αυτό ονομάζεται εξ' ορισμού περίπτωση και όπως αναφέρθηκε ήδη δεν είναι υποχρεωτικό να υπάρχει.

2.6.3. Εμφωλευμένες if

Άλλες φορές χρειάζεται στα προγράμματά μας αν ισχύει μια συνθήκη if να χρειάζεται να ελέγξουμε περαιτέρω με άλλες εσωτερικές συνθήκες if το πρόγραμμα μας. Οι εσωτερικές συνθήκες if ονομάζονται φωλιασμένες if. Για παράδειγμα:

```
if (code == 'R')
    if (height <= 20)
        System.out.println ("Κανονική κατάσταση ");
    else
        System.out.println ("Μπράβο!");
```

Ιδιαίτερη προσοχή στις φωλιασμένες if πρέπει να δίνουμε στο που ξεκινά και που τερματίζει κάθε φορά το μπλοκ εντολών της κάθε if και else.

Για παράδειγμα το αποτέλεσμα του ακόλουθου προγράμματος είναι διαφορετικό από το αποτέλεσμα του αμέσως παραπάνω προγράμματος:

```
if (code == 'R') {
     if (height <= 20)
        System.out.println ("Κανονική κατάσταση ");
} else
        System.out.println ("Μπράβο!");
```

2.6.4. Ο Τριαδικός Τελεστής υπό συνθήκη (Ternary Conditional Operator) X?Y:Z

Ο τριαδικός τελεστής ονομάζεται έτσι μιας και είναι ο μοναδικός που εφαρμόζεται σε τρεις τελεσταίους. Η σύνταξη του τριαδικού τελεστή είναι η ακόλουθη:

```
συνθήκη ? τιμή1 : τιμή2;
```

και έχει αντίστοιχη λειτουργία με αυτήν της απλής **if..else**, δηλαδή η παραπάνω σύνταξη θα μπορούσε να γραφτεί και να έχει το ίδιο αποτέλεσμα ως εξής:

```
if (συνθήκη)
      τιμή1;
else
      τιμή2;
```

Ετσι για παράδειγμα, η παρακάτω συνθήκη επιστρέφει true όταν η μεταβλητή Z έχει τιμή μεγαλύτερη από 100 και false όταν ισχύει το ανάποδο.

```
a = (z > 100) ? true : false
```

2.6.5. Σύνθετες Συνθήκες

Πολλές φορές χρειάζεται στα προγράμματά μας να δημιουργήσουμε σύνθετες συνθήκες για τους ελέγχους μας. Οι σύνθετες συνθήκες προκύπτουν από την σύνθεση δύο ή περισσότερων λογικών παραστάσεων σε μία μεγαλύτερη με τη βοήθεια λογικών τελεστών. Για παράδειγμα:

```
if (!done  &&  (count > MAX))
     System.out.println ("Ολοκληρώθηκε!!!");
```

2.6.6. Η δομή ελέγχου Switch

Η εντολή switch μπορεί να αντικατασταθεί από ένα σύνολο if … else εντολών. Παρακάτω βλέπουμε ένα παράδειγμα χρήσης της switch:

```
switch (idChar)  {
  case 'A':
     aCount = aCount + 1;
     break;
   case 'B':
     bCount = bCount + 1;
     break;
   case 'C':
     cCount = cCount + 1;
     break;
   default:
       System.out.println ("Σφάλμα στο χαρακτήρα εισαγωγής.");
}
```

Η switch παίρνει σαν παράμετρο την μεταβλητή idChar η οποία είναι τύπου char. Οι εντολές case έχουν την λογική των if statements. Αν η τιμή της μεταβλητής idChar είναι ίση με 'A' τότε εκτελείται η εντολή aCount = aCount + 1; Κατόπιν ακολουθεί η εντολή break όπου δηλώνει το τέλος της switch. Αν η τιμή της μεταβλητής idChar δεν είναι ίση με 'A' τότε ο compiler ελέγχει την case 'B'. Αν η τιμή της μεταβλητής idChar δεν είναι ούτε 'A', ούτε 'B', ούτε 'C' τότε ο compiler κατευθύνεται στην εντολή default και εκτελεί το System.out.println ("Σφάλμα στο χαρακτήρα εισαγωγής.");.

Η εντολή switch μπορεί να γραφεί ως ένα σύνολο if statements. Το παραπάνω παράδειγμα μπορεί να γραφεί ως:

```
char idChar;
```

```
if (idChar ==  'A')
     aCount = aCount + 1;
else if (idChar ==  'B')
      bCount = bCount + 1;
else if (idChar ==  'C')
      cCount = cCount + 1;
else
      System.out.println ("Σφάλμα στο χαρακτήρα εισαγωγής.");
```

2.7. Δομές Επανάληψης

Πολλές φορές για την εκτέλεση ενός προγράμματος είναι αναγκαίο να δοθεί η δυνατότητα στους προγραμματιστές να δημιουργούν ένα σύνολο εντολών οι οποία θα εκτελούνται επανειλημμένα όσο κάποια συνθήκη (η οποία φυσικά πρέπει να μεταβάλετε μέσα στο block των εντολών ώστε να μπορεί να ολοκληρωθεί η διαδικασία τρεξίματος του κώδικα). Η Java διαθέτει τρεις τέτοιες δομές, τη while, τη do..while και τη for, τις οποίες θα εξετάσουμε στο παρόν κεφάλαιο, αρχίζοντας από τη while.

2.7.1. Δομή Επανάληψης While

Ξεκινώντας με τη while, πρόκειται για μια δομή επανάληψης που χρησιμοποιείται κυρίως για τη δημιουργία βρόχων όταν δε γνωρίζουμε τον ακριβή αριθμό των επαναλήψεων. Η σύνταξη της while είναι η εξής:

```
while (συνθήκη){
      εντολή1;
      εντολή2;
      ...
}
```

Μια δομή while, αποτελείται από τα εξής βήματα. Αρχικά, αποτιμάται η συνθήκη. Εάν η συνθήλη είναι ψευδής, τερματίζεται η εκτέλεση της επανάληψης και η εκτέλεση του προγράμματος μεταφέρεται εκτός της δομής. Εάν η συνθήκη είναι αληθής, εκτελείται το μπλοκ των εντολών της while και η εκτέλεση επιστρέφει με το πέρας στην συνθήκη για νέο έλεγχο.

Ένα παράδειγμα εκτέλεσης παρουσιάζεται στον παρακάτω κώδικα.

```
public class SumExample {
    public static void main(String[] args) {
        int sum = 0;
        int number = 1;

        while (number <= 100) {
            sum += number;
            number++;
        }

        System.out.println("Το άθροισμα των αριθμών από το 1 έως το 100
είναι: " + sum);
    }
}
```

2.7.2. Δομή Επανάληψης For

Η σύνταξη μιας for αρχίζει με τη χρήση της δεσμευμένης λέξης for. Ακολουθεί μία παρένθεση που περιέχει τρία τμήματα που χωρίζονται μεταξύ τους από ερωτηματικά (;). Στο πρώτο τμήμα πραγματοποιείται η αρχικοποίηση του μετρητή. Η for είναι η μοναδική δομή που χρησιμοποιεί έναν μετρητή για να ελέγχει τον αριθμό των επαναλήψεων. Στο τμήμα αυτό λοιπόν, δίνουμε μία αρχική τιμή στον μετρητή (π.χ. 1) και γίνει μία εργασία που εκτελείται μία μονο φορά στην αρχή εκτέλεσης της δομής επανάληψης. Στο μεσαίο τμήμα βρίσκεται η γνωστή μας συνθήκη, μία λογική παράσταση που αποτιμάται σε ή false. Τέλος, στο τρίτο τμήμα βρίσκεται η έκφραση σύμφωνα με την οποία αυξάνεται ή μειώνεται η τιμή του μετρητή στο τέλος κάθε κύκλου (π.χ. i++)

Τα βήματα που ακολουθούνται κατά την εκτέλεση μιας εντολής αυτού του τύπου, είναι. Αρχικά πρ. Πραγματοποιείται η αρχικοποίηση του μετρητή. Εν συνεχεία αποτιμάται η συνθήκη. Εάν είναι ψευδής, τερματίζεται η εκτέλεση της προγράμματος με την επόμενη εντολή. Εάν είναι αληθής, εκτελείται το σώμα της επανάληψης. Εκτελείται η αύξηση (ή μείωση) του βήματος. Η εκτέλεση επιστρέφει στο βήμα μετά την αρχικοποίηση.

Ένα παράδειγμα εκτέλεσης παρουσιάζεται στον παρακάτω κώδικα.

```
public class ProductExample {
    public static void main(String[] args) {
        int product = 1;

        for (int i = 1; i <= 10; i++) {
            product *= i;
        }

        System.out.println("Το γινόμενο 1 έως το 10 είναι: " + product);
    }
}
```

2.7.3. Δομή Επανάληψης do/while

Η δομή do..while έχει αντίστοιχη λειτουργία με τη while, με τη διαφορά πως οι εντολές που περιέχονται στο σώμα της εκτελούνται πάντοτε τουλάχιστον μία φορά και η συνθήκη αποτιμάται κάθε φορά στο τέλος κάθε επανάληψης. Επίσης, θα παρατηρήσετε πως διαθέτει ιδιόμορφη σύνταξη, συγκρινόμενη με τη σύνταξη των δομών που έχουμε συναντήσει μέχρι τώρα.

```
do {
      εντολή1;
      εντολή2;
      ...
} while (συνθήκη);
```

Τα βήματα που θα ακολουθήσει ο διερμηνέας όταν συναντήσει μία εντολή do…while είναι τα ακόλουθα. Αρχικά εκτελείται το block των εντολών της do…while (άρα η επανάληψη θα εκτελεστεί κατ’ ελάχιστον μία φορά). Εν συνεχεία αποτιμάται η συνθήκη. Εάν είναι ψευδής, τερματίζεται η

εκτέλεση της προγράμματος με την επόμενη εντολή. Εάν είναι αληθής, επιστρέφει στο αρχικό βήμα.

Ένα παράδειγμα εκτέλεσης παρουσιάζεται στον παρακάτω κώδικα.

```
import java.util.Scanner;

public class PositiveNumberExample {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int number;

        do {
            System.out.print("Εισάγετε έναν θετικό αριθμό: ");
            number = scanner.nextInt();
            if (number <= 0) {
                System.out.println("Ο αριθμός δεν είναι θετικός. Δοκιμάστε
ξανά.");
            }
        } while (number <= 0);

        System.out.println("Εισάγατε τον θετικό αριθμό: " + number);
        scanner.close();
    }
}
```

2.7.4. break και continue keywords

Στην παρούσα υποενότητα θα εξετάσουμε δύο δεσμευμένες λέξεις, την `break` και την `continue` οι οποίες χρησιμοποιούνται κατά κύριο λόγο μέσα σε επαναλήψεις. Ξεκινώντας από την `break`, αν βρισκόμαστε σε μία επανάληψη και η ροή εκτέλεσης `συναντήσει` μία `break`, θα μεταφερθεί αμέσως εκτός του βρόχου και θα συνεχίσει εκτελώντας την αμέσως επόμενη εντολή. Η έξοδος είναι άμεση, που σημαίνει πως τυχόν εντολές που βρίσκονται κάτω από την break δεν θα εκτελεστούν.

Επιπρόσθετα η εντολή breaka από τον επίσημο τρόπο εξόδου από μία δομή switch.

Από την άλλη πλευρά, η `continue` αν και σε κάποιες περιπτώσεις μπορεί να παρέχει ευκολία στον προγραμματιστή, δε χρησιμοποιείται τόσο πολύ όσο η `break` γιατί το ίδιο αποτέλεσμα με την `continue` μπορεί να επιτευχθεί και με άλλον τρόπο (π.χ. με μία απλή if). Η `continue` γράφεται επίσης μέσα σε εντολές επανάληψης, και όταν η ροή εκτέλεσης φτάσει σε `μία` continue, τότε θα συνεχίσει με τον αμέσως επόμενο κύκλο, αν υπάρχει, αγνοώντας τις υπόλοιπες εντολές που βρίσκονται μέσα στο μπλοκ των εντολών της επανάληψης.

2.7.5. Εμφωλευμένες εντολές επανάληψης

Όπως στην περίπτωση των δομών επιλογής, έτσι και στην περίπτωση των επαναλήψεων μπορούμε να κάνουμε χρήση μιας τέτοιας δομής μέσα σε μία άλλη, οπότε λέμε ότι έχουμε εμφωλευμένες εντολές επανάληψης. Και εδώ θα πρέπει να προσεχτεί ιδιαίτερα έτσι ώστε η εσωτερική επανάληψη να περικλείεται στην εξωτερική, π.χ. όπως στο απόσπασμα κώδικα που ακολουθεί.

```
for (i = 1; i < 100; i++){
      do {
            System.out.println(i + " iteration");
            x++;
      } while (x < 10); // τέλος do while
} // τέλος for
```

2.7.6. Παράδειγμα

Ένα παράδειγμα χρήσης των ελέγχων ροής και των συνθηκών επανάληψης, παρουσιάζεται και στον κώδικα που ακολουθεί

```
public class Example {
    public static void main(String[] args) {
        int number = 15;

        // Χρήση if/else
        if (number < 10) {
            System.out.println("Το number είναι μικρότερο από 10");
        } else {
            System.out.println("Το number είναι μεγαλύτερο ή ίσο με 10");
        }

        // Χρήση if/elseif/else
        if (number < 10) {
            System.out.println("Το number είναι μικρότερο από 10");
        } else if (number < 20) {
            System.out.println("Το number είναι μικρότερο από 20 αλλά
μεγαλύτερο ή ίσο με 10");
        } else {
            System.out.println("Το number είναι μεγαλύτερο ή ίσο με 20");
        }

        // Χρήση switch
        switch (number) {
            case 10:
                System.out.println("Το number είναι 10");
                break;
            case 15:
                System.out.println("Το number είναι 15");
                break;
            case 20:
                System.out.println("Το number είναι 20");
                break;
            default:
                System.out.println("Το number δεν είναι 10, 15 ή 20");
                break;
        }

        // Χρήση while
        int count = 0;
        while (count < 5) {
```

```
            System.out.println("Η τιμή του count είναι: " + count);
            count++;
        }
    }
}
```

2.8. Πρακτική εξάσκηση

Γράψτε κώδικα που να υλοποιεί τις συνθήκες if…else, if…else if,εμφωλευμένες if, switch και ile παράδειγμα χρήσης των ελέγχων ροής και των συνθηκών επανάληψης. Γράψε ένα πρόγραμμα σε Java που θα διαβάζει τις βαθμολογίες μαθητών (από 0 έως 100) και θα υπολογίζει τον μέσο όρο τους. Το πρόγραμμα θα πρέπει να εκτυπώνει την κατηγορία του μέσου όρου (π.χ. Άριστα, Πολύ Καλά, Καλά, Μέτρια, Αποτυχία) χρησιμοποιώντας τις συνθήκες if/else, if/elseif, switch, και while.

Οδηγίες:

- Εισάγετε τον αριθμό των μαθητών.
- Χρησιμοποιήστε έναν βρόχο while για να διαβάσετε τις βαθμολογίες των μαθητών.
- Υπολογίστε τον μέσο όρο των βαθμολογιών.
- Χρησιμοποιήστε if/else για να ελέγξετε αν ο μέσος όρος είναι έγκυρος (0-100).
- Χρησιμοποιήστε if/elseif για να καθορίσετε την κατηγορία του μέσου όρου.
- Χρησιμοποιήστε switch για να εκτυπώσετε ένα μήνυμα ανάλογα με την κατηγορία.

2.9. Ερωτήσεις Αυτο-αξιολόγησης

1. **Ποιο από τα παρακάτω αντιπροσωπεύει σωστά τη δομή μιας Java κλάσης;**

   α) public class MyClass { public static void main(String[] args) {} }
   β) class MyClass { public void main() {} }
   γ) MyClass { void main() {} }
   δ) public class MyClass() { public main() {} }

2. **Ποια από τις παρακάτω δηλώσεις είναι υποχρεωτική σε κάθε Java εφαρμογή;**

   α) public class
   β) public static void main(String[] args)
   γ) System.out.println
   δ) import java.util.*

3. **Ποιος είναι ο σωστός τρόπος δήλωσης μιας μεταβλητής ακέραιου αριθμού στη Java;**

   α) int num = 10;
   β) num int = 10;
   γ) integer num = 10;
   δ) int = num 10;

4. **Ποιος τελεστής χρησιμοποιείται για την ανάθεση τιμής σε μια μεταβλητή;**

   α) =
   β) ==

γ) +=

δ) =>

5. Όταν ένας int τύπος μετατρέπεται σε double τύπο:

α) Υπάρχει απώλεια δεδομένων

β) Απαιτείται casting

γ) Δεν απαιτείται casting

δ) Δεν είναι εφικτό

6. Τι επιστρέφει η παρακάτω εντολή if (true);

α) Λάθος

β) Αληθές

γ) Σφάλμα κατά τον χρόνο εκτέλεσης

δ) Τίποτα

7. Σε μια δομή switch, ποια λέξη-κλειδί χρησιμοποιείται για να βγούμε από ένα block;

α) exit

β) break

γ) stop

δ) return

8. Ποιο loop εκτελείται τουλάχιστον μία φορά, ανεξαρτήτως συνθήκης;

α) while

β) for

γ) do/while

δ) Καμία από τα παραπάνω

9. Μπορεί μια while να περιέχει άλλη while μέσα της;

α) Ναι

β) Όχι

γ) Μόνο σε συγκεκριμένες περιπτώσεις

δ) Εξαρτάται από τον τύπο των δεδομένων

10. Τι κάνει η εντολή continue;

α) Τερματίζει την επανάληψη

β) Μεταβαίνει στην επόμενη επανάληψη

γ) Τερματίζει το πρόγραμμα

δ) Επαναφέρει την επανάληψη από την αρχή

11. Ποιο από τα παρακάτω χρησιμοποιείται για να εκτυπώσουμε στην κονσόλα στη Java;

α) console.write()

β) System.out.print()

γ) print()

δ) echo()

12. Ποια είναι η σωστή σύνταξη για να εκτυπώσουμε μια συμβολοσειρά (string);

α) System.out.print("Hello World");

β) System.out.print(Hello World);

γ) print("Hello World")

δ) out.print("Hello World");

13. Τι τύπου είναι η κυριολεκτική τιμή 10.5 στη Java;

α) int

β) float

γ) double

δ) long

14. Ποιο από τα παρακάτω είναι έγκυρη Boolean κυριολεκτική τιμή;

α) 1

β) true

γ) yes

δ) T

15. Ποια από τις παρακάτω δηλώσεις είναι σωστή;

α) int a = 5;

β) float num; num = "12.3";

γ) String str = 10;

δ) boolean flag = 5;

16. Μπορούμε να δηλώσουμε δύο μεταβλητές του ίδιου τύπου σε μια γραμμή;

α) Ναι, αλλά απαιτούν ξεχωριστή αρχικοποίηση

β) Όχι

γ) Ναι, με κόμμα

δ) Ναι, με άνω κάτω τελεία

17. Ποιος τύπος δεδομένων είναι κατάλληλος για την αποθήκευση ενός χαρακτήρα;

α) char

β) string

γ) character

δ) text

18. Ποια από τα παρακάτω είναι η σωστή μορφή επανάληψης for;

α) for (int i = 0; i < 10; i++)

β) for i = 0 to 10

γ) for (int i = 0, i < 10, i++)

δ) for (int i; i < 10; i++)

19. Ποιο loop είναι πιο κατάλληλο για άγνωστο αριθμό επαναλήψεων;

α) for

β) while

γ) do/while

δ) Καμία από τα παραπάνω

20. Ποια από τις παρακάτω συνθήκες είναι σωστή;

α) if (x > 5 && y < 10)

β) if (x > 5 ||)

γ) if (x > 5 AND y < 10)

δ) if (x > 5 &&)

21. Τι θα συμβεί αν μια επανάληψη for περιέχει την εντολή break;

α) Θα συνεχίσει στην επόμενη επανάληψη

β) Θα βγει από την επανάληψη

γ) Θα παραλείψει την τρέχουσα επανάληψη

δ) Δεν θα επηρεάσει την επανάληψη

22. Μπορεί η εντολή continue να χρησιμοποιηθεί εκτός επανάληψης;

α) Ναι

β) Όχι

γ) Μόνο μέσα σε if

δ) Μόνο μέσα σε switch

2.10. Απαντήσεις στις ερωτήσεις Αυτο-αξιολόγησης

1. α	2. β	3. α	4. α	5. γ
6. β	7. β	8. γ	9. α	10. β
11. β	12. α	13. γ	14. β	15. α
16. γ	17. α	18. α	19. β	20. α
21. β	22. β			

ΚΕΦΑΛΑΙΟ 3: Έννοιες αντικειμενοστραφούς προγραμματισμού Μέρος Α

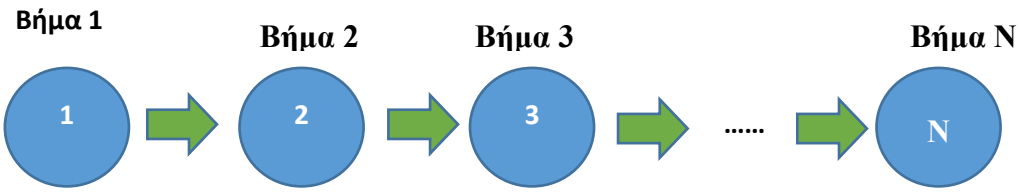
3.1. Εκπαιδευτικοί στόχοι του παρόντος κεφαλαίου

Οι εκπαιδευτικοί στόχοι του παρόντος κεφαλαίου, συνοψίζονται στα κάτωθι σημεία. Οι εκπαιδευόμενοι:

- θα γνωρίσουν τι είναι ο διαδικαστικός ή δομημένος προγραμματισμός,
- θα γνωρίσουν τον Αντικειμενοστραφή προγραμματισμό,
- θα γνωρίσουν πως υλοποιούνται οι κλάσεις και τα αντικείμενα,
- θα γνωρίσουν τι έννοια των Ιδιοτήτων (properties) και συμπεριφορών (behaviors) των κλάσεων,
- θα μάθουν πως να ομαδοποιούν κλάσεις δημιουργώντας packages.

3.2. Διαδικαστικός ή Δομημένος Προγραμματισμός (Procedural ή Structured Programming) (σύντομη αναφορά)

Οι παλαιότερες γλώσσες προγραμματισμού όπως οι C, PASCAL, FORTRAN έδιναν έμφαση στην διαδικασία και στα στάδια που ακολουθούνται για την επίτευξη κάποιου στόχου. Ο προγραμματισμός γινόταν με τον καθορισμό μιας ακολουθίας βημάτων από το Βήμα 1 έως το Βήμα Ν και η υλοποίηση των προγραμμάτων ακολουθούσε τη σειρά των αντίστοιχων υπορουτινών του κάθε βήματος.



Εικόνα 9 Διαδικαστικός ή δομημένος προγραμματισμός

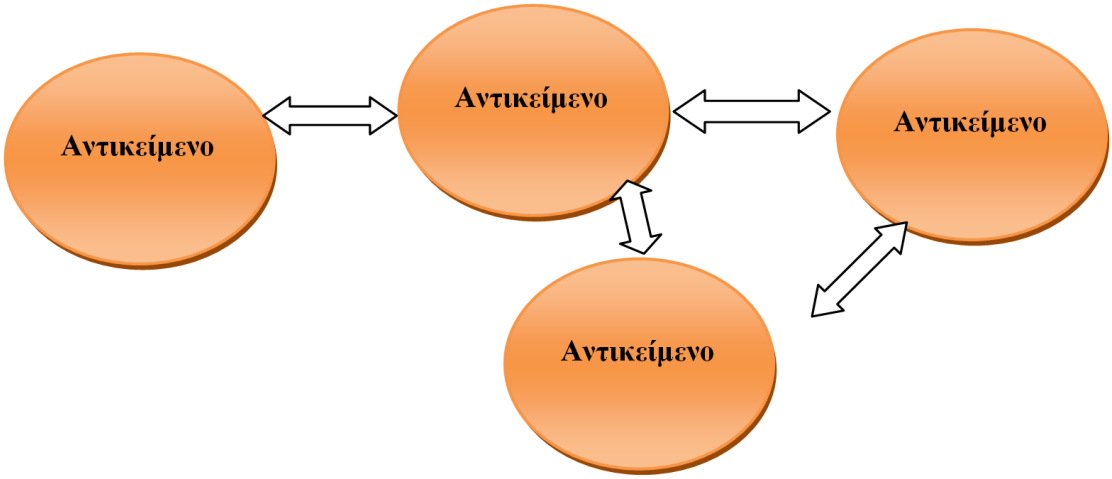
Τα μειονεκτήματα του Διαδικαστικού προγραμματισμού είναι:

- Ο Διαδικαστικός προγραμματισμός δεν μοντελοποιεί σωστά τον πραγματικό κόσμο. Στον φυσικό κόσμο έχουμε οντότητες (αντικείμενα, ανθρώπους) που έχουν χαρακτηριστικά, και μπορούν να εκτελέσουν κάποιες δράσεις αντίθετα ο διαδικαστικός προγραμματισμός θεωρεί μόνο την ύπαρξη σταδίων.
- Είναι δύσκολο να συντηρούμε προγράμματα. Μία αλλαγή σε λογική μπορεί να απαιτεί αλλαγή σε περισσότερα βήματα της διαδικασίας
- Έχει αποδειχθεί επίσης δύσκολη η βελτίωση και αναβάθμιση των προγραμμάτων μας.

3.3. Αντικειμενοστραφής προγραμματισμός

Η Java είναι μια αντικειμενοστραφής γλώσσα προγραμματισμού όπως και η C++. Ο πραγματικός κόσμος που ζούμε ακολουθεί τη φιλοσοφία του Αντικειμενοστραφή προγραμματισμού. Ο κόσμος μας αποτελείται από οντότητες με χαρακτηριστικά και οι οντότητες έχουν συγκεκριμένες συμπεριφορές και συμμετέχουν σε δράσεις. Τα αντικείμενα που υποστηρίζονται στον αντικειμενοστραφή προγραμματισμό αντιστοιχιστούν στις οντότητες του πραγματικού κόσμου. Για παράδειγμα το αυτοκίνητο, ο εργαζόμενος, το υπουργείο κ.α. είναι οντότητες του πραγματικού κόσμου και αντικείμενα στον αντικειμενοστραφή προγραμματισμό. Ένα από τα πλεονεκτήματα του αντικειμενοστραφή προγραμματισμού είναι το γεγονός ότι τα αντικείμενα μπορούν να χρησιμοποιηθούν ώστε να αναπαραστήσουν τις οντότητες του πραγματικού κόσμου αποτελεσματικά.

Επίσης ένα επιπλέον χαρακτηριστικό του αντικειμενοστραφή προγραμματισμού είναι η δημιουργία αντικειμένων, αυτόνομου δηλαδή κώδικα. Τα αντικείμενα μπορούν να αλληλεπιδρούν με άλλα αντικείμενα ώστε να επιλύονται προβλήματα.



Εικόνα 10 Αντικειμενοστραφής Προγραμματισμός - Αλληλεπίδραση Αντικειμένων

3.3.1. Βασική ορολογία αντικειμενοστραφή προγραμματισμού

Η βασική ορολογία του αντικειμενοστραφή προγραμματισμού είναι τα:

- αντικείμενο - object
- χαρακτηριστικό - attribute
- συμπεριφορά - method
- κλάση - class
- ενθυλάκωση - encapsulation
- κληρονομικότητα - inheritance
- πολυμορφισμός - polymorphism

Ένα **αντικείμενο** στην φιλοσοφία του αντικειμενοστραφή προγραμματισμού έχει **χαρακτηριστικά** και κάποια **συμπεριφορά** ή αλλιώς **μεθόδους**. Οι συμπεριφορές ενός αντικειμένου είναι οι δραστηριότητες που συνδέονται με το αντικείμενο.

Για παράδειγμα το αντικείμενο αυτοκίνητο έχει ως χαρακτηριστικά τον αριθμό πλαισίου, το χρώμα, το τιμόνι, το γκάζι, το φρένο, συμπλέκτης, ταχύτητες κ.α.. (το τιμόνι, το φρένο, το γκάζι κα. αποτελούν και τα ίδια αντικείμενα.) Αυτά μπορεί να διαφέρουν ανάμεσα στους κατασκευαστές αλλά όλα αυτά αποτελούν τα βασικά χαρακτηριστικά του αντικειμένου αυτοκίνητο. Γενικά τα χαρακτηριστικά ενός αντικειμένου μπορεί να είναι:

- **primitive-απλά χαρακτηριστικά** δηλαδή να αντιστοιχούν σε μια τιμή όπως το χαρακτηριστικό αριθμός πλαισίου του αντικειμένου αυτοκινήτου,
- αλλά μπορεί να είναι και πιο **σύνθετα χαρακτηριστικά δηλαδή ένα άλλο αντικείμενο** όπως το τιμόνι του αντικειμένου αυτοκινήτου.

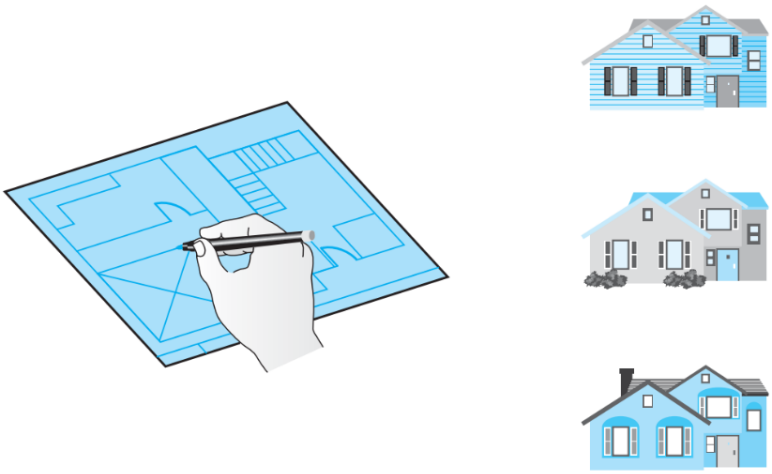
Οι τιμές των χαρακτηριστικών ενός αντικειμένου ορίζουν την τρέχουσα κατάσταση (state) του αντικειμένου.

Το **αντικείμενο** έχει **συμπεριφορές** ή αλλιώς **μεθόδους**. Για παράδειγμα το αυτοκίνητο έχει συμπεριφορές όπως:

- στρίβει αριστερά/δεξιά με τη χρήση του τιμονιού,
- κινείται πατώντας γκάζι και αλλάζοντας ταχύτητα με το παράλληλο πάτημα του συμπλέκτη,
- κάνει όπισθεν,
- κ.α.

Η υλοποίηση της κάθε συμπεριφοράς/μεθόδου διαφέρει σε κάθε κατασκευαστή, αλλά η χρήση της κάθεμιάς είναι η ίδια για όλους. Δηλαδή η χρήση του τιμονιού για να στρίβουμε δεξιά/αριστερά γίνεται με τον ίδιο τρόπο ανεξαρτήτως κατηγορίας, κατασκευαστή και μοντέλου του αυτοκινήτου. Επίσης, ο μηχανισμός με τον οποίο υλοποιείται αυτή η δυνατότητα του αυτοκινήτου να στρίβει δεξιά/αριστερά είναι άγνωστος στον χρήστη του αυτοκινήτου δηλαδή στον οδηγό. Ο οδηγός δεν χρειάζεται να γνωρίζει αλλά και ούτε ενδιαφέρεται για τον τρόπο μετάδοσης της κίνησης από το τιμόνι στους τροχούς ώστε το αυτοκίνητο να στρίψει. Επίσης, η χρήση ενός αυτοκινήτου δεν αλλάζει με την αναβάθμισή του ή την αλλαγή ορισμένων χαρακτηριστικών του (όπως π.χ. νέα μηχανή, λάστιχα, κλπ). Αλλάζει μόνο η υλοποίηση της συμπεριφοράς του αλλά όχι η χρήση του.

Στον αντικειμενοστραφή προγραμματισμό ένα **αντικείμενο ορίζεται** από την **κλάση**-**class**. Η **κλάση** είναι το **μοντέλο** ή αλλιώς το **καλούπι** από το οποίο δημιουργείται το αντικείμενο. Για παράδειγμα το αρχιτεκτονικό σχέδιο ενός οικήματος που δημιουργείται από τον αρχιτέκτονα, ορίζει τα βασικά χαρακτηριστικά του οικήματος όπως τους τοίχους, τα παράθυρα, τις πόρτες κ.α.. Όμως από ένα αρχιτεκτονικό σχέδιο μπορούν να χτιστούν διάφορα σπίτια.



Εικόνα 11 Κλάση - το αρχιτεκτονικό σχέδιο, Αντικείμενα - τα σπίτια που χτίζονται βάσει αυτού. (Πηγή John Lewis, Peter DePasquale, Joseph Chase Java Foundations Introduction to Program Design and Data Structures 2nd Edition)

Η κλάση-class αποτελεί το σχέδιο/καλούπι ενός αντικειμένου, ορίζει τα χαρακτηριστικά του αντικειμένου καθώς και τις μεθόδους–methods που αναπαριστούν την συμπεριφορά του αντικειμένου. Μέσω των μεθόδων-methods παίρνουν τιμές τα χαρακτηριστικά του αντικειμένου και ορίζεται έτσι η κατάσταση (state) του αντικειμένου. Πολλά αντικείμενα μπορούν να δημιουργηθούν από μια κλάση. Τα **αντικείμενα** τα **ονομάζουμε** αλλιώς και **στιγμιότυπα** (**instances**) της **κλάσης**.

3.3.2. Αντικείμενα

Θα μπορούσε κανείς να δει το αντικείμενο σε επίπεδο προγραμματισμού ως το μέρος του κώδικα που είναι επαναχρησιμοποιήσιμο και αναπαριστά κάποιο πραγματικό αντικείμενο στον πραγματικό κόσμο. Είναι φανερό δε ότι τα αντικείμενα αποτελούν τη βασικότερη έννοια του αντικειμενοστραφούς προγραμματισμού, καθώς από αυτά πηγάζει το όνομά του.

Τα στοιχεία εκείνα που χαρακτηρίζουν το αντικείμενο εμφανίζονται στο προγραμματιστικό μέρος ως μεταβλητές ενώ η συμπεριφορά του αντικειμένου, κωδικοποιείται με τη μορφή μεθόδων. Το σύνολο των δύο (στοιχεία και συμπεριφορά) ορίζει την κλάση όπως αυτή περιγράφεται στο κεφάλαιο που ακολουθεί.

3.3.3. Κλάσεις

Στη Java, όπως και στις περισσότερες αντικειμενοστρεφείς γλώσσες, για την κωδικοποίηση των αντικειμένων χρησιμοποιείται η δομή της κλάσης. Η κλάση λοιπόν αποτελεί τη δομική μονάδα του αντικειμενοστρεφούς προγραμματισμού, μέσω της οποίας ο προγραμματιστής ορίζει νέους τύπους. Στον κώδικα μιας κλάσης ο προγραμματιστής περικλείει όλα τα χαρακτηριστικά αλλά και τη συμπεριφορά του αντικειμένου στον πραγματικό κόσμο που θέλει να αναπαραστήσει. Ετσι με την δημιουργία μίας κλάσης στην ουσία αναπαριστούμε κάποιο αντικείμενο στον κώδικα, με συγκεκριμένο όνομα το οποίο περιέχει συγκεκριμένες μεταβλητές που το χαρακτηρίζουν και

συγκεκριμένες μεθόδους. Τα πραγματικά αντικείμενα δημιουργούνται αργότερα, χρησιμοποιώντας τον ορισμό της κλάσης.

Όταν δημιουργούμε μία νέα κλάση στη Java, μπορούμε να ορίσουμε την ορατότητά της, δηλαδή ποιος θα μπορεί να την χρησιμοποιήσει. Η ορατότητα καθορίζεται από τη χρήση ειδικών λέξεων που ονομάζονται προσδιοριστές. Στην περίπτωση του καθορισμού ορατότητας μιας κλάσης, ο μοναδικός προσδιοριστής που μπορεί να χρησιμοποιηθεί είναι ο public. Κάνοντας μια κλάση public, αυτή είναι ορατή από όλες τις υπόλοιπες (αυτό θα πρέπει να κάνουμε πάντα). Αν δεν χρησιμοποιήσουμε τη λέξη public, η κλάση παίρνει αυτόματα την default ορατότητα. Όταν μια κλάση έχει default ορατότητα, είναι ορατή μόνο από τις κλάσεις που βρίσκονται στο ίδιο πακέτο. Στα παρακάτω αποσπάσματα κώδικα έχουν δηλωθεί μια κλάση με public ορατότητα (αριστερά) και μία με default (δεξιά).

```
public class A {
	// ορατή παντού
}

class B {
	// ορατή στο πακέτο
}
```

Εκτός από τον προσδιοριστή `public`, υπάρχουν και άλλοι που μπορούν να χρησιμοποιηθούν σε επίπεδο ορισμού κλάσης. Ο προσδιοριστής `abstract` μετατρέπει μία κλάση σε αφηρημένη (δε μπορεί να δημιουργήσει αντικείμενα), ενώ ο `final` μαρκάρει μια κλάση ως τελική (δεν μπορεί να επεκταθεί μέσω κληρονομικότητας). Οι δύο αυτοί προσδιοριστές είναι αντίθετοι και άρα δε μπορούν να χρησιμοποιηθούν μαζί σε κάποιον ορισμό κλάσης. Στον κώδικα που ακολουθεί έχει οριστεί μία κλάση ως αφηρημένη (αριστερά) και μία ως τελική (δεξιά).

```
public abstract class A {
	// αφηρημένη κλάση
}

public final class B {
// τελική κλάση
}
```

3.3.4. Μέθοδοι (methods) και μεταβλητές (variables) Κλάσης

Μία τυπική κλάση αποτελείται από δύο μέρη. Από τα δεδομένα τα οποία ορίζουν τα χαρακτηριστικά του αντικειμένου με τη μορφή μεταβλητών μελών (member variables) και από τις συναρτήσεις που καθορίζουν τη συμπεριφορά του αντικειμένου και που μας δίνουν πρόσβαση στις μεταβλητές μέλη.

Οι συναρτήσεις αυτές ονομάζονται **μέθοδοι** (methods). Τόσο οι μεταβλητές όσο και οι μέθοδοι ονομάζονται μέλη της κλάσης.

3.3.4.1 Μεταβλητές Κλάσης

Μια μεταβλητή σε μια κλάση Java είναι ένα στοιχείο που χρησιμοποιείται για να αποθηκεύει δεδομένα που σχετίζονται με τα αντικείμενα της κλάσης. Οι μεταβλητές αυτές μπορούν να

περιγράφουν την κατάσταση ή τα χαρακτηριστικά ενός αντικειμένου. Κάθε μεταβλητή έχει έναν τύπο δεδομένων, όπως `int`, `double`, `String`, κ.λπ., που καθορίζει το είδος των τιμών που μπορεί να αποθηκεύσει.

Οι μεταβλητές σε μια κλάση μπορούν να κατηγοριοποιηθούν σε διάφορους τύπους, ανάλογα με το πού και πώς δηλώνονται. Οι μεταβλητές **στιγμιοτύπου** είναι αυτές που δηλώνονται μέσα στην κλάση αλλά έξω από οποιαδήποτε μέθοδο. Αυτές οι μεταβλητές είναι μοναδικές για κάθε αντικείμενο της κλάσης και αποθηκεύουν δεδομένα που είναι συγκεκριμένα για το κάθε αντικείμενο. Όταν δημιουργείται ένα νέο αντικείμενο, δημιουργείται και ένα νέο σύνολο μεταβλητών στιγμιοτύπου.

Οι **στατικές** μεταβλητές, από την άλλη πλευρά, δηλώνονται με τη λέξη-κλειδί `static` και είναι κοινές για όλα τα αντικείμενα της κλάσης. Αυτό σημαίνει ότι υπάρχει μόνο ένα αντίγραφο της στατικής μεταβλητής, ανεξάρτητα από τον αριθμό των αντικειμένων που έχουν δημιουργηθεί. Οι στατικές μεταβλητές χρησιμοποιούνται συχνά για να αποθηκεύουν δεδομένα ή καταστάσεις που είναι κοινές για όλα τα αντικείμενα της κλάσης.

Οι **τοπικές** μεταβλητές είναι αυτές που δηλώνονται μέσα σε μεθόδους, κατασκευαστές ή μπλοκ. Αυτές οι μεταβλητές είναι διαθέσιμες μόνο κατά την εκτέλεση της μεθόδου ή του μπλοκ και δεν μπορούν να χρησιμοποιηθούν έξω από αυτό. Οι τοπικές μεταβλητές χρησιμοποιούνται για προσωρινή αποθήκευση δεδομένων που είναι απαραίτητα μόνο κατά την εκτέλεση μιας συγκεκριμένης λειτουργίας.

Η χρήση των μεταβλητών σε μια κλάση είναι κρίσιμη για τον ορισμό της κατάστασης και της συμπεριφοράς των αντικειμένων. Οι μεταβλητές επιτρέπουν στα αντικείμενα να αποθηκεύουν και να διαχειρίζονται δεδομένα, τα οποία μπορούν να χρησιμοποιηθούν από τις μεθόδους της κλάσης για την εκτέλεση διαφόρων λειτουργιών. Η σωστή διαχείριση και χρήση των μεταβλητών είναι θεμελιώδης για την ανάπτυξη αποδοτικού και κατανοητού κώδικα σε Java.

Για παράδειγμα στην παρακάτω κλάση που περιγράφει το αυτοκίνητο, η μεταβλητή `speed` αφορά τη μέγιστη ταχύτητα που μπορεί να αναπτύξει το αυτοκίνητο, ενώ η μεταβλητή `color` αφορά το χρώμα.

```
public class Car {
      int maxSpeed;
      String color;
    }
}
```

3.3.4.2 Μέθοδοι (methods) Κλάσης

Μια μέθοδος σε μια κλάση Java είναι μια συλλογή από δηλώσεις που εκτελούν μια συγκεκριμένη λειτουργία ή υπολογισμό. Οι μέθοδοι είναι ουσιαστικά οι λειτουργίες ή οι συμπεριφορές που μπορεί να εκτελέσει ένα αντικείμενο της κλάσης. Κάθε μέθοδος έχει ένα όνομα, έναν τύπο επιστροφής, και μπορεί να δέχεται παραμέτρους.

Οι μέθοδοι μπορούν να κατηγοριοποιηθούν σε δύο κύριους τύπους: τις μεθόδους **στιγμιοτύπου** και τις στατικές μεθόδους. Οι μέθοδοι στιγμιοτύπου είναι αυτές που δεν δηλώνονται με τη λέξη-κλειδί `static` και μπορούν να καλούνται σε αντικείμενα της κλάσης. Αυτές οι μέθοδοι έχουν

πρόσβαση σε μεταβλητές στιγμιοτύπου και άλλες μεθόδους στιγμιοτύπου της ίδιας κλάσης. Για παράδειγμα, μια μέθοδος στιγμιοτύπου μπορεί να είναι μια μέθοδος που αυξάνει την ταχύτητα ενός αυτοκινήτου:

```
public class Car {
    int speed;
    // Μέθοδος στιγμιοτύπου
    void accelerate(int increment) {
        speed += increment;
    }
}
```

Από την άλλη πλευρά, οι στατικές μέθοδοι δηλώνονται με τη λέξη-κλειδί `static` και μπορούν να καλούνται χωρίς να δημιουργηθεί αντικείμενο της κλάσης. Οι στατικές μέθοδοι έχουν πρόσβαση μόνο σε στατικές μεταβλητές και άλλες στατικές μεθόδους. Χρησιμοποιούνται συχνά για λειτουργίες που δεν εξαρτώνται από την κατάσταση ενός συγκεκριμένου αντικειμένου. Για παράδειγμα, μια στατική μέθοδος μπορεί να είναι μια μέθοδος που εμφανίζει τον αριθμό των αυτοκινήτων:

```
public class Car {
    static int numberOfCars;

    // Στατική μέθοδος
    static void displayNumberOfCars() {
        System.out.println("Number of cars: " + numberOfCars);
    }
}
```

Οι μέθοδοι μπορούν επίσης να έχουν παραμέτρους, οι οποίες είναι δεδομένα που περνούν στη μέθοδο όταν καλείται. Οι παράμετροι επιτρέπουν στις μεθόδους να δέχονται εισόδους και να εκτελούν λειτουργίες με βάση αυτές τις εισόδους. Για παράδειγμα, η μέθοδος `accelerate` που είδαμε παραπάνω δέχεται μια παράμετρο `increment`, η οποία καθορίζει πόσο θα αυξηθεί η ταχύτητα του αυτοκινήτου.

Επιπλέον, οι μέθοδοι μπορούν να επιστρέφουν τιμές. Ο τύπος επιστροφής μιας μεθόδου καθορίζει το είδος της τιμής που θα επιστρέψει η μέθοδος. Αν μια μέθοδος δεν επιστρέφει καμία τιμή, χρησιμοποιείται ο τύπος `void`. Για παράδειγμα, μια μέθοδος που υπολογίζει και επιστρέφει την απόσταση που έχει διανύσει ένα αυτοκίνητο μπορεί να έχει τον εξής ορισμό:

```
public class Car {
    int speed;
    int time;

    // Μέθοδος που επιστρέφει τιμή
    int calculateDistance() {
        return speed * time;
    }
}
```

Οι μέθοδοι είναι θεμελιώδεις για τον αντικειμενοστραφή προγραμματισμό, καθώς επιτρέπουν την ενθυλάκωση της λειτουργικότητας μέσα σε αντικείμενα και την αλληλεπίδραση μεταξύ αυτών των αντικειμένων. Η σωστή χρήση των μεθόδων μπορεί να κάνει τον κώδικα πιο οργανωμένο, επαναχρησιμοποιήσιμο και ευκολότερο στη συντήρηση.

3.3.4.3 Παράδειγμα Μεθόδων και Μεταβλητών κλάσης

Ένα παράδειγμα μεθόδων και μεταβλητών κλάσεων σε κώδικα Java ακολουθεί.

```
public class Car {
    // Μεταβλητές στιγμιοτύπου
    String color;
    int speed;

    // Στατική μεταβλητή
    static int numberOfCars;

    // Κατασκευαστής
    public Car(String color, int speed) {
        this.color = color;
        this.speed = speed;
        numberOfCars++;
    }

    // Μέθοδος στιγμιοτύπου
    void accelerate(int increment) {
        speed += increment;
    }

    // Στατική μέθοδος
    static void displayNumberOfCars() {
        System.out.println("Number of cars: " + numberOfCars);
    }

    // Τοπική μέθοδος
    void displaySpeed() {
        int currentSpeed = speed;
        System.out.println("Current speed: " + currentSpeed);
    }
}
```

3.3.5. Ιδιότητες (properties) και συμπεριφορές (behaviors) Κλάσης

3.3.5.1 Ιδιότητες (properties) Κλάσης

Ο όρος “ιδιότητες” χρησιμοποιείται συχνά για να περιγράψει τις μεταβλητές μιας κλάσης μαζί με τις μεθόδους πρόσβασης (getters) και τροποποίησης (setters) που επιτρέπουν την ανάγνωση και την τροποποίηση αυτών των μεταβλητών. Οι **properties** είναι ένας τρόπος να ενθυλακώσουμε τις μεταβλητές και να ελέγξουμε την πρόσβαση σε αυτές.

Οι ιδιότητες μπορούν να είναι δημόσιες (public), ιδιωτικές (private), προστατευμένες (protected) ή να έχουν προεπιλεγμένη πρόσβαση (default), ανάλογα με το επίπεδο πρόσβασης που θέλουμε να επιτρέψουμε.

Δημόσιες Ιδιότητες (Public Properties)

Οι δημόσιες ιδιότητες είναι προσβάσιμες από οποιοδήποτε σημείο του κώδικα. Αυτό σημαίνει ότι μπορούμε να διαβάσουμε ή να τροποποιήσουμε την τιμή τους από οποιαδήποτε κλάση. Ωστόσο, η χρήση δημόσιων ιδιοτήτων δεν είναι πάντα η καλύτερη πρακτική, καθώς μπορεί να οδηγήσει σε απρόβλεπτες αλλαγές στην κατάσταση του αντικειμένου.

```
public class Car {
    // Δημόσια ιδιότητα
    public String color;
}
```

Ιδιωτικές Ιδιότητες (Private Properties)

Οι ιδιωτικές ιδιότητες είναι προσβάσιμες μόνο μέσα από την ίδια την κλάση. Αυτό σημαίνει ότι δεν μπορούμε να τις διαβάσουμε ή να τις τροποποιήσουμε απευθείας από άλλες κλάσεις. Η χρήση ιδιωτικών ιδιοτήτων είναι μια καλή πρακτική, καθώς επιτρέπει την ενθυλάκωση (encapsulation) και προστατεύει την κατάσταση του αντικειμένου από ανεπιθύμητες αλλαγές.

```
public class Car {
    // Ιδιωτική ιδιότητα
    private String color;

    // Μέθοδος για την ανάκτηση της τιμής της ιδιότητας
    public String getColor() {
        return color;
    }

    // Μέθοδος για την τροποποίηση της τιμής της ιδιότητας
    public void setColor(String color) {
        this.color = color;
    }
}
```

Προστατευμένες Ιδιότητες (Protected Properties)

Οι προστατευμένες ιδιότητες είναι προσβάσιμες μέσα από την ίδια την κλάση, τις υποκλάσεις της (subclasses), και άλλες κλάσεις στο ίδιο πακέτο. Αυτό το επίπεδο πρόσβασης είναι χρήσιμο όταν θέλουμε να επιτρέψουμε την πρόσβαση σε ιδιότητες από κλάσεις που επεκτείνουν την αρχική κλάση.

```
public class Car {
    // Προστατευμένη ιδιότητα
    protected String color;
}
```

Ιδιότητες με Προεπιλεγμένη Πρόσβαση (Default Properties)

Οι ιδιότητες με προεπιλεγμένη πρόσβαση είναι προσβάσιμες μόνο μέσα από το ίδιο πακέτο. Δεν χρησιμοποιούν καμία λέξη-κλειδί για τον καθορισμό του επιπέδου πρόσβασης.

```
Java

public class Car {
    // Ιδιότητα με προεπιλεγμένη πρόσβαση
    String color;
}
```

Παράδειγμα Κλάσης με Ιδιότητες

Ας δούμε ένα πλήρες παράδειγμα κλάσης που περιλαμβάνει ιδιότητες με διαφορετικά επίπεδα πρόσβασης:

```
public class Car {
    // Ιδιωτική ιδιότητα
    private String color;

    // Δημόσια ιδιότητα
    public int speed;

    // Προστατευμένη ιδιότητα
    protected String model;

    // Κατασκευαστής
    public Car(String color, int speed, String model) {
        this.color = color;
        this.speed = speed;
        this.model = model;
    }

    // Μέθοδος για την ανάκτηση της τιμής της ιδιωτικής ιδιότητας
    public String getColor() {
        return color;
    }

    // Μέθοδος για την τροποποίηση της τιμής της ιδιωτικής ιδιότητας
    public void setColor(String color) {
        this.color = color;
    }
}
```

Σε αυτό το παράδειγμα, η κλάση Car έχει μια ιδιωτική ιδιότητα color, μια δημόσια ιδιότητα speed, και μια προστατευμένη ιδιότητα model. Οι μέθοδοι getColor και setColor επιτρέπουν την πρόσβαση και την τροποποίηση της ιδιωτικής ιδιότητας color.

Οι ιδιότητες είναι θεμελιώδεις για την περιγραφή της κατάστασης των αντικειμένων και την ενθυλάκωση δεδομένων μέσα σε μια κλάση. Η σωστή χρήση των ιδιοτήτων και των επιπέδων πρόσβασης μπορεί να κάνει τον κώδικα πιο ασφαλή, οργανωμένο και ευκολότερο στη συντήρηση.

3.3.5.2 Συμπεριφορές (behaviors) Κλάσης

Στον αντικειμενοστραφή προγραμματισμό, μια κλάση δεν είναι μόνο μια συλλογή δεδομένων, αλλά και μια συλλογή από λειτουργίες ή μεθόδους που καθορίζουν πώς αυτά τα δεδομένα μπορούν να χρησιμοποιηθούν και να τροποποιηθούν. Αυτές οι λειτουργίες ή μέθοδοι είναι γνωστές ως behavior της κλάσης. Το behavior μιας κλάσης καθορίζει πώς τα αντικείμενα αυτής της κλάσης αλληλεπιδρούν με τον έξω κόσμο και μεταξύ τους.

Παράδειγμα Κλάσης με Behavior

Ας πάρουμε για παράδειγμα μια κλάση Car στην Java:

```
public class Car {
    // Ιδιότητες (attributes)
    private String brand;
    private String model;
    private int year;
    private double speed;

    // Constructor
    public Car(String brand, String model, int year) {
        this.brand = brand;
        this.model = model;
        this.year = year;
        this.speed = 0.0;
    }

    // Μέθοδοι (behavior)
    public void accelerate(double increment) {
        this.speed += increment;
    }

    public void brake(double decrement) {
        this.speed -= decrement;
        if (this.speed < 0) {
            this.speed = 0;
        }
    }

    public double getSpeed() {
        return this.speed;
    }

    public String getDetails() {
        return "Brand: " + this.brand + ", Model: " + this.model + ", Year:
" + this.year + ", Speed: " + this.speed;
    }
}
```

Στο παραπάνω παράδειγμα, η κλάση Car έχει τέσσερις ιδιότητες: `brand`, `model`, `year`, και `speed`. Αυτές οι ιδιότητες αντιπροσωπεύουν τα δεδομένα της κλάσης. Το behavior της κλάσης καθορίζεται από τις μεθόδους `accelerate`, `brake`, `getSpeed`, και `getDetails`.

- `accelerate`(double increment): Αυτή η μέθοδος αυξάνει την ταχύτητα του αυτοκινήτου κατά την τιμή του increment.
- `brake`(double decrement): Αυτή η μέθοδος μειώνει την ταχύτητα του αυτοκινήτου κατά την τιμή του decrement. Αν η ταχύτητα γίνει αρνητική, την επαναφέρει στο 0.
- `getSpeed`(): Αυτή η μέθοδος επιστρέφει την τρέχουσα ταχύτητα του αυτοκινήτου.
- `getDetails`(): Αυτή η μέθοδος επιστρέφει μια συμβολοσειρά με τις λεπτομέρειες του αυτοκινήτου.

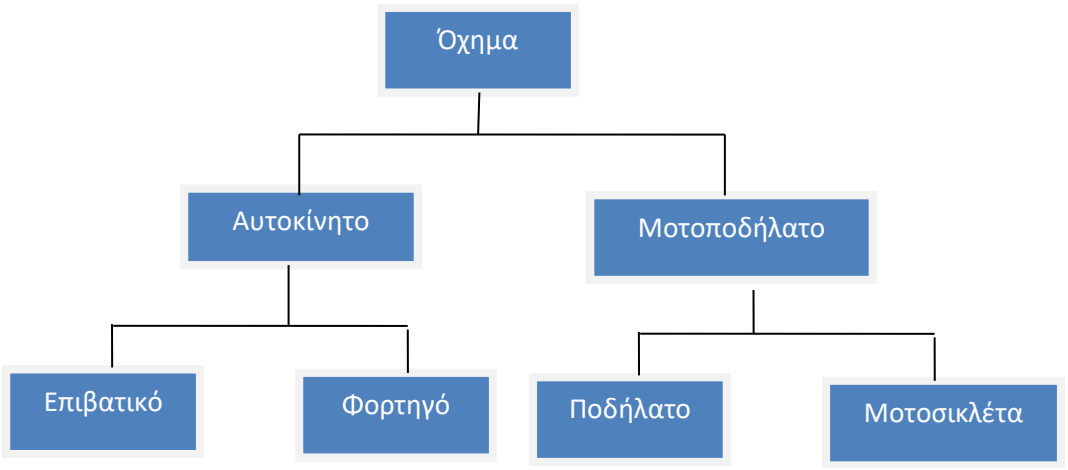
Σημασία του Behavior

Το behavior είναι κρίσιμο για την αντικειμενοστραφή προγραμματιστική προσέγγιση, καθώς επιτρέπει την ενθυλάκωση (encapsulation) και την απόκρυψη δεδομένων (data hiding). Οι μέθοδοι μιας κλάσης παρέχουν έναν ελεγχόμενο τρόπο αλληλεπίδρασης με τα δεδομένα της κλάσης, διασφαλίζοντας ότι οι αλλαγές στα δεδομένα γίνονται με έναν προβλέψιμο και ασφαλή τρόπο.

3.3.6. Χαρακτηριστικά του αντικειμενοστραφούς προγραμματισμού

Από το παραπάνω διαφαίνονται τα πιο σημαντικά χαρακτηριστικά του αντικειμενοστραφούς προγραμματισμού:

- **Encapsulation** (Ενθυλάκωση): Η υλοποίηση των μεθόδων ενός αντικειμένου κρύβονται από τον χρήστη. Ο χρήστης χρησιμοποιεί τις μεθόδους/συμπεριφορές των αντικειμένων για να ορίσει τα χαρακτηριστικά των αντικειμένων.
- **Inheritance** (κληρονομικότητα): Μπορούμε να δημιουργήσουμε κλάσεις από άλλες κλάσεις χρησιμοποιώντας την κληρονομικότητα. Η νέα κλάση έχει τα χαρακτηριστικά αλλά και τις μεθόδους της γονικής κλάσης ενώ μπορεί να τροποποιήσει τα χαρακτηριστικά και μεθόδους, να τα επεκτείνει και να προσθέσει καινούρια για να καλύψει συγκεκριμένες ανάγκες. Η κληρονομικότητα είναι μια μορφή επαναχρησιμοποίησης λογισμικού (software reuse). Για παράδειγμα μπορούμε να έχουμε την κλάση Όχημα ως γονέα, την κλάση Αυτοκίνητο και Μοτοποδήλατο που κληρονομούν την κλάση Όχημα και κατόπιν τις κλάσεις Επιβατικό και Φορτηγό που κληρονομούν από την κλάση Αυτοκίνητο.



Εικόνα 12 Ιεραρχία Κλάσεων

- **Polymorphism** (πολυμορφισμός): είναι μια έννοια που αφορά ιεραρχίες κλάσεων. Δίνει τη δυνατότητα το ίδιο όνομα μεθόδου να προκαλεί την εκτέλεση διαφορετικών εντολών ανάλογα με τον τύπο του αντικειμένου στο οποίο καλείται. Με αποτέλεσμα ο χρήστης να μην χρειάζεται να μαθαίνει νέες συμπεριφορές/μεθόδους σε ιεραρχίες κλάσεων.
- **Modularity**: Το κάθε αντικείμενο έχει τον δικό του πηγαίο κώδικα, ο οποίος γράφεται και διατηρείται ξεχωριστά από τον υπόλοιπο κώδικα της εφαρμογής.
- **Maintainability** (Συντήρηση-Διατηρησιμότητα): Αν κάποιο από τα αντικείμενα είναι προβληματικό μπορεί να αντικατασταθεί πολύ εύκολα από ένα άλλο αντικείμενο. Όπως και στον πραγματικό κόσμο αν χαλάσει μια βίδα, αλλάζουμε την βίδα όχι όλη την μηχανή.

3.4. Κλάσεις και Αντικείμενα

3.4.1. Δημιουργία Κλάσης

Όπως έχουμε ήδη αναφέρει η Java στηρίζεται στον αντικειμενοστραφή προγραμματισμό όπως η C++ και η πιο πρόσφατη C#. Η φιλοσοφία του νέου αυτού προγραμματιστικού στυλ είναι η: «Αποφάσισε ποιες κλάσεις χρειάζεσαι, όρισε ένα πλήρες σύνολο από λειτουργίες για την κάθε κλάση, έκφρασε ρητά τις κοινές λειτουργίες μέσω κληρονομικότητας».

Η κλάση αποτελεί το σχέδιο/καλούπι ενός αντικειμένου και ορίζει τα χαρακτηριστικά του αντικειμένου καθώς και τις μεθόδους–methods που αναπαριστούν την συμπεριφορά του αντικειμένου. Πολλά αντικείμενα μπορούν να δημιουργηθούν από μια κλάση. Τα αντικείμενα τα ονομάζουμε αλλιώς και στιγμιότυπα (instances) της κλάσης. Κάθε φορά που δημιουργούμε μία κλάση, στην ουσία είναι σαν να λέμε στον compiler: «Ορίζω έναν νέο τύπο δεδομένων με το τάδε όνομα που περιέχει τις εξής μεταβλητές μέλη και τις εξής μεθόδους».

Παράδειγμα κλάσης που ονομάζεται Student αντιπροσωπεύει την οντότητα Φοιτητής στο πανεπιστήμιο. Το αντικείμενο που δημιουργείται από την κλάση Student αντιστοιχεί σε έναν συγκεκριμένο Student του πανεπιστημίου για παράδειγμα τον Student Ανδρέα Πετρόπουλο. Η

κλάση Student αντιπροσωπεύει τη γενική έννοια ενός φοιτητή, και κάθε αντικείμενο που δημιουργείται από αυτή την κλάση αντιπροσωπεύει έναν πραγματικό φοιτητή που παρακολουθεί το πανεπιστήμιο. Σε ένα σύστημα Διαχείρισης Φοιτητών που διαθέτουν στις γραμματείες των Σχολών των πανεπιστημίων θα είχαμε την κλάση Student και χιλιάδες αντικείμενα Student.

Όπως έχουμε ήδη αναφέρει ένα αντικείμενο έχει μια κατάσταση (state), η οποία ορίζεται από τις τιμές που έχουν τα χαρακτηριστικά του. Για παράδειγμα, τα χαρακτηριστικά ενός Student μπορεί να είναι το όνομα, η διεύθυνση κατοικίας του φοιτητή, ο βαθμός του φοιτητή που αντιστοιχεί στο μέσο των βαθμών των μαθημάτων του και αναλυτικά οι βαθμοί στα μαθήματα που έχει ολοκληρώσει ο φοιτητής. Η κλάση Student καθορίζει ποια είναι τα χαρακτηριστικά του κάθε φοιτητή και κάθε αντικείμενο Student αποθηκεύει τις τιμές αυτών των χαρακτηριστικών για έναν συγκεκριμένο φοιτητή. Στην Java, τα χαρακτηριστικά ενός αντικειμένου καθορίζονται από μεταβλητές μέλη που δηλώνονται σε μια κλάση.

Ένα αντικείμενο έχει επίσης συμπεριφορές, οι οποίες ορίζονται από τις λειτουργίες που σχετίζονται με το αντικείμενο. Για παράδειγμα οι λειτουργίες ενός φοιτητή μπορεί να είναι η δυνατότητα ενημέρωσης της διεύθυνσης κατοικίας του φοιτητή και ο υπολογισμός του τρέχοντος βαθμού του που αντιστοιχεί στον μέσο βαθμό στα μαθήματα που έχει ολοκληρώσει ο φοιτητής. Η κλάση Student ορίζει τις λειτουργίες, όπως για παράδειγμα τις λεπτομέρειες για το πώς υπολογίζεται ο μέσος όρος βαθμών στα μαθήματα που έχει ολοκληρώσει ο φοιτητής. Αυτές οι λειτουργίες μπορούν στη συνέχεια να εκτελεστούν από ένα συγκεκριμένο αντικείμενο Student. Οι λειτουργίες ενός αντικειμένου συνήθως τροποποιούν την κατάσταση του αντικειμένου. Στην Java, οι λειτουργίες ενός αντικειμένου ορίζονται με μεθόδους που δηλώνονται στην κλάση.

Παρακάτω αναφέρονται παραδείγματα κλάσεων με ορισμένα χαρακτηριστικά και λειτουργίες τους. Για το ποια χαρακτηριστικά και ποιες λειτουργίες έχει μια κλάση εξαρτάται κάθε φορά από τον προγραμματιστή αλλά και από την ανάλυση απαιτήσεων της εφαρμογής που πρόκειται να δημιουργηθεί.

Κλάση	**Χαρακτηριστικά (μεταβλητές μέλη)**	**Λειτουργίες (μέθοδοι)**
Student	firstName lastName homeAddress averageGrade	Set Home Address (setHomeAddress) Compute Avergae Grade (computeAvGrade)
Employee	firstName lastName adt afm department salary	Set ADT (setAdt) Set Department (setDepartment) Set Salary (setSalary) Compute Bonus (computeBonus) Compute Taxes (computeTaxes)
Flight	Airline Destination city Arrival City Flight Number Status	Set Airline (setAirline) Set Flight Number (setFlightNumber) Set current Status (setStatus)

3.4.2. Δημιουργία Αντικειμένων

Πριν προχωρήσουμε στον εμπλουτισμό της κλάσης μας με μεθόδους, θα δούμε πως μπορούμε να δημιουργήσουμε αντικείμενα, θεωρώντας πως η κλάση είναι πλήρης. Για να δημιουργήσουμε ένα αντικείμενο κάποιας κλάσης η σύνταξη είναι:

```
ΟΝΟΜΑ_ΚΛΑΣΗΣ  όνομα_μεταβλητής = new  ΟΝΟΜΑ_ΚΛΑΣΗΣ();
```

Χρησιμοποιώντας το keyword **new** δημιουργείται ένα στιγμιότυπο (instance) της κλάσης. Η μεταβλητή που δημιουργήθηκε αποτελεί αναφορά του αντικειμένου της κλάσης. Για παράδειγμα:

```
Customer  refCustomer = new  Customer();
```

Στην παραπάνω εντολή λαμβάνουν χώρα τρείς διεργασίες. Στο αριστερό τμήμα της ισότητας δηλώνεται μία αναφορά τύπου **Customer** με το όνομα **refCustomer**. Στο δεξί τμήμα που αποτελεί την δεύτερη διεργασίας δημιουργείται το αντικείμενο της κλάσης. Η δημιουργία του αντικειμένου μιας κλάσης στην Java γίνεται με την κλίση ειδικών μεθόδων τους constructors που θα τους δούμε αναλυτικά σε επόμενη ενότητα. Κάθε κλάση έχει και τον ή τους constructors της. Θα δούμε σε επόμενη παράγραφο ότι ο constructor έχει πάντα το ίδιο όνομα με την κλάση. Αυτό που ουσιαστικά συμβαίνει στο παραπάνω παράδειγμα είναι πως ο compiler καλεί έμμεσα τον constructor της κλάσης **Customer** οπότε δεσμεύεται η κατάλληλη μνήμη και δημιουργείται το αντικείμενο. Τέλος η τρίτη διεργασία είναι η ανάθεση του αντικειμένου στην αναφορά **refCustomer**. Η αναφορά **refCustomer** που δείχνει τώρα στο αντικείμενο τύπου Customer μπορεί να χρησιμοποιηθεί για να έχουμε πρόσβαση στα χαρακτηριστικά και τις λειτουργίες της κλάσης οι οποίες όμως αναφέρονται σε ένα συγκεκριμένο αντικείμενο.

Θα πρέπει να σημειώσουμε ότι με τον παραπάνω τρόπο δηλώνουμε και δημιουργούμε μια αναφορά σε αντικείμενο με μια εντολή. Θα μπορούσαμε να δηλώσουμε πρώτα την μεταβλητή αναφοράς σε αντικείμενο τύπου Customer κατόπιν να δημιουργήσουμε το αντικείμενο καλώντας τον constructor της κλάσης και να αναθέσουμε την τιμή στην μεταβλητή αναφοράς. Στο παρακάτω παράδειγμα φαίνονται οι δύο τρόποι:

```
public static void main(String[] args) {
      //1ος τρόπος: Δηλώνουμε και δημιουργούμε το αντικείμενο της κλάσης
      Customer refcustomer1 = new Customer(); // declare and instatiate

      //2ος τρόπος: Δηλώνουμε και κατόπιν δημιουργούμε το αντικείμενο
      //της κλάσης
      Customer refcustomer2; //declare the reference
      refcustomer2 = new Customer(); //instatiate
}
```

3.5. Δημιουργία package - Ομαδοποίηση Κλάσεων

3.5.1. Packages στη Java

Στην Java, η ομαδοποίηση κλάσεων γίνεται μέσω των packages. Ένα package είναι ένας τρόπος να οργανώσουμε τις κλάσεις και τις διεπαφές μας σε λογικές ομάδες, διευκολύνοντας τη διαχείριση και την επαναχρησιμοποίηση του κώδικα.

Για να δημιουργήσουμε ένα package, ακολουθούμε τα εξής βήματα: Στην αρχή του αρχείου Java, δηλώνουμε το package στο οποίο ανήκει η κλάση. Η δήλωση γίνεται με τη λέξη-κλειδί `package` ακολουθούμενη από το όνομα του package.

```
package com.example.myapp;
```

Η δομή των φακέλων πρέπει να αντιστοιχεί στο όνομα του package. Για παράδειγμα, για το package `com.example.myapp`, η δομή των φακέλων θα είναι:

```
com/
    example/
        myapp/
            MyClass.java
```

Όταν μεταγλωττίζουμε την κλάση, η Java θα τοποθετήσει το αρχείο `.class` στον αντίστοιχο φάκελο. Για να εκτελέσουμε την κλάση, πρέπει να αναφέρουμε το πλήρες όνομα του package.

```
javac com/example/myapp/MyClass.java

java com.example.myapp.MyClass
```

Τα packages μας επιτρέπουν να αποφεύγουμε συγκρούσεις ονομάτων και να οργανώνουμε τον κώδικά μας με τρόπο που είναι εύκολος στη συντήρηση.

3.5.2. Import Statement

Σε περίπτωση που θέλουμε σε ένα package να εισάγουμε κλάσεις από άλλα packages, αυτό μπορεί να γίνει χρησιμοποιώντας τη λέξη-κλειδί `import`.

```
import com.example.myapp.MyClass;

public class Main {
    public static void main(String[] args) {
        MyClass myObject = new MyClass();
        // Χρήση του myObject
    }
}
```

3.6. Πρακτική Εξάσκηση Java Project στο eclipse

Θέμα: Δημιουργία ενός Java Project με Κλάσεις, Μεθόδους και Αντικείμενα στο Eclipse

Στόχος: Να εξοικειωθείτε με τη δημιουργία και χρήση κλάσεων, μεθόδων, μεταβλητών και αντικειμένων σε Java, καθώς και με τη διαχείριση πακέτων στο Eclipse.

Οδηγίες:

- Δημιουργία Project:
- Ανοίξτε το Eclipse IDE.
- Επιλέξτε File > New > Java Project.
- Ονομάστε το project σας AdvancedProject και πατήστε Finish.
- Δημιουργία Πακέτων και Κλάσεων:
- Κάντε δεξί κλικ στο src φάκελο του project σας και επιλέξτε New > Package.
- Ονομάστε το πακέτο com.example.advancedproject.
- Δημιουργήστε ένα δεύτερο πακέτο με όνομα com.example.advancedproject.model.
- Κάντε δεξί κλικ στο πακέτο com.example.advancedproject και επιλέξτε New > Class.
- Ονομάστε την κλάση Main και επιλέξτε το πλαίσιο public static void main(String[] args) για να δημιουργηθεί η μέθοδος main.
- Κάντε δεξί κλικ στο πακέτο com.example.advancedproject.model και επιλέξτε New > Class.
- Ονομάστε την κλάση Person.
- Υλοποίηση Κλάσης Person:
- Στην κλάση Person, προσθέστε τον παρακάτω κώδικα:

```
package com.example.advancedproject.model;

public class Person {
    private String name;
    private int age;

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }
```

```
    public void setAge(int age) {
        this.age = age;
    }

    public void printDetails() {
        System.out.println("Name: " + name + ", Age: " + age);
    }
}
```

- Υλοποίηση Κλάσης Main:
- Στην κλάση Main, προσθέστε τον παρακάτω κώδικα:

```
package com.example.advancedproject;

import com.example.advancedproject.model.Person;

public class Main {
    public static void main(String[] args) {
        Person person = new Person("John Doe", 30);
        person.printDetails();
    }
}
```

- Εκτέλεση Προγράμματος:
- Κάντε δεξί κλικ στην κλάση Main και επιλέξτε Run As > Java Application.
- Βεβαιωθείτε ότι το πρόγραμμα εκτυπώνει το μήνυμα “Name: John Doe, Age: 30” στην κονσόλα.

3.7. Ερωτήσεις Αυτο-αξιολόγησης

1. Τι χαρακτηρίζει κυρίως τον διαδικαστικό προγραμματισμό;

α) Επικεντρώνεται σε αντικείμενα

β) Οργανώνει τον κώδικα σε διαδικασίες και συναρτήσεις

γ) Υλοποιεί κλάσεις και αντικείμενα

δ) Οργανώνει τον κώδικα με βάση τις ιδιότητες

2. Στον δομημένο προγραμματισμό, τα δεδομένα:

α) Είναι οργανωμένα σε αντικείμενα

β) Είναι απομονωμένα σε διαφορετικά πακέτα

γ) Είναι διαθέσιμα σε όλο το πρόγραμμα

δ) Διαχειρίζονται μέσω διαδικασιών και συναρτήσεων

3. Ποιο από τα παρακάτω ΔΕΝ είναι χαρακτηριστικό του αντικειμενοστραφούς προγραμματισμού;

α) Ενθυλάκωση (Encapsulation)

β) Κληρονομικότητα (Inheritance)

γ) Επανάληψη (Iteration)

δ) Πολυμορφισμός (Polymorphism)

4. Μια κλάση στη Java είναι:

α) Ένα στιγμιότυπο δεδομένων

β) Ένας ορισμός ή πρότυπο για τη δημιουργία αντικειμένων

γ) Μια μέθοδος του αντικειμένου

δ) Μια στατική μεταβλητή

5. Πώς δημιουργείται ένα αντικείμενο από μια κλάση στη Java;

α) MyClass obj;

β) new MyClass obj;

γ) MyClass obj = new MyClass();

δ) MyClass obj = create MyClass();

6. Οι ιδιότητες μιας κλάσης είναι:

α) Μέθοδοι που ανήκουν στην κλάση

β) Μεταβλητές που περιγράφουν την κατάσταση ενός αντικειμένου

γ) Στατικές μεταβλητές

δ) Επανάληψη δεδομένων

7. Οι συμπεριφορές μιας κλάσης αναφέρονται:

α) Στα δεδομένα που αποθηκεύει

β) Στις μεθόδους που εκτελούν ενέργειες

γ) Στα αντικείμενα που δημιουργεί

δ) Στις ιδιότητες που κληρονομεί

8. Τι είναι ένα package στη Java;

α) Ένα λογισμικό που εγκαθιστούμε

β) Ένας τρόπος ομαδοποίησης κλάσεων

γ) Μια βιβλιοθήκη δεδομένων

δ) Ένα framework για προγραμματισμό

9. Γιατί χρησιμοποιούμε packages στη Java;

α) Για βελτιστοποίηση της ταχύτητας εκτέλεσης

β) Για οργάνωση και αποφυγή συγκρούσεων ονομάτων

γ) Για την αποθήκευση δεδομένων

δ) Για την αυτόματη δημιουργία κώδικα

10. Ποια είναι η σωστή σύνταξη για να καλέσετε μια μέθοδο σε ένα αντικείμενο;

α) method(obj);

β) obj.method();

γ) call obj.method();

δ) method.call(obj);

11. Ποιο από τα παρακάτω είναι έγκυρο όνομα κλάσης στη Java;

α) 1Class

β) My Class

γ) _MyClass

δ) class

12. Ποιο είναι το κύριο πλεονέκτημα του αντικειμενοστραφούς προγραμματισμού;

α) Αύξηση ταχύτητας εκτέλεσης

β) Ευκολία κατανόησης και συντήρησης

γ) Μείωση κώδικα

δ) Υποστήριξη περισσότερων γλωσσών

13. Ποια είναι η κύρια διαφορά μεταξύ κλάσης και αντικειμένου;

α) Οι κλάσεις εκτελούν κώδικα, τα αντικείμενα όχι

β) Οι κλάσεις είναι πρότυπα, τα αντικείμενα είναι στιγμιότυπα

γ) Τα αντικείμενα είναι στατικά, οι κλάσεις είναι δυναμικές

δ) Οι κλάσεις ανήκουν στα αντικείμενα

14. Ποια είναι η προεπιλεγμένη ορατότητα (visibility) των μελών μιας κλάσης στη Java;

α) public

β) protected

γ) private

δ) package-private

15. Ποια είναι η σωστή μορφή δήλωσης κατασκευαστή σε μια κλάση;

α) MyClass() {}

β) void MyClass() {}

γ) static MyClass() {}

δ) constructor MyClass() {}

16. Τι είναι ο πολυμορφισμός στον αντικειμενοστραφή προγραμματισμό;

α) Η δυνατότητα ενός αντικειμένου να αλλάζει τύπο

β) Η δυνατότητα μιας μεθόδου να παίρνει πολλές μορφές

γ) Η δημιουργία πολλών κλάσεων από ένα αντικείμενο

δ) Η δημιουργία αντικειμένων από πολλές κλάσεις

17. Ποιο από τα παρακάτω είναι χαρακτηριστικό της κληρονομικότητας;

α) Αναγκάζει όλες τις κλάσεις να είναι ίδιες

β) Επιτρέπει σε μια κλάση να κληρονομήσει τις ιδιότητες μιας άλλης

γ) Εμποδίζει την πρόσβαση σε μέλη της υπερκλάσης

δ) Δημιουργεί αντικείμενα χωρίς κλάση

18. Ποια είναι η σωστή μορφή εισαγωγής ενός package στη Java;

α) import packageName.*;

β) use packageName.*;

γ) include packageName.*;

δ) library packageName.*;

19. Τι κάνει η λέξη-κλειδί super στη Java;

α) Δημιουργεί ένα νέο αντικείμενο

β) Αναφέρεται σε μέλη της υπερκλάσης

γ) Καλεί στατική μέθοδο

δ) Επιστρέφει ένα package

20. Τι ορίζεται ως ενθυλάκωση στον αντικειμενοστραφή προγραμματισμό;

α) Η χρήση κατασκευαστών

β) Η προστασία των δεδομένων μέσω ιδιωτικής πρόσβασης

γ) Η κληρονομικότητα ιδιοτήτων

δ) Η υλοποίηση στατικών μεθόδων

3.8. Απαντήσεις στις ερωτήσεις Αυτο-αξιολόγησης

1. β	2. δ	3. γ	4. β	5. γ
6. β	7. β	8. β	9. β	10. β
11. γ	12. β	13. β	14. δ	15. α
16. β	17. β	18. α	19. β	20. β

ΚΕΦΑΛΑΙΟ 4: Έννοιες αντικειμενοστραφούς προγραμματισμού Μέρος Β

4.1. Εκπαιδευτικοί στόχοι του παρόντος κεφαλαίου

Οι εκπαιδευτικοί στόχοι του παρόντος κεφαλαίου, συνοψίζονται στα κάτωθι σημεία. Οι εκπαιδευόμενοι:

- θα μάθουν πως υλοποιούνται οι μέθοδοι (σύνταξη Μεθόδου, παράμετροι και Ορίσματα Μεθόδου, επιστρεφόμενοι Τύποι Μεθόδου, κλήση Μεθόδων, Scope Μεταβλητών),
- θα μάθουν πως υλοποιούνται οι δημιουργοί (Constructors) ,
- θα γνωρίσουν πως γίνεται η καταστροφή Αντικειμένων στην Java και πως λειτουργεί ο Garbage Collector,
- θα γνωρίσουν την έννοιας της υπερφόρτωσης Μεθόδων και Δημιουργών (Method and Constructor Overloading) και πώς χρησιμοποιείται,
- θα γνωρίσουν την έννοια της ενθυλάκωσης (Encapsulation),
- θα γνωρίσουν την ορατότητα των Μεθόδων (Modifiers),
- θα μάθουν τις Static Μεθόδους (Methods) και Μεταβλητές (Variables),
- θα γνωρίσουν πως υλοποιούνται οι Απαριθμητοί Τύποι (Enumerated Types) .

4.2. Μέθοδοι (methods)

4.2.1. Σύνταξη Μεθόδου

Μία μέθοδος αντιστοιχεί σε μία λειτουργία του αντικειμένου, τεχνικά είναι μία συνάρτηση που ανήκει σε μία κλάση και εκτελεί την λειτουργία του αντικειμένου. Μία κλάση περιέχει τις εξής κατηγορίες μεθόδων:

- Διαχειριστικές. Στην κατηγορία αυτή οι μέθοδοι παρέχουν διαχειριστικές λειτουργίες για το αντικείμενο. Σε αυτήν την κατηγορία ανήκουν για παράδειγμα οι constructors, οι getters/setters. Οι getters/setters δημιουργούνται για κάθε χαρακτηριστικό-field της κλάσης προκειμένου να έχουμε πρόσβαση στα χαρακτηριστικά-fields της κλάσης. Με την λογική αυτή ακολουθείται η έννοια του Encapsulation σύμφωνα με την οποία ο χρήστης χρησιμοποιεί τις μεθόδους/συμπεριφορές των αντικειμένων για να έχει πρόσβαση στα χαρακτηριστικά των αντικειμένων. Κρύβεται δηλαδή από τον χρήστη η απευθείας πρόσβαση στα fields της κλάσης και μειώνεται η πιθανότητα εισαγωγής μη επιτρεπτών τιμών στα χαρακτηριστικά αφού οι setters ελέγχουν την εγκυρότητα των τιμών που εισάγονται σε κάθε χαρακτηριστικό της κλάσης.
- Μεθόδους που καθορίζουν τη συμπεριφορά του αντικειμένου.
- Μεθόδους που παρέχουν βοηθητικές λειτουργίες, π.χ. που υπολογίζουν κάποιο χαρακτηριστικό του αντικειμένου το οποίο μπορεί να χρησιμοποιηθεί για κάποιο σκοπό.

Γενικά οι μέθοδοι είναι ένα σύνολο εντολών με κάποιο όνομα που μπορεί να κληθούν κατά βούληση χρησιμοποιώντας το όνομα αυτό. Η χρήση μεθόδων στα προγράμματά μας προσφέρει πολλά πλεονεκτήματα. Το βασικότερο από αυτά είναι η αποφυγή επανάληψης του ίδιου κώδικα. Γενικά η σύνταξη της μεθόδου είναι:

```
Modifier Τύπος_Δεδομένων_Επιστροφής Όνομα_Μεθόδου(
[Τύπος_Δεδομένων_Παραμέτρου Όνομα_ Παραμέτρου, …]) {
       //σύνολο εντολών
          return Τιμή_ή_Μεταβλητή_Τύπου_Δεδομένων_Επιστροφής;
}
```

Ο **Modifier** των μεθόδων ανάλογα με τους modifiers των χαρακτηριστικών-fields της κλάσης δηλώνει την ορατότητα της μεθόδου από τις άλλες κλάσεις. Η Java υποστηρίζει τους εξής modifiers:

- public: η μέθοδος είναι ορατή από όλες τις κλάσεις τους java προγράμματος
- private: η μέθοδος είναι ορατή μόνο μέσα στην ίδια κλάση, δηλαδή από μεθόδους της κλάσης.
- default (χωρίς τιμή ο modifier): η μέθοδος είναι ορατή μόνο από κλάσεις του ίδιου package.

Ο **Τύπος_Δεδομένων_Επιστροφής** αντιστοιχεί στους Τύπους Δεδομένων είτε αυτοί είναι βασικοί τύπο (int, float, double, char κ.α.), είτε πιο πολύπλοκοι τύποι π.χ. πίνακες int[], είτε ακόμα τύποι κλάσεων που εμείς έχουμε δημιουργήσει ή ολόκληρες δομές δεδομένων π.χ. τύπου πίνακας σε αντικείμενα κλάσεων που έχεουμε δημιουργήσει. Ο Τύπος_Δεδομένων_Επιστροφής στην περίπτωση που η μέθοδος δεν επιστρέφει κάτι είναι **void.**

Το **Όνομα_Μεθόδου** είναι ακριβώς το όνομα της μεθόδου το οποίο χρησιμοποιείται και για την κλήση της μεθόδου. Εντός των παρενθέσεων δηλαδή **[Τύπος_Δεδομένων_ Παραμέτρου Όνομα_ Παραμέτρου, …]** μια μέθοδος μπορεί να έχει ένα πλήθος παραμέτρων **(parameters)** χωριζόμενα με κόμμα (,) ή κανένα όρισμα. Η κάθε **παράμετρος (parameter)** κατά την δήλωση της μεθόδου δηλώνεται με τον Τύπο Δεδομένων της και το όνομά της. Το σώμα της μεθόδου ξεκινά από το άγκιστρο **{** και τελειώνει στο άγκιστρο **}**. Όταν ο Τύπος_Δεδομένων_Επιστροφής δεν είναι void τότε η τελευταία εντολή της μεθόδου είναι η **return** που επιστρέφει μία τιμή που πρέπει να αντιστοιχεί σε τύπο δεδομένων ίδιο με τον Τύπο_Δεδομένων_Επιστροφής. Η τιμή αυτή του return είναι ακριβώς η τιμή που επιστρέφει η μέθοδος.

4.2.2. Παράμετροι και Ορίσματα Μεθόδου (Method Arguments and Parameters)

Παράμετροι είναι οι μεταβλητές που δηλώνονται στη δήλωση μιας μεθόδου και χρησιμοποιούνται για να δεχτούν τιμές όταν η μέθοδος καλείται. Ορίσματα είναι οι πραγματικές τιμές που περνιούνται στη μέθοδο όταν καλείται.

Παράδειγμα Μεθόδου με **Παραμέτρους**

```
public class Calculator {
    // Μέθοδος με δύο παραμέτρους
    public int add(int a, int b) {
        return a + b;
```

```
    }
}
```
Στο παραπάνω παράδειγμα, a και b είναι οι παράμετροι της μεθόδου add.

Κλήση Μεθόδου με **Ορίσματα**

```
public class Main {
    public static void main(String[] args) {
        Calculator calc = new Calculator();
        int result = calc.add(5, 3); // 5 και 3 είναι τα ορίσματα
        System.out.println("Result: " + result);
    }
}
```
Στην κλήση της μεθόδου add, τα 5 και 3 είναι τα ορίσματα που περνιούνται στις παραμέτρους a και b.

4.2.3. Παραδείγματα μεθόδων

Παραδείγματα μεθόδων παρουσιάζονται στα μέρη κώδικα που ακολουθεί.

```
public void displayCustomerInfo() {
            //σύνολο εντολών
}
```
Η παραπάνω μέθοδος είναι ορατή από όλες τις κλάσεις, δεν επιστρέφει καμμία τιμή. Το όνομά της είναι displayCustomerInfo και δεν έχει κάποιο όρισμα. Το σώμα της μεθόδου ξεκινά από το άγκιστρο { και τελιώνει στο άγκιστρο }.

```
private void setAddress(String newAddress){
             //σύνολο εντολών
}
```
Η παραπάνω μέθοδος είναι ορατή μόνο από την κλάση στην οποία ανήκει, δεν επιστρέφει καμμία τιμή. Το όνομά της είναι setAddress και έχει ως παράμετρο την newAdress τύπου String. Η μέθοδος αυτή ενημερώνει το field address της κλάσης. Το σώμα της μεθόδου ξεκινά από το άγκιστρο { και τελιώνει στο άγκιστρο }.

```
public int add(int x, int y) {
           return x+y;
}
```
Η παραπάνω μέθοδος υπολογίζει και επιστρέφει το άθροισμα δύο ακεραίων αριθμών. Είναι ορατή από οποιαδήποτε κλάση και το όνομά της είναι add και έχει ως παραμέτρους τους 2 ακέραιους x και y των οποίων το άθροισμα υπολογίζει. Το σώμα της μεθόδου ξεκινά από το άγκιστρο { και τελιώνει στο άγκιστρο }. Στην εντολή return η οποία αποτελεί πάντα την τελευταία εντολή μιας μεθόδου που επιστρέφει τιμή, εκτιμάται το άθροισμα x+y και επιστρέφεται η τιμή του αθροίσματος.

Πολλές φορές οι όροι παράμετρος και όρισμα δημιουργούν σύγχυση κατά την δημιουργία του προγράμματος. Γενικά:

Παράμετρος είναι η μεταβλητή που ορίζεται κατά την δήλωση της μεθόδου. Για παράδειγμα στην παρακάτω μέθοδο, οι μεταβλητές x και y ονομάζονται παράμετροι της μεθόδου multiply:

```
public int multiply(int x, int y) {
      return x*y;
}
```
Όρισμα είναι η τιμή που περνάμε όταν καλούμε την μέθοδο. Για παράδειγμα στην κλήση της μέθοδου multiply οι μεταβλητές a και b ονομάζονται ορίσματα της μεθόδου.

```
public static void main(String[] args) {
           int a = 10;
           int b = 10;
           int result;
           Actions actions = new Actions();
           result = actions.multiply(a, b);
}
```
Κατά την κλήση των μεθόδων που έχουν ορίσματα (arguments) στην Java ακολουθούνται δύο διαφορετικοί μηχανισμοί κλήσης οι οποίοι εξαρτώνται από τους Τύπους Δεδομένων των παραμέτρων των μεθόδων:

- Ο μηχανισμός **κλήσης κατ' αξία** (Call by Value) όταν ο τύπος δεδομένων των παραμέτρων των μεθόδων είναι βασικός τύπος δηλαδή int, double, float, char κ.α.. Κατά την κλήση της μεθόδου οι τιμές των ορισμάτων αντιγράφονται σε άλλες μεταβλητές (αποθηκεύονται σε άλλη θέση μνήμης) και αυτές οι μεταβλητές χρησιμοποιούνται στο σώμα της συνάρτησης. Το αποτέλεσμα είναι οι μεταβλητές των ορισμάτων να παραμένουν με τις αρχικές τους τιμές και μετά την κλήση των μεθόδων. Για παράδειγμα:
```
public int calculate(int x, int y) {
       x = 20;
       y=20;
       return x+y;
}
 public static void main(String[] args) {
       int a = 10;
       int b = 10;
       int result;
       Actions actions = new Actions();
       result = actions.calculate(a, b);
       System.out.println("a = "+a+" b  = "+b);
}
```
Το αποτέλεσμα θα είναι: a=10 b=10

- Ο μηχανισμός **κλήσης κατ' αναφορά** (Call by Reference) όταν οι παράμετροι των μεθόδων είναι αναφορές σε Αντικείμενα οποιουδήποτε τύπου ή σε πίνακες. Στην περίπτωση αυτή αν οι μεταβλητές ορίσματα της μεθόδου τροποποιούνται μέσα στο σώμα της μεθόδου, οι τιμές αυτές των μεταβλητών διατηρούνται και μετά το πέρας της κλήσης της μεθόδου. Για παράδειγμα:
```
public void setNames(Customer customer) {
          customer.firstname = "ΜΑΡΙΑ";
          customer.lastname = "ΑΝΔΡΕΟΥ";
      }
      public static void main(String[] args) {
          Customer tcustomer = new Customer();
          tcustomer.firstname = "ΓΙΩΡΓΟΣ";
          tcustomer.lastname = "ΠΑΠΑΙΩΑΝΝΟΥ";
```

```
                tcustomer.setNames(tcustomer) ;
                System.out.println("firstname = "+tcustomer.firstname
                                +"lastname  = "+ tcustomer.lastname);
            }
    Το αποτέλεσμα θα είναι:
    firstname = ΜΑΡΙΑ lastname  = ΑΝΔΡΕΟΥ
```

4.2.4. Επιστρεφόμενοι Τύποι Μεθόδου (Method Return Types)

Ο επιστρεφόμενος τύπος μιας μεθόδου καθορίζει το είδος της τιμής που η μέθοδος θα επιστρέψει όταν ολοκληρωθεί η εκτέλεσή της. Ο επιστρεφόμενος τύπος δηλώνεται πριν από το όνομα της μεθόδου στη δήλωση της μεθόδου.

Παράδειγμα Μεθόδου με Επιστρεφόμενο Τύπο

```
public class Calculator {
    // Μέθοδος που επιστρέφει έναν ακέραιο
    public int add(int a, int b) {
        return a + b;
    }
}
```

Στο παραπάνω παράδειγμα, η μέθοδος add έχει επιστρεφόμενο τύπο int, που σημαίνει ότι επιστρέφει έναν ακέραιο αριθμό. Οι τύποι Επιστρεφόμενων Τιμών μπορεί να είναι:

- Πρωτογενείς Τύποι: int, float, double, char, boolean, κλπ.
- Αντικείμενα: Οποιοδήποτε αντικείμενο κλάσης, όπως String, ArrayList, κλπ.
- Πίνακες: Πίνακες οποιουδήποτε τύπου, όπως int[], String[], κλπ.
- Void: Χρησιμοποιείται όταν η μέθοδος δεν επιστρέφει καμία τιμή.

Παράδειγμα Μεθόδου που Επιστρέφει Αντικείμενο

```
public class Person {
    private String name;

    public Person(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }
}
```

Στο παραπάνω παράδειγμα, η μέθοδος getName επιστρέφει ένα αντικείμενο τύπου String.

Οι μέθοδοι με επιστρεφόμενο τύπο `void` δεν επιστρέφουν καμία τιμή. Χρησιμοποιούνται συνήθως για εκτέλεση ενεργειών.

```
public class Printer {
    public void printMessage(String message) {
        System.out.println(message);
    }
```

```
}
```

Επιστροφή Τιμών από Μεθόδους

Για να επιστρέψουμε μια τιμή από μια μέθοδο, χρησιμοποιούμε τη λέξη-κλειδί `return` ακολουθούμενη από την τιμή που θέλεις να επιστρέψεις. Παράδειγμα Επιστροφής Τιμής είναι το παρακάτω.

```
public class Calculator {
    public int multiply(int a, int b) {
        return a * b;
    }
}
```

Επιστροφή Πολλαπλών Τιμών

Η Java δεν υποστηρίζει άμεσα την επιστροφή πολλαπλών τιμών από μια μέθοδο. Ωστόσο, μπορείς να χρησιμοποιήσεις αντικείμενα ή πίνακες για να επιστρέψεις πολλαπλές τιμές. Παράδειγμα Επιστροφής Πολλαπλών Τιμών με Αντικείμενο ακολουθεί.

```
public class Result {
    private int sum;
    private int product;

    public Result(int sum, int product) {
        this.sum = sum;
        this.product = product;
    }

    public int getSum() {
        return sum;
    }

    public int getProduct() {
        return product;
    }
}

public class Calculator {
    public Result calculate(int a, int b) {
        int sum = a + b;
        int product = a * b;
        return new Result(sum, product);
    }
}
```

4.2.5. Κλήση Μεθόδων

Παραδείγματα της κλήσης των μεθόδων που παρουσιάσαμε στο προηγούμενο κεφάλαιο ακολουθούν.

```
public class Main {
    public static void main(String[] args) {
        Calculator calc = new Calculator();
```

```
        int result = calc.multiply(5, 3); // Κλήση της μεθόδου multiply
        System.out.println("Result of multiplication: " + result); //
Εκτυπώνει: Result of multiplication: 15
    }
}
```

Κλήση της Μεθόδου calculate και Χρήση του Αντικειμένου Result

```
public class Main {
    public static void main(String[] args) {
        Calculator calc = new Calculator();
        Result res = calc.calculate(5, 3); // Κλήση της μεθόδου calculate

        // Χρήση των μεθόδων getSum και getProduct του αντικειμένου Result
        System.out.println("Sum: " + res.getSum()); // Εκτυπώνει: Sum: 8
        System.out.println("Product: " + res.getProduct()); // Εκτυπώνει:
Product: 15
    }
}
```

4.2.6. Scope Μεταβλητών

Το scope (πεδίο) των μεταβλητών στη Java καθορίζει το μέρος του κώδικα όπου η μεταβλητή είναι ορατή και μπορεί να χρησιμοποιηθεί. Υπάρχουν διάφορα επίπεδα scope που πρέπει να γνωρίζουμε. Αυτά είναι:

- Τοπικό Scope (Local Scope)
- Scope Μεταβλητών Κλάσης (Class Scope)
- Scope Μεταβλητών Πακέτου (Package Scope)
- Scope Μεταβλητών Μεθόδου (Method Scope)

Τοπικό Scope (Local Scope)

Οι τοπικές μεταβλητές δηλώνονται μέσα σε μια μέθοδο, κατασκευαστή ή μπλοκ κώδικα και είναι ορατές μόνο μέσα σε αυτό το μπλοκ. Παράδειγμα Τοπικού Scope

```
public class Example {
    public void myMethod() {
        int localVar = 10; // Τοπική μεταβλητή
        System.out.println(localVar); // Ορατή μόνο μέσα στη μέθοδο
    }
}
```

Scope Μεταβλητών Κλάσης (Class Scope)

Οι μεταβλητές κλάσης (ή πεδία) δηλώνονται μέσα σε μια κλάση αλλά έξω από οποιαδήποτε μέθοδο. Είναι ορατές σε όλες τις μεθόδους της κλάσης. Παράδειγμα Scope Μεταβλητών Κλάσης

```
public class Example {
    private int classVar; // Μεταβλητή κλάσης

    public void myMethod() {
        classVar = 10; // Ορατή σε όλες τις μεθόδους της κλάσης
```

```
    }

    public void anotherMethod() {
        System.out.println(classVar); // Ορατή και εδώ
    }
}
```

Scope Μεταβλητών Πακέτου (Package Scope)

Οι μεταβλητές πακέτου είναι ορατές σε όλες τις κλάσεις του ίδιου πακέτου. Αυτό επιτυγχάνεται όταν δεν χρησιμοποιείται κανένας τροποποιητής πρόσβασης (default access modifier). Παράδειγμα Scope Μεταβλητών Πακέτου.

```
class Example {
    int packageVar; // Μεταβλητή πακέτου

    public void myMethod() {
        packageVar = 10; // Ορατή σε όλες τις κλάσεις του ίδιου πακέτου
    }
}
```

Scope Μεταβλητών Μεθόδου (Method Scope)

Οι μεταβλητές μεθόδου είναι οι παράμετροι που δηλώνονται στη δήλωση της μεθόδου και είναι ορατές μόνο μέσα στη μέθοδο. Παράδειγμα Scope Μεταβλητών Μεθόδου.

```
public class Example {
    public void myMethod(int methodVar) { // Μεταβλητή μεθόδου
        System.out.println(methodVar); // Ορατή μόνο μέσα στη μέθοδο
    }
}
```

Σκιώδεις Μεταβλητές (Shadowing)

Όταν μια τοπική μεταβλητή έχει το ίδιο όνομα με μια μεταβλητή κλάσης, η τοπική μεταβλητή “σκιάζει” τη μεταβλητή κλάσης μέσα στο scope της.

Ζωή των Μεταβλητών

Οι τοπικές μεταβλητές δημιουργούνται και καταστρέφονται κάθε φορά που η μέθοδος καλείται και ολοκληρώνεται, ενώ οι μεταβλητές κλάσης ζουν όσο ζει το αντικείμενο.

Παράδειγμα Σκιώδους Μεταβλητής

```
public class Example {
    private int var = 5; // Μεταβλητή κλάσης

    public void myMethod() {
        int var = 10; // Τοπική μεταβλητή που σκιάζει τη μεταβλητή κλάσης
        System.out.println(var); // Εκτυπώνει 10
    }

    public void anotherMethod() {
        System.out.println(var); // Εκτυπώνει 5
    }
}
```

4.3. Δημιουργοί (Constructors)

Ο constructor είναι μία ειδική μέθοδος της κλάσης η οποία είναι υπεύθυνη για τη δημιουργία των αντικειμένων της κλάσης. Ουσιαστικά ο Constructor δεσμεύει την απαραίτητη μνήμη στο σύστημα για το αντικείμενο. Καλείται χρησιμοποιώντας το keyword new και το Όνομα της Κλάσης. Ο Constructor πάντα έχει το ίδιο όνομα με την κλάση. Για παράδειγμα:

```
Customer myCustomer = new Customer();
```

Στο παράδειγμα, καλείται ο constructor της κλάσης Customer για να δημιουργήσει ένα αντικείμενο και η αναφορά του αντικειμένου που δημιουργήθηκε αποθηκεύεται στην μεταβλητή αναφοράς myCustomer.

Η δήλωση του constructor μέσα στην κλάση έχει την σύνταξη:

```
public Όνομα_Κλασης([Τύπος_Δεδομένων Παράμετρος, …]) {
      //περιέχει εντολές
}
```

Κάθε κλάση θα πρέπει να περιέχει τουλάχιστον έναν Constructor. Ένας constructor μπορεί όπως οι κοινές μέθοδοι να έχει παραμέτρους (**([Τύπος_Δεδομένων Παράμετρος, …]**) ή να μην έχει καμμία παράμετρο. Για παράδειγμα:

```
public class Customer {
   private String firstname;
   private String lastname;

      //constructors
      public Customer() {
         firstname = null;
         lastname = null;
       }
       public Customer(String fname, String lname) {
              firstname = fname;
              lastname = lname;
        }
        //methods
        public void displayInfo() {
             System.out.println("firstname = "+ firstname+"lastname   = "+
lastname);
        }
}//end class
```

Στο παραπάνω παράδειγμα η κλάση Customer έχει δύο constructors. Ο πρώτος δεν έχει παραμέτρους και ο δεύτερος έχει δύο παραμέτρους. Γενικά σκοπός των constructors είναι η δημιουργία των αντικειμένων και η αρχικοποίηση των χαρακτηριστικών τους. Γι' αυτό οι constructors είναι το μέρος που τοποθετούμε τον κώδικα αρχικοποίησης του αντικειμένου, όπως στο παράδειγμα του δεύτερου constructor. Λόγω του ότι η ύπαρξη ενός τουλάχιστον constructor είναι υποχρεωτική για κάθε κλάση.

Ώστε να μπορεί να δημιουργήσει αντικείμενα, η Java χρησιμοποιεί έναν μηχανισμό που εξασφαλίζει ότι σε κάθε κλάση θα υπάρχει σίγουρα ένας constructor. Σύμφωνα με αυτόν τον μηχανισμό, αν σε μία κλάση δεν ορίσει ο προγραμματιστής κάποιον constructor, ο compiler θα

δημιουργήσει έναν από μόνος του, ακόμη κι αν δεν είναι ορατός σε εμάς ο κώδικας του. Ο constructor που δημιουργεί ο compiler είναι ο default constructor έχει την δήλωση του πρώτου constructor στο παράδειγμά μας και δεν περιλαμβάνει καμία εντολή στο σώμα του.

Έστω μια παραλλαγή του προηγούμενου παραδείγματος:

```
public class Customer {
   private String firstname;
   private String lastname;

      //constructors
      public Customer() {
          this.firstname = null;
          this.lastname = null;
      }
      public Customer(String fname, String lname) {
          this.firstname = fname;
          this.lastname = lname;
          this.displayInfo();   //κλήση της μεθόδου με το this
          displayInfo(); //κλήση της μεθόδου
      }
      //methods
      public void displayInfo() {
           System.out.println("firstname = "+ this.firstname+"lastname   =
"+ this.lastname);
      }
}//end class
```

Στην παραπάνω παραλλαγή της κλάσης Customer εμφανίζεται ένας νέος τελεστή το **this**. Ο τελεστής **this** χρησιμοποιείται ως αναφορά στο τρέχον αντικείμενο της κλάσης. Για να αναφερθούμε σε ένα χαρακτηριστικό-field του αντικειμένου χρησιμοποιούμε τον τελεστή this ως εξής:

```
this.firstname = "ΝΙΚΟΣ";
```

Στις περιπτώσεις που έχουμε σε μια μέθοδο τοπικές μεταβλητές με το ίδιο όνομα με τα χαρακτηριστικά μιας κλάσης, για να έχουμε πρόσβαση στα χαρακτηριστικά του αντικειμένου της κλάσης χρησιμοποιούμε τον τελεστή this.

```
public class Customer {
   private String firstname = null;

   public void displayInfo() {
            String firstname = "ΜΑΡΙΑ"; //τοπική μεταβλητή
            this.firstname = "ANNA"; //χαρακτηριστικό αντικειμενου
            System.out.println("Local     firstname="+firstname+"     Object
firstname="+this.firstname);
      }
}
```

Για να αναφερθούμε σε μια μέθοδο του αντικειμένου μπορούμε επίσης να χρησιμοποιήσουμε τον τελεστή this ως εξής:

```
this.displayInfo(); //κλήση της μεθόδου με το this
```

Στην παραπάνω παραλλαγή της κλάσης Customer η displayInfo μέθοδος καλείται δύο (2) φορές.

4.4. Καταστροφή Αντικειμένων - Λειτουργία του Garbage Collector

Σε άλλες γλώσσες προγραμματισμού όπως η C, C++ αποτελεί ευθύνη του προγραμματιστή όταν ένα αντικείμενο δεν χρησιμοποιείται να ελευθερώνεται ο χώρος της μνήμης που αυτό καταλαμβάνει. Αντίθετα στην Java η απελευθέρωση της μνήμης δεν αποτελεί λειτουργία που θα πρέπει να εκτελέσει ο προγραμματιστής. Η Java διαθέτει αυτόματο μηχανισμό απελευθέρωσης της μνήμης γνωστός ως Garbage Collector. Όταν ένα αντικείμενο έχει ολοκληρώσει τη διάρκεια ζωής του, ή όταν έχει χάσει όλες τις αναφορές σε αυτό, τότε μαρκάρεται ως υποψήφιο για διαγραφή. Το αντικείμενο παραμένει στην μνήμη του υπολογιστή μέχρι να δράσει ο Garbage Collector.

Ο Garbage Collector είναι μια διεργασία/πρόγραμμα το οποίο εκτελείται ανά τακτά χρονικά διαστήματα και "συλλέγει" όλα τα μαρκαρισμένα προς διαγραφή αντικείμενα και επιστρέφει τη μνήμη τους στο σύστημα για μελλοντική χρήση. Όπως έχουμε ήδη αναφέρει η δέσμευση της μνήμης πραγματοποιείται από τους Constructors.

4.5. Υπερφόρτωση Μεθόδων και Δημιουργών (Method and Constructor Overloading)

4.5.1. Υπερφόρτωση Μεθόδων (Method Overloading)

Ένα από τα πιο ισχυρά πλεονεκτήματα που διαθέτει η Java είναι η δυνατότητα **υπερφόρτωσης μεθόδων (Method Overloading)**. Η έννοια της υπερφόρτωσης σημαίνει ότι στην Java μια κλάση μπορεί να περιλαμβάνει διαφορετικές μεθόδους που έχουν το ίδιο όνομα αλλά οι μέθοδοι αυτοί διαφοροποιούνται μεταξύ τους ως προς:

- Τον αριθμό των παραμέτρων
- Των τύπο δεδομένων των παραμέτρων
- Την σειρά των παραμέτρων

Για να καταλάβετε γιατί η συγκεκριμένη δυνατότητα είναι χρήσιμη, υποθέστε πως έχουμε το εξής πρόβλημα. Θέλουμε να γράψουμε μία μέθοδο που θα υπολογίζει το άθροισμα δύο αριθμών που θα τους λαμβάνει ως παραμέτρους. Η μέθοδος αυτή θέλουμε να λειτουργεί με όλους τους βασικούς τύπους που αντιστοιχούν σε αριθμό, δηλαδή int, float, double. Ξεκινώντας να γράφαμε τον κώδικα θα δημιουργούσαμε τρεις διαφορετικές μεθόδους με περίπου τον ίδιο κώδικα αλλά με διαφορετικό όνομα δηλαδή:

```
public class Computer {
	public int addInt(int x, int y) {
		return x+y;
	}
	public float addFoat(float x, float y) {
		return x+y;
	}
	public double addDouble(double x, double y) {
		return x+y;
	}
```

```
}
```

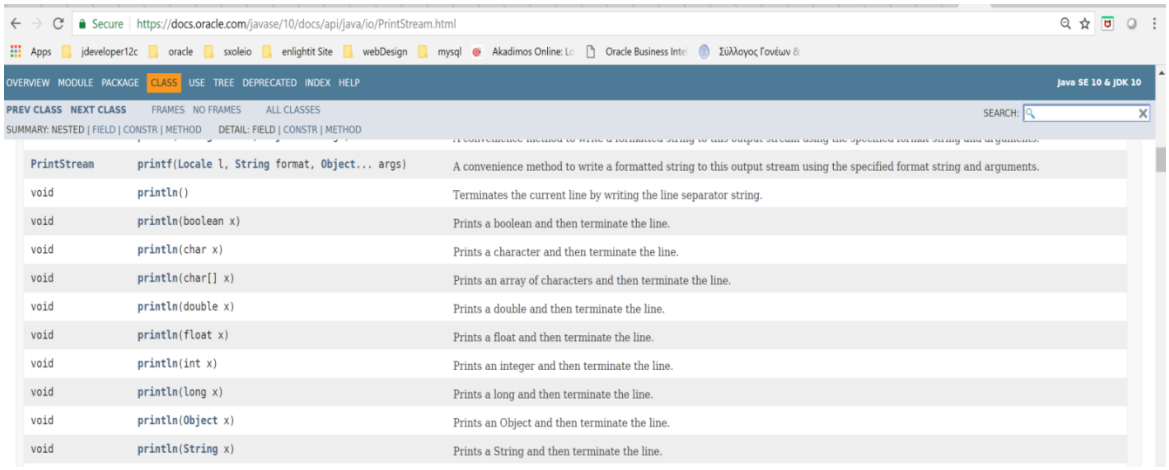
Στην πραγματικότητα δημιουργώντας τρεις διαφορετικές μεθόδους επαναλαμβάνεται ο ίδιος κώδικας ενώ το πρόγραμμά μας γεμίζει με μεθόδους που έχουν διαφορετικά ονόματα ενώ κάνουν την ίδια δουλειά.

Ακολουθώντας της έννοια της υπερφόρτωσης που υποστηρίζει η Java θα δημιουργούσαμε τρεις μεθόδους με το ίδιο όνομα αλλά διαφορετικό signature. Signature είναι το τμήμα εκείνο της μεθόδου που αποτελείται από το όνομα της μεθόδου και τις παραμέτρους της. Η κλάση Computer θα γίνει:

```
public class Computer {
	public int add(int x, int y) {
		return x+y;
	}
	public float add(float x, float y) {
		return x+y;
	}
	public double add(double x, double y) {
		return x+y;
	}
}
```

Η χρήση της υπερφόρτωσης μεθόδων έχει σαν αποτέλεσμα την ομαδοποίηση μεθόδων που εκτελούν την ίδια διεργασία.

Η έννοια της υπερφόρτωσης ακολουθείται κατά κόρον στην Java. Για παράδειγμα αν δούμε το Java API για την μέθοδο println αυτό είναι:



Εικόνα 13 Η έννοια της υπερφόρτωσης στην println (https://docs.oracle.com/javase/10/docs/api/index.html?overview-summary.html)

Υπάρχουν δέκα μεθόδοι με το όνομα println αλλά με διαφορετικές παραμέτρους.

4.5.2. Υπερφόρτωση Δημιουργών (Constructor Overloading)

Η υπερφόρτωση δημιουργών (constructor overloading) είναι μια τεχνική στην οποία μια κλάση μπορεί να έχει περισσότερους από έναν δημιουργούς (constructors) με διαφορετικές λίστες παραμέτρων. Αυτό επιτρέπει τη δημιουργία αντικειμένων της κλάσης με διαφορετικούς τρόπους.

Η υπερφόρτωση δημιουργών επιτρέπει την παροχή πολλαπλών τρόπων για την αρχικοποίηση ενός αντικειμένου. Κάθε δημιουργός πρέπει να έχει διαφορετική λίστα παραμέτρων (διαφορετικό αριθμό ή τύπο παραμέτρων).

Το παρακάτω παράδειγμα είναι ενδεικτικό της υπερφόρτωσης δημιουργών.

```
public class Box {
    private double width;
    private double height;
    private double depth;

    // Δημιουργός χωρίς παραμέτρους
    public Box() {
        width = 0;
        height = 0;
        depth = 0;
    }

    // Δημιουργός με μία παράμετρο
    public Box(double side) {
        width = side;
        height = side;
        depth = side;
    }

    // Δημιουργός με τρεις παραμέτρους
    public Box(double width, double height, double depth) {
        this.width = width;
        this.height = height;
        this.depth = depth;
    }

    public double volume() {
        return width * height * depth;
    }
}
```

Έχοντας το παραπάνω, μπορούμε να δημιουργήσουμε αντικείμενα της κλάσσης χρησιμοποιώντας οποιονδήποτε από τους υπερφορτωμένους δημιουργούς όπως φαίνεται και στο παράδειγμα που ακολουθεί.

```
public class Main {
    public static void main(String[] args) {
        // Χρήση του δημιουργού χωρίς παραμέτρους
        Box box1 = new Box();
        System.out.println("Volume of box1: " + box1.volume()); //
Εκτυπώνει: Volume of box1: 0.0

        // Χρήση του δημιουργού με μία παράμετρο
        Box box2 = new Box(5);
        System.out.println("Volume of box2: " + box2.volume()); //
Εκτυπώνει: Volume of box2: 125.0
```

```
        // Χρήση του δημιουργού με τρεις παραμέτρους
        Box box3 = new Box(2, 3, 4);
        System.out.println("Volume of box3: " + box3.volume()); // Εκτυπώνει: Volume of box3: 24.0
    }
}
```

Τα πλεονεκτήματα της Υπερφόρτωσης Δημιουργών είναι:

- Ευελιξία: Επιτρέπει τη δημιουργία αντικειμένων με διαφορετικούς τρόπους, ανάλογα με τις ανάγκες.
- Αναγνωσιμότητα: Κάνει τον κώδικα πιο κατανοητό και ευανάγνωστο, καθώς οι δημιουργοί μπορούν να έχουν περιγραφικές λίστες παραμέτρων.
- Επαναχρησιμοποίηση Κώδικα: Μπορείς να καλέσεις έναν δημιουργό από έναν άλλο χρησιμοποιώντας τη λέξη-κλειδί this, μειώνοντας την επανάληψη κώδικα.

Ακολουθεί παράδειγμα Κλήσης Δημιουργού από Άλλον Δημιουργό

```
public class Box {
    private double width;
    private double height;
    private double depth;

    // Δημιουργός χωρίς παραμέτρους
    public Box() {
        this(0, 0, 0); // Κλήση του δημιουργού με τρεις παραμέτρους
    }

    // Δημιουργός με μία παράμετρο
    public Box(double side) {
        this(side, side, side); // Κλήση του δημιουργού με τρεις παραμέτρους
    }

    // Δημιουργός με τρεις παραμέτρους
    public Box(double width, double height, double depth) {
        this.width = width;
        this.height = height;
        this.depth = depth;
    }

    public double volume() {
        return width * height * depth;
    }
}
```

4.6. Ενθυλάκωση (Encapsulation)

4.6.1. Η έννοια της Ενθυλάκωσης (Encapsulation)

Η ενθυλάκωση (encapsulation) είναι μια από τις βασικές αρχές του αντικειμενοστραφούς προγραμματισμού (OOP). Αναφέρεται στη διαδικασία απόκρυψης των εσωτερικών λεπτομερειών μιας κλάσης και στην παροχή ενός καθαρού και ελεγχόμενου τρόπου πρόσβασης και τροποποίησης των δεδομένων της.

Οι βασικές Αρχές της Ενθυλάκωσης είναι:

- Απόκρυψη Δεδομένων (Data Hiding): Τα δεδομένα μιας κλάσης είναι ιδιωτικά (private) και δεν είναι προσβάσιμα απευθείας από εξωτερικές κλάσεις. Αυτό προστατεύει τα δεδομένα από ανεπιθύμητες τροποποιήσεις.
- Δημόσιες Μέθοδοι (Public Methods): Παρέχονται δημόσιες μέθοδοι (getters και setters) για την πρόσβαση και την τροποποίηση των ιδιωτικών δεδομένων. Αυτές οι μέθοδοι επιτρέπουν τον έλεγχο της πρόσβασης και της τροποποίησης των δεδομένων.

Το παρακάτω αποτελεί παράδειγμα ενθυλάκωσης.

```
public class Person {
    // Ιδιωτικά δεδομένα
    private String name;
    private int age;

    // Δημόσια μέθοδος για την απόκτηση του ονόματος
    public String getName() {
        return name;
    }

    // Δημόσια μέθοδος για την ρύθμιση του ονόματος
    public void setName(String name) {
        this.name = name;
    }

    // Δημόσια μέθοδος για την απόκτηση της ηλικίας
    public int getAge() {
        return age;
    }

    // Δημόσια μέθοδος για την ρύθμιση της ηλικίας
    public void setAge(int age) {
        if (age > 0) { // Έλεγχος εγκυρότητας
            this.age = age;
        }
    }
}
```

Τα πλεονεκτήματα της Ενθυλάκωσης είναι:

- Ασφάλεια Δεδομένων: Προστατεύει τα δεδομένα από ανεπιθύμητες τροποποιήσεις και διασφαλίζει ότι οι αλλαγές γίνονται με ελεγχόμενο τρόπο.
- Ευκολία Συντήρησης: Κάνει τον κώδικα πιο ευανάγνωστο και ευκολότερο στη συντήρηση, καθώς οι αλλαγές στα δεδομένα γίνονται μέσω καθορισμένων μεθόδων.
- Επαναχρησιμοποίηση Κώδικα: Οι κλάσεις με καλά ενθυλακωμένα δεδομένα μπορούν να επαναχρησιμοποιηθούν εύκολα σε διαφορετικά μέρη του προγράμματος χωρίς να ανησυχούμε για την εσωτερική τους κατάσταση.

Ένα παράδειγμα χρήσης της ενθυλάκωσης ακολουθεί

```
public class Main {
    public static void main(String[] args) {
        Person person = new Person();
        person.setName("John");
        person.setAge(30);

        System.out.println("Name: " + person.getName()); // Εκτυπώνει:
Name: John
        System.out.println("Age: " + person.getAge()); // Εκτυπώνει: Age:
30
    }
}
```

Στο παραπάνω παράδειγμα, η κλάση Person χρησιμοποιεί την ενθυλάκωση για να προστατεύσει τα δεδομένα της και να παρέχει ελεγχόμενη πρόσβαση σε αυτά μέσω των μεθόδων getName, setName, getAge, και setAge.

4.6.2. Ορατότητα Μεθόδων (Modifiers)

Η Java υποστηρίζει τέσσερα διαφορετικά επίπεδα ορατότητας, αντίστοιχα με τους modifiers των μεθόδων:

- το **public** : η κλάση είναι ορατή από όλες τις κλάσεις του Java προγράμματος
- το **protected**: η κλάση είναι ορατή από τις άλλες κλάσεις που βρίσκονται στο ίδιο package και από τις υποκλάσεις της
- το **default** : η κλάση είναι ορατή από τις κλάσεις του package στο οποίο ανήκει
- και το **private**: η κλάση δεν είναι ορατή σε καμμία άλλη κλάση του Java προγράμματος.

Στον παρακάτω πίνακα φαίνονται αναλυτικά η ορατότητα κάθε modifier:

Ορατότητα	**public**	**protected**	**default**	**private**
Από την ίδια κλάση	Ναι	Ναι	Ναι	Ναι
Από άλλη κλάση στο ίδιο πακέτο	Ναι	Ναι	Ναι	Όχι
Από οποιαδήποτε μη υποκλάση σε άλλο πακέτο	Ναι	Όχι	Όχι	Όχι
Από υποκλάση στο ίδιο πακέτο	Ναι	Ναι	Ναι	Όχι
Από υποκλάση σε άλλο πακέτο	Ναι	Ναι	Όχι	Όχι

4.6.3. Παράδειγμα Ενθυλάκωσης

Η ενθυλάκωση (encapsulation) είναι μια από τις βασικές αρχές του αντικειμενοστραφούς προγραμματισμού (OOP). Στην Java, η ενθυλάκωση επιτυγχάνεται με τη χρήση των προσδιοριστών πρόσβασης (`access modifiers`) όπως `private`, `public`, και `protected`. Αυτό επιτρέπει την απόκρυψη των δεδομένων μιας κλάσης από άλλες κλάσεις και την παροχή ελεγχόμενης πρόσβασης μέσω μεθόδων.

Ας δούμε ένα παράδειγμα κλάσης `Person` που χρησιμοποιεί την ενθυλάκωση:

```
public class Person {
    // Ιδιωτικές μεταβλητές (κρυφά δεδομένα)
    private String name;
    private int age;

    // Δημόσιος κατασκευαστής
    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    // Δημόσιες μέθοδοι για πρόσβαση και τροποποίηση των ιδιωτικών
μεταβλητών
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        if (age > 0) { // Έλεγχος εγκυρότητας
            this.age = age;
        }
    }
}
```

Στο παράδειγμα αυτό, οι μεταβλητές `name` και `age` είναι δηλωμένες ως `private`, που σημαίνει ότι δεν μπορούν να προσπελαστούν απευθείας από άλλες κλάσεις.

Οι μέθοδοι `getName`, `setName`, `getAge`, και `setAge` είναι δηλωμένες ως `public`, επιτρέποντας την ελεγχόμενη πρόσβαση και τροποποίηση των ιδιωτικών μεταβλητών.

Η μέθοδος setAge περιλαμβάνει έναν έλεγχο εγκυρότητας για να διασφαλίσει ότι η ηλικία είναι θετική πριν την αποθήκευση.

```
Public class Main {
    public static void main(String[] args) {
```

```
        // Δημιουργία αντικειμένου Person
        Person person = new Person("John", 25);

        // Πρόσβαση και τροποποίηση των δεδομένων μέσω των δημόσιων μεθόδων
        System.out.println("Name: " + person.getName());
        System.out.println("Age: " + person.getAge());

        // Τροποποίηση των δεδομένων
        person.setName("Jane");
        person.setAge(30);

        // Εμφάνιση των ενημερωμένων δεδομένων
        System.out.println("Updated Name: " + person.getName());
        System.out.println("Updated Age: " + person.getAge());
    }
}
```

Με αυτόν τον τρόπο, η ενθυλάκωση βοηθά στην προστασία των δεδομένων και στην παροχή ελεγχόμενης πρόσβασης, καθιστώντας τον κώδικα πιο ασφαλή και ευανάγνωστο.

4.7. Static Μέθοδοι (Methods) και Μεταβλητές (Variables)

4.7.1. Παράδειγμα Static Μεθόδων

Οι static μέθοδοι στη Java ανήκουν στην κλάση και όχι σε κάποιο αντικείμενο της κλάσης. Αυτό σημαίνει ότι μπορούν να κληθούν χωρίς να δημιουργηθεί ένα αντικείμενο της κλάσης. Ας δούμε μερικά παραδείγματα static μεθόδων.

Παράδειγμα 1: Απλή Static Μέθοδος

```
public class MathUtils {
    // Static μέθοδος για την εύρεση του μέγιστου αριθμού
    public static int max(int a, int b) {
        return (a > b) ? a : b;
    }
}
public class Main {
    public static void main(String[] args) {
        // Κλήση της static μεθόδου χωρίς δημιουργία αντικειμένου
        int result = MathUtils.max(10, 20);
        System.out.println("Max: " + result); // Εκτυπώνει: Max: 20
    }
}
```

Παράδειγμα 2: Static Μέθοδος για Υπολογισμούς

```
public class Calculator {
    // Static μέθοδος για την πρόσθεση δύο αριθμών
    public static int add(int a, int b) {
        return a + b;
    }
    // Static μέθοδος για την αφαίρεση δύο αριθμών
    public static int subtract(int a, int b) {
        return a - b;
```

```
    }
}

public class Main {
    public static void main(String[] args) {
        // Κλήση των static μεθόδων χωρίς δημιουργία αντικειμένου
        int sum = Calculator.add(15, 5);
        int difference = Calculator.subtract(15, 5);

        System.out.println("Sum: " + sum); // Εκτυπώνει: Sum: 20
        System.out.println("Difference: " + difference); // Εκτυπώνει:
Difference: 10
    }
}
```

Παράδειγμα 3: Static Μέθοδος για Μετατροπές

```
public class Converter {
    // Static μέθοδος για τη μετατροπή βαθμών Κελσίου σε Φαρενάιτ
    public static double celsiusToFahrenheit(double celsius) {
        return (celsius * 9/5) + 32;
    }

    // Static μέθοδος για τη μετατροπή βαθμών Φαρενάιτ σε Κελσίου
    public static double fahrenheitToCelsius(double fahrenheit) {
        return (fahrenheit - 32) * 5/9;
    }
}

public class Main {
    public static void main(String[] args) {
        // Κλήση των static μεθόδων χωρίς δημιουργία αντικειμένου
        double fahrenheit = Converter.celsiusToFahrenheit(25);
        double celsius = Converter.fahrenheitToCelsius(77);

        System.out.println("25°C in Fahrenheit: " + fahrenheit); //
Εκτυπώνει: 25°C in Fahrenheit: 77.0
        System.out.println("77°F in Celsius: " + celsius); // Εκτυπώνει:
77°F in Celsius: 25.0
    }
}
```

Σημεία Προσοχής για Static Μεθόδους

- Πρόσβαση σε Static Μεταβλητές: Οι static μέθοδοι μπορούν να έχουν πρόσβαση μόνο σε static μεταβλητές και άλλες static μεθόδους της κλάσης.
- Περιορισμοί: Δεν μπορούν να έχουν πρόσβαση σε μη-static (instance) μεταβλητές ή μεθόδους απευθείας. Πρέπει να δημιουργηθεί ένα αντικείμενο της κλάσης για να γίνει αυτό.

Παράδειγμα Περιορισμού

```
public class Example {
    private int instanceVar = 10;
```

```
    private static int staticVar = 20;

    public static void staticMethod() {
        // System.out.println(instanceVar); // Σφάλμα: Δεν μπορεί να έχει πρόσβαση σε instance μεταβλητή
        System.out.println(staticVar); // Επιτρέπεται
    }

    public void instanceMethod() {
        System.out.println(instanceVar); // Επιτρέπεται
        System.out.println(staticVar); // Επιτρέπεται
    }
}
```

4.8. Απαριθμητοί Τύποι (Enumerated Types)

Οι Απαριθμητοί Τύποι (ή Enums) στην Java είναι ένας ειδικός τύπος δεδομένων που επιτρέπει στον προγραμματιστή να ορίσει μια μεταβλητή που μπορεί να έχει μόνο μία από μια προκαθορισμένη λίστα τιμών. Αυτό είναι ιδιαίτερα χρήσιμο για την αναπαράσταση σταθερών τιμών με ευανάγνωστο και διαχειρίσιμο τρόπο.

Οι απαριθμητοί τύποι δηλώνονται χρησιμοποιώντας τη λέξη-κλειδί `enum` όπως φαίνεται και στο παράδειγμα που ακολουθεί.

```
public enum Day {
    SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY
}
```

Μπορείτε να χρησιμοποιήσετε τους απαριθμητούς τύπους όπως οποιονδήποτε άλλο τύπο δεδομένων.

```
Day today = Day.MONDAY;
```

Οι απαριθμητοί τύποι μπορούν να περιέχουν μεθόδους και ιδιότητες όπως οι κανονικές κλάσεις.

```
public enum Day {
       SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY;

       public boolean isWeekend() {
           return this == SUNDAY || this == SATURDAY;
       }
}
```

Οι απαριθμητοί τύποι είναι επίσης ιδιαίτερα χρήσιμοι σε `switch` statements.

```
switch (today) {
       case MONDAY:
           System.out.println("Start of the work week");
           break;
       case FRIDAY:
           System.out.println("End of the work week");
           break;
       case SATURDAY:
       case SUNDAY:
           System.out.println("Weekend!");
```

```
            break;
}
```

Μπορείτε να προσθέσετε ιδιότητες και μεθόδους στους απαριθμητούς τύπους για να τους κάνετε πιο λειτουργικούς.

```
public enum Planet {
        MERCURY(3.303e+23, 2.4397e6),
        VENUS(4.869e+24, 6.0518e6),
        EARTH(5.976e+24, 6.37814e6),
        MARS(6.421e+23, 3.3972e6);

        private final double mass;   // in kilograms
        private final double radius; // in meters

        Planet(double mass, double radius) {
            this.mass = mass;
            this.radius = radius;
        }

        public double getMass() {
            return mass;
        }

        public double getRadius() {
            return radius;
        }
}
```

4.9. Ερωτήσεις Αυτο-αξιολόγησης

1. **Ποια από τις παρακάτω μεθόδους της κλάσης String επιστρέφει το μήκος μιας συμβολοσειράς;**

   α) length()

   β) size()

   γ) getLength()

   δ) count()

2. **Ποια μέθοδος της κλάσης String χρησιμοποιείται για να μετατρέψει όλα τα γράμματα σε πεζά;**

   α) toLowerCase()

   β) toLower()

   γ) lowerCase()

   δ) convertToLower()

3. **Ποια είναι η διαφορά μεταξύ String και StringBuilder;**

   α) Το String είναι immutable, ενώ το StringBuilder mutable

   β) Το StringBuilder είναι πιο αργό από το String

γ) Το String δεν μπορεί να περιέχει κενά

δ) Το StringBuilder δεν υποστηρίζει μεθόδους για χειρισμό κειμένου

4. Ποια από τις παρακάτω εντολές δημιουργεί έναν πίνακα μήκους 5 στη Java;

α) int[] array = new int[5];

β) int array[5];

γ) int array = new int(5);

δ) array int = new int[5];

5. Ποια είναι η σωστή σύνταξη για να αρχικοποιήσετε έναν πίνακα με τιμές;

α) int[] array = {1, 2, 3};

β) int array = new {1, 2, 3};

γ) int[] array = new int{1, 2, 3};

δ) int array[3] = {1, 2, 3};

6. Ποια από τις παρακάτω εντολές είναι σωστή για επανάληψη for-each σε έναν πίνακα;

α) for (int i : array) {}

β) for (array i : int) {}

γ) for each (int i in array) {}

δ) foreach (int i : array) {}

7. Ποια μέθοδος χρησιμοποιείται για να προσθέσετε ένα στοιχείο σε μια ArrayList;

α) insert()

β) add()

γ) push()

δ) append()

8. Ποια από τις παρακάτω κλάσεις χρησιμοποιείται για την εργασία με ημερομηνίες στη Java;

α) java.util.Date

β) java.time.LocalDate

γ) java.date.DateTime

δ) java.util.Time

9. Ποιο είναι το κύριο εργαλείο αποσφαλμάτωσης στο Eclipse;

α) Debugger

β) Inspector

γ) Console

δ) Breakpoints

10. Ποια μέθοδος χρησιμοποιείται για τη μορφοποίηση μιας ημερομηνίας;

α) format()

β) toString()

γ) parse()

δ) convert()

11. Ποια από τις παρακάτω είναι μια Wrapper κλάση για τον τύπο int;

α) Integer

β) Double

γ) Float

δ) String

12. Ποια μέθοδος χρησιμοποιείται για να αντικαταστήσετε έναν χαρακτήρα σε μια συμβολοσειρά;

α) replace()

β) replaceChar()

γ) updateChar()

δ) substitute()

13. Ποιο από τα παρακάτω ΔΕΝ είναι χαρακτηριστικό της κλάσης StringBuilder;

α) Είναι mutable

β) Μπορεί να χρησιμοποιηθεί για την τροποποίηση μιας συμβολοσειράς

γ) Υποστηρίζει την προσθήκη χαρακτήρων με τη μέθοδο append()

δ) Είναι immutable όπως η κλάση String

14. Ποια είναι η σωστή σύνταξη για την επεξεργασία ενός δισδιάστατου πίνακα με επαναλήψεις;

α)

```
for (int i = 0; i < array.length; i++) {
   for (int j = 0; j < array[i].length; j++) {
      // Επεξεργασία
   }
}
```

β)

```
for (int i = 0; i < array.length; i++) {
   for (int j = 0; j < array.length; j++) {
      // Επεξεργασία
   }
}
```

γ)

```
for each (int[] row : array) {
   for each (int value : row) {
      // Επεξεργασία
   }
}
```

δ) Όλα τα παραπάνω

Λύση: α

15. Ποια είναι η κύρια διαφορά μεταξύ πίνακα και ArrayList;

α) Ο πίνακας είναι dynamic, ενώ η ArrayList static

β) Η ArrayList μπορεί να αλλάζει μέγεθος δυναμικά

γ) Ο πίνακας υποστηρίζει μόνο αριθμητικά δεδομένα

δ) Η ArrayList είναι πιο αργή από έναν πίνακα

16. Ποια μέθοδος της κλάσης LocalDate επιστρέφει την τρέχουσα ημερομηνία;

α) LocalDate.getCurrentDate()

β) LocalDate.today()

γ) LocalDate.now()

δ) LocalDate.getNow()

17. Ποια κλάση του πακέτου java.time χρησιμοποιείται για την καταγραφή χρονικής διάρκειας;

α) Period

β) Duration

γ) TimeSpan

δ) TimeFrame

18. Ποια μέθοδος χρησιμοποιείται για τη διαμόρφωση μιας ημερομηνίας σε μορφή συμβολοσειράς;

α) toFormat()

β) format()

γ) toString()

δ) convert()

19. Ποια είναι η διαφορά μεταξύ parseInt() και valueOf() στην κλάση Integer;

α) Το parseInt() επιστρέφει ακέραιο τύπο, ενώ το valueOf() επιστρέφει αντικείμενο Integer

β) Το parseInt() είναι πιο αργό από το valueOf()

γ) Το valueOf() είναι μόνο για αριθμούς κινητής υποδιαστολής

δ) Δεν υπάρχει διαφορά

20. Τι σας επιτρέπει να δείτε η καρτέλα "Variables" στον Debugger του Eclipse;

α) Τις τιμές όλων των μεταβλητών στο τρέχον σημείο

β) Το πλήρες ιστορικό εκτέλεσης

γ) Όλες τις δηλώσεις μεθόδων

δ) Τις θέσεις των breakpoints

4.10. Απαντήσεις στις ερωτήσεις Αυτο-αξιολόγησης

1. α	2. α	3. α	4. α	5. α
6. α	7. β	8. β	9. α	10. α
11. α	12. α	13. δ	14. α	15. β
16. γ	17. β	18. β	19. α	20. α

ΚΕΦΑΛΑΙΟ 5: Χειρισμός και μορφοποίηση δεδομένων, Χρήση Αποσφαλμάτωσης

5.1. Εκπαιδευτικοί στόχοι του παρόντος κεφαλαίου

Οι εκπαιδευτικοί στόχοι του παρόντος κεφαλαίου, συνοψίζονται στα κάτωθι σημεία. Οι εκπαιδευόμενοι:

- θα μάθουν πως υλοποιούνται στη Java τα αλφαριθμητικά,
- θα γνωρίσουν τη κλάση String και τις βασικές της μεθόδους,
- θα μάθουν τον τρόπο ελέγχου ισότητας των String,
- θα γνωρίσουν τη κλάση StringBuilder,
- θα μάθουν πως γίνεται η διαχείριση Πολλαπλών Τιμών με την χρήση Πινάκων,
- θα μάθουν πως γίνεται η δήλωση, δημιουργία και αρχικοποίηση Πινάκων,
- θα μάθουν πως γίνεται η Επεξεργασία Πολυδιάστατων πινάκων,
- θα μάθουν πως υλοποιούνται οι εντολές Επανάληψης στους Πίνακες,
- θα μάθουν πως υλοποιούνται οι εντολέΕπανάληψης for-each στους Πίνακες,
- θα μάθουν πως υλοποιείται η κλάση ArrayList,
- θα μάθουν πως χρησιμοποιούνται οι ημερομηνίες στη Java,
- θα μάθουν να χειρίζονται ημερομηνίες,
- θα δουν τις κλάσεις του package java.time,
- μάθουν να χειρίζονται τις κλάσεις LocalDate, LocalDateTime, ZonedDateTime, Instant, Period , Duration,
- θα μάθουν πως γίνεται η Μορφοποίηση Ημερομηνιών,
- θα γνωρίσουν τις Wrapper Κλάσεις,
- θα γνωρίσουν τη χρήση Αποσφαλμάτωσης στο Eclipse (Eclipse Debugger).

5.2. Αλφαριθμητικά

5.2.1. Η κλάση String και οι βασικές της μέθοδοι (length, concatenation, indexOf, substring, trim, toUppercase, toLowercase, charAt κ.α.)

Ως αλφαριθμητικό στην Java ορίζεται μία ακολουθία χαρακτήρων που συμπεριφέρονται ως ενιαία οντότητα. Ο πιο απλός τρόπος και αυτός που χρησιμοποιείται περισσότερο από κάθε άλλον για την δήλωση και αρχικοποίηση ενός αλφαριθμητικού, είναι αυτός που παράγει ένα αλφαριθμητικό από μία κυριολεκτική τιμή με χρήση του τελεστή ίσον , όπως στο παράδειγμα που ακολουθεί.

```
String s1 = "Some string
```

Η κλάση String προσφέρει πληθώρα χρήσιμων μεθόδων για τον χειρισμό των αλφαριθμητικών των προγραμμάτων μας, όπως:

- `length()`: Επιστρέφει τον αριθμό χαρακτήρων του αλφαριθμητικού
- `charAt(int)`: Επιστρέφει τον χαρακτήρα που βρίσκεται στη θέση που περνάμε ως παράμετρο. Η αρίθμηση ξεκινάει πάντοτε από το 0
- `isEmpty()`: Επιστρέφει true αν το αλφαριθμητικό έχει μήκος 0 αλλιώς false
- `equals(String)`: Επιστρέφει true αν το αλφαριθμητικό που περνάμε ως παράμετρο είναι όμοιο με αυτό που καλεί τη μέθοδο, αλλιώς επιστρέφει false.
- `equalsIgnoreCase(String)`: Κάνει ό,τι ακριβώς και η προηγούμενη με τη διαφορά όμως πως δεν κάνει διαχωρισμό μεταξύ κεφαλαίων και πεζών χαρακτήρων, δηλαδή δύο αλφαριθμητικά που διαφέρουν μόνο ως προς τον τύπο χαρακτήρων θεωρούνται από τη συγκεκριμένη μέθοδο ως ίδια και άρα σε μία τέτοια περίπτωση θα επιστρέψει true.
- `compareTo(String)`: Συγκρίνει το αλφαριθμητικό που καλεί τη μέθοδο με αυτό που περνάμε ως παράμετρο και επιστρέφει <0 αν το πρώτο είναι μικρότερο του δευτέρου, 0 αν είναι ίσα σε μέγεθος και >0 αν είναι μεγαλύτερο.
- `toUppercase`(): Επιστρέφει το αλφαριθμητικό που καλεί τη μέθοδο έχοντας μετατρέψει τους αλφαβητικούς χαρακτήρες που περιέχει σε κεφαλαίους.
- `toLowercase`(): Επιστρέφει το αλφαριθμητικό που καλεί τη μέθοδο έχοντας μετατρέψει όλους τους αλφαβητικούς χαρακτήρες που περιέχει σε πεζούς. Και οι δύο αυτές μέθοδοι διατίθενται και σε μία δεύτερη έκδοση που λαμβάνει ως παράμετρο ένα locale.
- `concat(String)`: Προσαρτεί το αλφαριθμητικό που περνάμε ως παράμετρο, σε αυτό που καλεί την μέθοδο. Λειτουργεί ακριβώς όπως ο τελεστής συν (+).
- `indexOf(int)`: Επιστρέφει τη θέση που βρίσκεται ο χαρακτήρας που περνάμε ως παράμετρο στο αλφαριθμητικό (index) ή -1 αν δεν βρεθεί.
- `contains(String)`: Επιστρέφει true αν η φράση που περνάμε ως παράμετρο υπάρχει στο αλφαριθμητικό, αλλιώς επιστρέφει false.
- `replace(char, char)`: Επιστρέφει το αλφαριθμητικό που καλεί τη μέθοδο έχοντας αντικαταστήσει τον χαρακτήρα που περνάμε ως πρώτη παράμετρο με αυτόν που περνάμε ως δεύτερη παράμετρο.
- `trim()`: Αφαιρεί τα κενά που μπορεί να περιέχονται στην αρχή ή το τέλος του αλφαριθμητικού που καλεί τη μέθοδο και επιστρέφει το αλφαριθμητικό χωρίς τα κενά.
- `substring(int, int)`: Επιτρέφει το υπο-αλφαριθμητικό που περιέχει τους χαρακτήρες που υπάρχουν σε εκείνο που καλεί τη μέθοδο, από τη θέση που περνάμε ως πρώτη παράμετρο μέχρι μία θέση πριν από αυτή που περνάμε ως δεύτερη παράμετρο.

- `format(String)`: Επιστρέφει το αλφαριθμητικό που παράγεται ως αποτέλεσμα εφαρμογής του αλφαριθμητικού μορφοποίησης που περάσαμε ως παράμετρο. Διατίθεται και σε υπερφορτωμένη έκδοση που λαμβάνει ως παράμετρο και ένα Locale.
- `matches(String)`: Επιστρέφει true αν το αλφαριθμητικό αντιστοιχεί πλήρως την έκφραση μορφοποίησης (regular expression) του αλφαριθμητικού που περάσαμε ως παράμετρο, αλλιώς επιστρέφει false.
- `replaceFirst(String, String)`: Επιστρέφει το αλφαριθμητικό που προκύπτει από την αντικατάσταση της πρώτης ακολουθίας χαρακτήρων που θα βρεθεί να ταιριάζει με την έκφραση μορφοποίησης της πρώτης παραμέτρου, από το αλφαριθμητικό που περνάμε ως δεύτερη παράμετρο.
- `replaceAll(String, String)`: Επιστρέφει το αλφαριθμητικό που προκύπτει από την αντικατάσταση όλων των ακολουθιών χαρακτήρων που θα βρεθούν να ταιριάζουν με την έκφραση μορφοποίησης της πρώτης παραμέτρου, από το αλφαριθμητικό που περνάμε ως δεύτερη παράμετρο.
- `split(String)`: Κατακερματίζει το αλφαριθμητικό σε μικρότερα ανάλογα με την έκφραση που περνάμε ως παράμετρο και τα επιστρέφει με τη μορφή πίνακα String.

5.2.2. Έλεγχος ισότητας String

Στην Java, για να ελέγξουμε την ισότητα δύο αντικειμένων της κλάσης String, χρησιμοποιούμε τη μέθοδο equals(). Η μέθοδος αυτή συγκρίνει το περιεχόμενο των δύο αλφαριθμητικών και επιστρέφει true αν είναι ίσα και false αν δεν είναι.

Εδώ είναι ένα παράδειγμα:

```
String str1 = "Hello";
String str2 = "Hello";
String str3 = "World";

boolean isEqual1 = str1.equals(str2); // Επιστρέφει true
boolean isEqual2 = str1.equals(str3); // Επιστρέφει false
```

Καλό είναι να μην χρησιμοποιούμε τον τελεστή == για τη σύγκριση αλφαριθμητικών, καθώς αυτός συγκρίνει τις αναφορές των αντικειμένων και όχι το περιεχόμενό τους.

5.2.3. Η κλάση StringBuilder και τα πλεονεκτήματά της σε σχέση με την κλάση String (Δήλωση, δημιουργία αντικειμένου, Concatenation, append κ.α.)

Τα αλφαριθμητικά της κλάσης String είναι μη τροποποιήσιμα. Λόγω της ιδιότητάς τους αυτής, η χρήση της κλάσης String σε προγράμματα που υπάρχουν πολλαπλές και συνεχείς μετατροπές αλφαριθμητικών είναι κατανοητό πως κρίνεται αναποτελεσματική. Για τις περιπτώσεις αυτές, δηλαδή για προγράμματα στα οποία λαμβάνουν χώρα πολλαπλές μετατροπές αλφαριθμητικών η Java παρέχει τις κλάσεις `StringBuilder` και `StringBuffer`.

Μία ακόμη βασική διαφορά των κλάσεων αυτών από τη `String` είναι πως κατά την εφαρμογή μεθόδων σε κάποιο αντικείμενο για την τροποποίησή του δεν απαιτείται η χρήση του τελεστή ίσον.

Πλην των δύο αυτών διαφορών, η χρήση των κλάσεων αυτών είναι παρόμοια με της `String`.

Για να δημιουργήσουμε ένα αντικείμενο τύπου `StringBuilder` χρησιμοποιούμε συνήθως τον constructor που λαμβάνει ως παράμετρο μία κυριολεκτική τιμή:

```
StringBuilder sb = new StringBuilder("Hello there!");
```

Ακολουθεί η αναφορά των πιο χρήσιμων μεθόδων των κλάσεων `StringBuilder` και `StringBuffer`:

- `length()`: Επιστρέφει τον αριθμό χαρακτήρων του αλφαριθμητικού που περιέχεται στο αντικείμενο `StringBuilder`.
- `setLength(int)`: Αυξάνει/μειώνει τη χωρητικότητα του `StringBuilder` σύμφωνα με την παράμετρο. Σε περίπτωση μείωσης, οι χαρακτήρες που μένουν εκτός, χάνονται. Περνώντας ως παράμετρο το 0, αδειάζει τελείως ο `StringBuilder` μιας και θέτουμε το μέγεθός του ίσο με 0.
- `capacity()`: Επιστρέφει τη χωρητικότητα του `StringBuilder` σε χαρακτήρες.
- `charAt(int)`: Επιστρέφει τον χαρακτήρα που βρίσκεται στη θέση που περνάμε ως παράμετρο. Η αρίθμηση και εδώ ξεκινάει από το 0.
- `setCharAt(int, char)`: Αντικαθιστά τον χαρακτήρα που βρίσκεται στη θέση που περνάμε ως πρώτη παράμετρο με αυτόν που περνάμε ως δεύτερη παράμετρο.
- `toString()`: Επιστρέφει το αλφαριθμητικό που περιέχεται στο `StringBuilder` ως `String`.
- `append(String)`: Προσαρτεί το αλφαριθμητικό που περνάμε ως παράμετρο, σε αυτό του `StringBuilder`. Λειτουργεί ακριβώς όπως ο τελεστής συν (+).
- `insert(int, String)`: Εισάγει το αλφαριθμητικό που περνάμε ως παράμετρο σε αυτό του `StringBuilder` ξεκινώντας από τη θέση που περνάμε ως παράμετρο και μεταφέροντας δεξιά τυχόν υπάρχοντες χαρακτήρες.
- `delete(int, int)`: Διαγράφει τα περιεχόμενα του `StringBuilder` από τη θέση που περνάμε ως πρώτη παράμετρο μέχρι και μία θέση πριν από αυτήν που περνάμε ως δεύτερη παράμετρο.
- `deleteCharAt(int)`: Διαγράφει τον χαρακτήρα που βρίσκεται στην θέση που περνάμε ως παράμετρο.
- `reverse()`: Αντιστρέφει τους χαρακτήρες του αλφαριθμητικού που περιέχεται στο αντικείμενο `StringBuilder`.

5.2.4. Παράδειγμα: Χειρισμός Αλφαριθμητικών

Ένα παράδειγμα χειρισμού των αλφαριθμητικών παρουσιάζεται στον κώδικα που ακολουθεί.

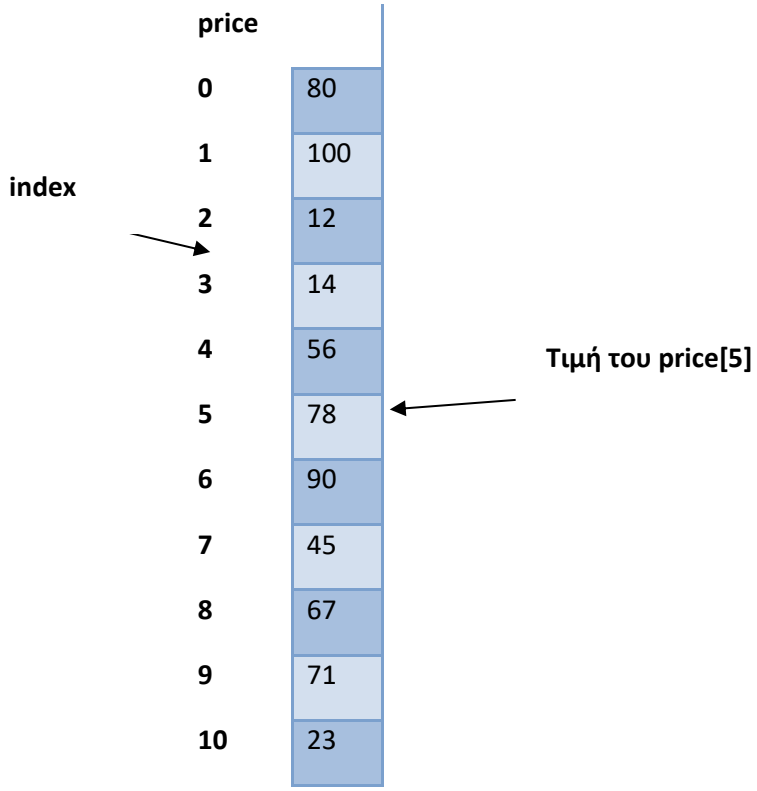
```
public class StringExample {
    public static void main(String[] args) {
        // Σύγκριση String
        String str1 = "Hello";
        String str2 = "Hello";
        String str3 = "World";

        // Χρήση της μεθόδου equals() για σύγκριση περιεχομένου
        System.out.println("str1 equals str2: " + str1.equals(str2)); //
true
        System.out.println("str1 equals str3: " + str1.equals(str3)); //
false

        // Χρήση της μεθόδου compareTo() για λεξικογραφική σύγκριση
        System.out.println("str1 compareTo str2: " + str1.compareTo(str2));
// 0
        System.out.println("str1 compareTo str3: " + str1.compareTo(str3));
// < 0

        // Χρήση StringBuilder
        StringBuilder sb = new StringBuilder();
        sb.append("Hello");
        sb.append(" ");
        sb.append("World");
        System.out.println("StringBuilder content: " + sb.toString()); //
Hello World

        // Διάφορες μέθοδοι του StringBuilder
        sb.insert(6, "Java ");
        System.out.println("After insert: " + sb.toString()); // Hello Java
World

        sb.replace(6, 10, "Beautiful");
        System.out.println("After replace: " + sb.toString()); // Hello
Beautiful World

        sb.delete(6, 15);
        System.out.println("After delete: " + sb.toString()); // Hello
World

        sb.reverse();
        System.out.println("After reverse: " + sb.toString()); // dlroW
olleH
    }
}
```

5.3. Διαχείριση Πολλαπλών Τιμών με την χρήση Πινάκων

5.3.1. Μονοδιάστατοι Πίνακες

Πολλές φορές στα προγράμματά μας χρειάζεται να αποθηκεύουμε έναν μεγάλο αριθμό από δεδομένα ιδίου τύπου, π.χ. 100 ακέραιες τιμές. Με βάση αυτά που γνωρίζουμε έως τώρα για να διατηρήσουμε ένα μεγάλο αριθμό δεδομένων θα πρέπει να δηλώσουμε τόσες μεταβλητές όσες τα δεδομένα που θέλουμε να διατηρήσουμε. Οι μεταβλητές αυτές θα πρέπει να ακολουθούν τον τύπο των δεδομένων. Όμως μια τέτοια λύση θα γέμιζε το πρόγραμμά μας με ονόματα μεταβλητών και θα ήταν δύσχρηστο στην περαιτέρω επεξεργασία των δεδομένων. Στις περιπτώσεις αυτές που θέλουμε να οργανώσουμε τα δεδομένα μας και να μπορούμε να έχουμε εύκολη πρόσβαση σε αυτά καθώς και να είναι δυνατή και η επεξεργασία τους χρησιμοποιούμε την Δομή Δεδομένων Πίνακες.

Ένας πίνακας (array) ομαδοποιεί τα δεδομένα σε μορφή λίστας. Ουσιαστικά ένας πίνακας είναι ένα σύνολο μεταβλητών ιδίου τύπου που στην πραγματικότητα είναι αποθηκευμένες σε διαδοχικά κελιά μνήμης. Η δομή δεδομένων Πίνακας διατηρεί το μέγεθος του πίνακα. Κάθε μεμονωμένη μεταβλητή του πίνακα ονομάζεται στοιχείο και ο προγραμματιστής έχει άμεση πρόσβαση σε κάθε μία από αυτές χρησιμοποιώντας συγκεκριμένη σύνταξη. Στο παρακάτω σχήμα φαίνεται ο πίνακας **price** που περιλαμβάνει 11 ακέραιες τιμές:



Η κάθε τιμή του πίνακα είναι αποθηκευμένη σε μία θέση του πίνακα. Ο αριθμός που δείχνει την θέση του πίνακα ονομάζεται **index**. Ο **index** ξεκινά πάντα από την θέση 0. Το μέγεθος τους παραπάνω πίνακα είναι 11 και το index ξεκινά από την θέση 0 έως την θέση 10. Γενικά ο index ενός πίνακα ξεκινά από την θέση 0 έως την θέση size-1, όπου size είναι το μέγεθος του πίνακα. Για να πάρουμε την τιμή ενός στοιχείου του πίνακα χρησιμοποιούμε το όνομα του πίνακα και τον index του στοιχείου. Συγκεκριμένα η σύνταξη είναι:

```
Η_Τιμή_ενός_στοιχείου  =  Όνομα_Πίνακα[index_στοιχείου] ;
```

Για παράδειγμα στον παραπάνω πίνακα η τιμή του στοιχείου στην θέση 5 είναι `price[5] = 78`

Οι πίνακες προσφέρουν μια εύκολη λύση όταν χρειαζόμαστε ένα μέσο για να αποθηκεύσουμε ένα μεγάλο πλήθος δεδομένων ιδίου τύπου. Παρόλα αυτά, βασικό τους μειονέκτημα είναι το γεγονός πως θα πρέπει εκ των προτέρων να γνωρίζουμε τον ακριβή αριθμό των δεδομένων ώστε να δημιουργήσουμε τον πίνακα με το κατάλληλο μέγεθος. Για τον λόγο αυτόν, οι πίνακες ονομάζονται στατικοί τύποι δεδομένων.

Σημειώνεται πως εάν κατά την προσπέλαση κάποιου μεμονωμένου στοιχείου βγούμε εκτός των ορίων του πίνακα, θα προκληθεί ένα `ArrayIndexOutOfBoundsException` και το πρόγραμμα θα τερματίσει βίαια. Για τον λόγο αυτόν χρειάζεται προσοχή, ειδικά όταν χειριζόμαστε μη συμμετρικούς πίνακες όπως αυτός του προηγούμενου παραδείγματος.

5.3.2. Πολυδιάστατοι Πίνακες

Στην προηγούμενη παράγραφο μελετήσαμε τους μονοδιάστατους πίνακες, η Java υποστηρίζει και την δημιουργίας πινάκων με περισσότερες της μίας διάστασης. Έτσι, μπορούμε να έχουμε πίνακες δύο (2) διαστάσεων, τριών (3) διαστάσεων ή και μεγαλύτερους. Στην πράξη εκτός από τους μονοδιάστατους που είναι και οι πιο κοινοί, επίσης σύνηθες είναι στα προγράμματα που γράφουμε να χρησιμοποιήσουμε δισδιάστατους πίνακες.

Για παράδειγμα αν θέλουμε να διατηρούμε για τα τελευταία δέκα (10) χρόνια τα στοιχεία των πωλήσεων ανά εξάμηνο χρειαζόμαστε έναν δισδιάστατο πίνακα όπου η κάθετη διάσταση θα αναπαριστά τον χρόνο και η οριζόντια διάσταση θα αναπαριστά το εξάμηνο. Τα στοιχεία του πίνακά που αναπαριστούν τις πωλήσεις μας σε ευρώ θα μπορούσε να είναι τύπου `float`. Δηλαδή ο πίνακας που χρειαζόμαστε είναι

```
float[][] annualSales = new float[10][2];
```

Όπου η πρώτη διάσταση είναι 10 όσα τα έτη που θέλουμε να διατηρήσουμε και η δεύτερη διάσταση είναι δύο (2) όσα δηλαδή και τα εξάμηνα του έτους. Η τιμή πώλησης του πρώτου εξαμήνου του δεύτερου έτους είναι `annualSales[1][0]`, όπου 1 είναι το `index` του έτους και 0 είναι το `index` του εξαμήνου.

5.3.3. Επεξεργασία Πολυδιάστατων πινάκων

Όπως και με τους μονοδιάστατους πίνακες, έτσι και με τους πολυδιάστατους πίνακες υπάρχει η δυνατότητα δημιουργίας και ταυτόχρονης απόδοσης τιμών, χρησιμοποιώντας τη σύνταξη με τα άγκιστρα ({}) σε μία γραμμή, όπως φαίνεται στο παράδειγμα που ακολουθεί:

```
int[][] a = {{5, 2, 4, 7}, {9, 2}, {3, 4}};
```

Στο παραπάνω παράδειγμα το στοιχείο a[0] περιέχει την αναφορά στον πίνακα {5, 2, 4, 7}, το a[1] στον πίνακα {9, 2} και το a[2] στον πίνακα {3, 4}. Τα μεμονωμένα στοιχεία ενός πολυδιάστατου πίνακα προσπελαύνονται με αντίστοιχο τρόπο με αυτόν των μονοδιάστατων, με τη διαφορά πως

χρησιμοποιούμε τον τελεστή πινάκων τόσες φορές όσες και οι διαστάσεις του πίνακα. Η αρίθμηση και στην περίπτωση των πολυδιάστατων πινάκων αρχίζει επίσης από το 0 και αυτό αφορά όλες τις διαστάσεις.

Έτσι, αν χρησιμοποιώντας τον πίνακα του παραδείγματος θέλαμε να εμφανίσουμε στην κονσόλα την τιμή 7, θα γράφαμε:

```
System.out.println(a[0][3]);
```

Όπως και στην περίπτωση των μονοδιάστατων πινάκων, έτσι και στους πολυδιάστατους αν κατά την προσπέλαση κάποιου μεμονωμένου στοιχείου βγούμε εκτός των ορίων του πίνακα, θα προκληθεί ένα `ArrayIndexOutOfBoundsException` και το πρόγραμμα θα τερματίσει βίαια. Για τον λόγο αυτόν χρειάζεται προσοχή, ειδικά όταν χειριζόμαστε μη συμμετρικούς πίνακες όπως αυτός του προηγούμενου παραδείγματος.

5.3.4. Δήλωση, δημιουργία και αρχικοποίηση Πινάκων

Στην Java οι πίνακες είναι αντικείμενα, δεν ανήκουν στους `primitive` τύπους δεδομένων. Για να δηλώσουμε και να αρχικοποιήσουμε έναν πίνακα υπάρχουν δύο (2) τρόποι.

Ο πρώτος τρόπος περιλαμβάνει την δήλωση του πίνακα και την αρχικοποίηση του:

```
Τύπος_δεδομένων_των_στοιχείων[] Όνομα_πίνακα = {λίστα_τιμών_χωριζόμενες_με_κόμμα};
```

Ο `Τύπος_δεδομένων_των_στοιχείων` αντιστοιχεί στον τύπο δεδομένων των τιμών που θα αποθηκεύσουμε στον πίνακα. Επίσης, με τις αγκύλες [] πληροφορούμε τον compiler ότι δηλώνουμε πίνακα. Το `Όνομα_πίνακα` αποτελεί όνομα μεταβλητής και ακολουθεί όλες τις συμβάσεις που ακολουθούμε στην ονοματοδοσία μεταβλητών. Η `λίστα_τιμών_χωριζόμενες_με_κόμμα` αποτελεί λίστα τιμών που ακολουθούν τον τύπο δεδομένων χωριζόμενες με το σύμβολο κόμμα.

Για παράδειγμα:

```
int[] price = {80,100,12,14,56,78,90,45,67,71,23}; //δήλωση και αρχικοποίηση πίνακα
String[] firstnames = {"ΜΑΡΙΑ","ΝΙΚΟΣ", "ΓΙΩΡΓΟΣ"};
```

Ο δεύτερος τρόπος περιλαμβάνει μόνο την δήλωση του πίνακα:

```
Τύπος_δεδομένων[] Όνομα_πίνακα = new Τύπος_δεδομένων[μέγεθος_πίνακα];
```

Ο τελεστής `new` χρησιμοποιείται πάντοτε όπως θα δούμε και στις επόμενες ενότητες για τη δέσμευση μνήμης και τη δυναμική δημιουργία αντικειμένων.

Για παράδειγμα:

```
int[] price  = new int[11]; //δήλωση πίνακα
//αρχικοποίηση πίνακα
price[0] = 80;
price[1] = 100;
price[2] = 12;
price[3] = 14;
price[4] = 56;
```

```
price[5] = 78;
price[6] = 90;
price[7] = 45;
price[8] = 67;
price[9] = 71;
price[10] = 23;

String[] firstnames = new String[3];
firstnames[0] = "ΜΑΡΙΑ";
firstnames[1] = "ΝΙΚΟΣ";
firstnames[2] =  "ΓΙΩΡΓΟΣ";
```

Τα στοιχεία του πίνακα είναι προσβάσιμα:

```
Η_Τιμή_ενός_στοιχείου  =  Όνομα_Πίνακα[index_στοιχείου] ;
```

Για παράδειγμα:

```
int boxPrice  = price[2];
int phonePrice = price[0];

System.out.println("Η τιμή του Κουτιού είναι: "+ boxPrice  +",  η τιμή του Βιβλίου είναι: "+price[9]);
```

Η επεξεργασία των στοιχείων του πίνακα γίνεται:

```
Όνομα_Πίνακα[index_στοιχείου] = Νέα_Τιμή_στοιχείου;
```

Για παράδειγμα:

```
String[] firstnames = {"ΜΑΡΙΑ","ΝΙΚΟΣ",  "ΓΙΩΡΓΟΣ"};
firstnames[0] = "ΓΕΩΡΓΙΑ";

System.out.println("Το όνομά μου είναι: "+ firstnames[0]);
```

Οι πίνακες διαθέτουν μία μεταβλητή με όνομα `length` που περιέχει το μέγεθος του πίνακα (αριθμό στοιχείων). Έτσι, στο παράδειγμά μας με τον πίνακα price, η έκφραση `price.length` θα επιστρέψει την τιμή 11. Ενώ η `firstnames.length` θα επιστρέψει 3.

Στην περίπτωση που δεν αρχικοποιήσουμε τον πίνακα οι τιμές που θα έχει εξαρτάται από τον τύπο δεδομένων του πίνακα, έτσι:

Τύπος πίνακα	**Αρχική Default τιμή**
byte, short, int, long	0
float, double	0.0
boolean	false
char	'\u0000'
Αντικείμενα	null

5.3.5. Πίνακες Σύνθετων Τύπων

Μέχρι τώρα μιλήσαμε για πίνακες που μπορούν να αποθηκεύσουν τιμές βασικών τύπων. Παρόλα αυτά, στη Java έχουμε τη δυνατότητα να δημιουργήσουμε εκτός των βασικών τύπων και πίνακες σύνθετων τύπων. Οι πίνακες της δεύτερης κατηγορίας αποθηκεύουν στα στοιχεία τους αναφορές

του σύνθετου τύπου. Έτσι λοιπόν, μπορούμε να δημιουργήσουμε πίνακες που αποθηκεύουν αναφορές σε αντικείμενα οποιασδήποτε από τις υπάρχουσες κλάσεις του προγράμματος. Με τη δημιουργία μίας νέας κλάσης έχουμε αυτόματα τη δυνατότητα να δηλώσουμε πίνακες του συγκεκριμένου τύπου. Για παράδειγμα, οι εντολές:

```
myCar[] b = new Car[20]; // πίνακας τύπου Button
myMoto[] c = new Moto[100]; // πίνακας τύπου Person
```

κατασκευάζουν έναν πίνακα μεγέθους 20 στοιχείων που μπορεί να αποθηκεύσει αναφορές σε αντικείμενα τύπου `Car` και έναν πίνακα 100 στοιχείων που αποθηκεύει αναφορές σε αντικείμενα τύπου `Moto` αντίστοιχα.

5.3.6. Εντολές Επανάληψης στους Πίνακες

Άλλος είναι συνηθισμένος τρόπος για την αρχικοποίηση (αλλά και την χρήση) των τιμών ενός πίνακα, ο οποίος είναι χρήσιμος και χρησιμοποιείται συχνά, είναι κάνοντας χρήση μιας δομής επανάληψης, συνήθως της for, όπως φαίνεται στο ακόλουθο παράδειγμα:

```
for(int i = 0; i < a.length; i++);
a[i] = i + 1;
```

Η `for` στο συγκεκριμένο παράδειγμα θα γεμίσει τα κελιά του πίνακα a με τις τιμές 1, 2 και 3 αντίστοιχα.
Προϋπόθεση βέβαια για τη χρήση της συγκεκριμένης μεθόδου είναι οι τιμές που θα αποθηκευτούν να μπορούν να παραχθούν μέσω κάποιου τύπου συναρτήσει του μετρητή της for (π.χ. θέλουμε να αποθηκεύσουμε σε έναν πίνακα 10 στοιχείων τους ακέραιους από το 1 έως το 10). Επίσης μπορεί να χρησιμοποιηθεί όταν οι τιμές που θα αποθηκευτούν πρόκειται να εισαχθούν μέσω του πληκτρολογίου από τον χρήστη.

Στο πρόγραμμα που ακολουθεί γίνεται χρήση ενός δισδιάστατου πίνακα με διαστάσεις 30x40. Ο πίνακας αρχικοποιείται με τη βοήθεια δύο for και στη συνέχεια οι τιμές του προβάλλονται στην κονσόλα.

```
package myArrays;
public class moreArrays {
	public static void main(String[] args) {
		// multi dimensional array
		int[][] multiDimArray = new int[30][40];
		// initialization
		for(int i = 0; i < multiDimArray.length; i++)
			for(int j = 0; j < 40; j++)
				array1[i][j] = i + j;
		// values
		for(int i = 0; i < multiDimArray.length; i++)
			for(int j = 0; j < 40; j++){
				System.out.print("multiDimArray ["+i+"]["+j+"]: "
					+ multiDimArray [i][j] + " ");
				System.out.print("\n");
			}
```

```
    }
}
```

5.3.7. Εντολή Επανάληψης for-each στους Πίνακες

Επιπρόσθετα της χρήσης της εντολής for για να διατρέξει κανείς τα στοιχεία ενός πίνακα, μπορεί να κάνει χρήση της εντολής `for-each`, η οποία συντάσσεται με τον ακόλουθό τρόπο.

```
for (Τύπος μεταβλητή : συλλογή)
     εντολή
```

έτσι για παράδειγμα, για να διατρέξει κανείς ένα μονοδιάστατο πίνακα και να τυπώσει τις τιμές του, μπορεί να χρησιμοποιήσει το ακόλουθο κώδικα σε java.

```
class PrintArray {
    public static void main(String args[]) {
        for (var s : args)
            System.out.print(s + " ");
        System.out.println();
    }
}
```

Το παραπάνω παράδειγμα διατρέχει τον πίνακα των ορισμάτων της `main` κλάσης και τυπώνει τα στοιχεία αυτά στην έξοδο. Στο παρακάτω δε παράδειγμα, γίνεται η ανάθεση των τιμών στον πίνακα με την χρήση της `for` επανάληψης και εν συνεχεία γίνεται χρήση της `for-each` για την εκτύπωση των τιμών του πίνακα στην έξοδο.

```
// Εισαγωγή τιμών με την κλασσική for
for (int i = 0; i < a.length; i++) {
     a[i] = i + 1;
}

     // Εκτύπωση τιμών πίνακα με χρήση foreach επανάληψης
for (int value : a) {
     System.out.println(value);
}
```

Αντίστοιχα με τους μονοδιάστατους πίνακες, η for-each μπορεί να χρησιμοποιηθεί και στους πολυδιαστατους πίνακες για εισαγωγή ή εκτύπωση των στοιχείων τους. Στο παράδειγμα που ακολουθεί, παρουσιάζεται το παράδειγμα που δείξαμε στους πολυδιάστατους πίνακες με την χρήση της for-each επανάληψης.

```
package myArrays;
public class moreArrays {
    public static void main(String[] args) {
        // multi dimensional array
        int[][] multiDimArray = new int[30][40];

        // initialization
        for (int i = 0; i < multiDimArray.length; i++)
            for (int j = 0; j < 40; j++)
                multiDimArray[i][j] = i + j;

        // values using foreach loop
```

```
        for (int[] row : multiDimArray) {
            for (int value : row) {
                System.out.print(value + " ");
            }
            System.out.print("\n");
        }
    }
}
```

5.3.8. Η κλάση ArrayList

Η κλάση `ArrayList` είναι μέρος του πακέτου `java.util` και παρέχει μια δυναμική δομή δεδομένων που μπορεί να αποθηκεύσει αντικείμενα. Σε αντίθεση με τους πίνακες (arrays), το μέγεθος μιας `ArrayList` μπορεί να αλλάξει δυναμικά κατά τη διάρκεια της εκτέλεσης του προγράμματος. Η `ArrayList` υλοποιεί τη διεπαφή `List` και παρέχει μεθόδους για την προσθήκη, διαγραφή και αναζήτηση στοιχείων.

Βασικές Μέθοδοι της `ArrayList` είναι οι παρακάτω:

- `add(E e)` : Προσθέτει το στοιχείο e στο τέλος της λίστας.
- `add(int index, E element)`: Προσθέτει το στοιχείο element στη θέση index.
- `get(int index)` : Επιστρέφει το στοιχείο στη θέση index.
- `remove(int index)` : Αφαιρεί το στοιχείο στη θέση index.
- `size()` : Επιστρέφει το μέγεθος της λίστας.
- `clear()` : Αφαιρεί όλα τα στοιχεία από τη λίστα.
- `isEmpty()` : Επιστρέφει true αν η λίστα είναι κενή, αλλιώς false.

5.3.9. Παραδείγματα Διαχείρισης Πινάκων με την ArrayList

Ακολουθεί ένα παράδειγμα κώδικα που δείχνει πώς να χρησιμοποιήσετε την ArrayList στην Java:

```
import java.util.ArrayList;

public class ArrayListExample {
    public static void main(String[] args) {
        // Δημιουργία μιας νέας ArrayList
        ArrayList<String> Numbers = new ArrayList<>();

        // Προσθήκη στοιχείων στην ArrayList
        Numbers.add("One");
        Numbers.add("Two");
        Numbers.add("Three");

        // Προσθήκη στοιχείου σε συγκεκριμένη θέση
        Numbers.add(1, "Zero");

        // Εκτύπωση όλων των στοιχείων
```

```
        System.out.println("Numbers: " + Numbers);

        // Λήψη στοιχείου από συγκεκριμένη θέση
        String Number = Numbers.get(2);
        System.out.println("Number at index 2: " + Number);

        // Αφαίρεση στοιχείου από συγκεκριμένη θέση
        Numbers.remove(3);
        System.out.println("Numbers after removal: " + Numbers);

        // Μέγεθος της ArrayList
        int size = Numbers.size();
        System.out.println("Size of the ArrayList: " + size);

        // Έλεγχος αν η ArrayList είναι κενή
        boolean isEmpty = Numbers.isEmpty();
        System.out.println("Is the ArrayList empty? " + isEmpty);

        // Καθαρισμός της ArrayList
        Numbers.clear();
        System.out.println("Numbers after clearing: " + Numbers);
    }
}
```

Σε ότι αφορά το συγκεκριμένο παράδειγμα, δημιουργούμε αρχικά μια νέα `ArrayList` για την αποθήκευση συμβολοσειρών (`Strings`). Η εντολή `ArrayList<String> Numbers = new ArrayList<>();` δημιουργεί μια κενή λίστα που μπορεί να αποθηκεύσει αντικείμενα τύπου `String`. Χρησιμοποιούμε τη μέθοδο add για να προσθέσουμε στοιχεία στην `ArrayList`. Το `Numbers.add("One");` προσθέτει το στοιχείο “One” στο τέλος της λίστας. Οι `Numbers.add("Two");` και `Numbers.add("Three");` προσθέτουν τα στοιχεία “`Two`” και “`Three`” αντίστοιχα. Τέλος η εντολή `Numbers.add(1, "Zero");` προσθέτει το στοιχείο “`Zero`” στη θέση 1, μετακινώντας τα υπόλοιπα στοιχεία προς τα δεξιά.

Για την εκτύπωση των στοιχείων, χρησιμοποιούμε την εντολή `System.out.println("Numbers: " + Numbers);` η οποία εκτυπώνει όλα τα στοιχεία της `ArrayList`. Η εντολή `String Number = Numbers.get(2);` λαμβάνει το στοιχείο στη θέση 2 και το αποθηκεύει στη μεταβλητή `Number`. Η `System.out.println("Number at index 2: " + Number);` εκτυπώνει το στοιχείο στη θέση 2. Τέλος η `Numbers.remove(3);` αφαιρεί το στοιχείο στη θέση 3 και η `System.out.println("Numbers after removal: " + Numbers);` εκτυπώνει τα στοιχεία της `ArrayList` μετά την αφαίρεση.

Η εντολή `int size = Numbers.size();` επιστρέφει το μέγεθος της ArrayList και η `System.out.println("Size of the ArrayList: " + size);` εκτυπώνει το μέγεθος της λίστας.

Εν συνεχεία γίνε έλεγχος με την εντολή `boolean isEmpty = Numbers.isEmpty();` αν η λίστα είναι κενή (επιστρέφει true) η όχι (επιστρέφει false). Η εντολή `System.out.println("Is the ArrayList empty? " + isEmpty);` εκτυπώνει αν η λίστα είναι κενή.

Τέλος γίνεται καθαρισμός της `ArrayList`. Η εντολή `Numbers.clear();` αφαιρεί όλα τα στοιχεία από την `ArrayList` και η `System.out.println("Numbers after clearing: " + Numbers);` εκτυπώνει την `ArrayList` μετά τον καθαρισμό.

5.4. Ημερομηνίες

5.4.1. Χειρισμός Ημερομηνιών - Το package java.time

Το πακέτο java.time είναι το κύριο API για την εργασία με ημερομηνίες, ώρες, στιγμές (instants) και διάρκειες (durations). Οι κλάσεις που ορίζονται σε αυτό το πακέτο αντιπροσωπεύουν τις βασικές έννοιες της ημερομηνίας και της ώρας, συμπεριλαμβανομένων των στιγμών, των διαρκειών, των ημερομηνιών, των ωρών, των ζωνών ώρας και των περιόδων. Βασίζονται στο σύστημα ημερολογίου ISO, το οποίο είναι το de facto παγκόσμιο ημερολόγιο που ακολουθεί τους κανόνες του Γρηγοριανού ημερολογίου.

5.4.1.1 Η κλάση LocalTime

Η κλάση `java.time.LocalTime` είναι η κλάση που χρησιμοποιούμε στη γλώσσα προγραμματισμού Java για να πάρουμε την τρέχουμε ώρα. Αποτελεί μέρος του πακέτου `java.time` και ένα παράδειγμα παραθέτουμε στον κώδικα που ακολουθεί.

```
// import τη LocalTime κλάση
import java.time.LocalTime;

public class Main {
  public static void main(String[] args) {
    LocalTime myObj = LocalTime.now();
    System.out.println(myObj);
  }
}
```

Το εξαγόμενο για το παράδειγμα θα είναι κάτι σαν το παρακάτω:

```
19:20:24.399035
```

5.4.1.2 Η κλάση LocalDate

Η κλάση `java.time.LocalDate` είναι η κλάση που χρησιμοποιούμε στη γλώσσα προγραμματισμού Java για να πάρουμε την τρέχουμε ημερομηνία. Αποτελεί μέρος του πακέτου `java.time` και ένα παράδειγμα παραθέτουμε στον κώδικα που ακολουθεί.

```
// import τη LocalDate κλάση
import java.time.LocalDate;

public class Main {
  public static void main(String[] args) {
    LocalDate myObj = LocalDate.now();
    System.out.println(myObj);
  }
}
```

Το εξαγόμενο για το παράδειγμα θα είναι κάτι σαν το παρακάτω:

```
2024-09-29
```

5.4.1.3 Η κλάση LocalDateTime

Η κλάση `java.time.LocalDateTime` είναι η κλάση που χρησιμοποιούμε στη γλώσσα προγραμματισμού Java για να πάρουμε την τρέχουμε ημερομηνία και ώρα. Αποτελεί μέρος του πακέτου `java.time` και ένα παράδειγμα παραθέτουμε στον κώδικα που ακολουθεί.

```
// import τη LocalDateTime κλάση
import java.time.LocalDateTime;

public class Main {
  public static void main(String[] args) {
    LocalDateTime myObj = LocalDateTime.now();
    System.out.println(myObj);
  }
}
```

Το εξαγόμενο για το παράδειγμα θα είναι κάτι σαν το παρακάτω:

```
2024-09-29T20:06:07.067624
```

5.4.2. Η κλάση ZonedDateTime

Η κλάση ZonedDateTime στην Java είναι μέρος του πακέτου java.time. Αυτή η κλάση αντιπροσωπεύει μια ημερομηνία και ώρα με ζώνη ώρας στο σύστημα ημερολογίου ISO-8601, όπως π.χ. `2007-12-03T10:15:30+01:00[Europe/Paris]`. Είναι αμετάβλητη (immutable) και χρησιμοποιείται ευρέως για λειτουργίες ημερομηνίας και ώρας στις νεότερες εκδόσεις της Java.

Τα βασικά χαρακτηριστικά της κλάσης είναι:

- **Αμεταβλητότητα**: Η κλάση είναι αμετάβλητη, που σημαίνει ότι κάθε τροποποίηση δημιουργεί ένα νέο αντικείμενο.
- **Ακρίβεια**: Αποθηκεύει όλα τα πεδία ημερομηνίας και ώρας με ακρίβεια νανοδευτερολέπτων.
- **Ζώνη Ώρας**: Περιλαμβάνει πληροφορίες ζώνης ώρας και ζώνης αντιστάθμισης (zone offset) για τη διαχείριση τοπικών ημερομηνιών και ωρών.

Ακολουθεί ένα παραδείγματα χρήσης της συγκεκριμένης κλάσης στη γλώσσα προγραμματισμού

```
import java.time.ZonedDateTime;
import java.time.ZoneId;

public class ZonedDateTimeExample {
    public static void main(String[] args) {
        // Δημιουργία ZonedDateTime με την τρέχουσα ημερομηνία και ώρα
        ZonedDateTime now = ZonedDateTime.now();
        System.out.println("Current Date and Time: " + now);

        // Δημιουργία ZonedDateTime με συγκεκριμένη ζώνη ώρας
        ZonedDateTime                    parisTime                    =
ZonedDateTime.now(ZoneId.of("Europe/Paris"));
        System.out.println("Paris Date and Time: " + parisTime);
```

```
        // Μετατροπή LocalDateTime σε ZonedDateTime
        LocalDateTime localDateTime = LocalDateTime.of(2024, 9, 30, 22,
36);
        ZonedDateTime zonedDateTime =
localDateTime.atZone(ZoneId.of("Europe/Athens"));
        System.out.println("Athens Date and Time: " + zonedDateTime);
    }
}
```

Οι κύριες μέθοδοι που είναι διαθέσιμες για την συγκεκριμένη κλάση είναι:

- `now()`: Επιστρέφει την τρέχουσα ημερομηνία και ώρα με ζώνη ώρας.
- `of()`: Δημιουργεί ένα ZonedDateTime με συγκεκριμένη ημερομηνία, ώρα και ζώνη ώρας.
- `withZoneSameInstant(ZoneId zone)`: Μετατρέπει το ZonedDateTime σε άλλη ζώνη ώρας διατηρώντας την ίδια στιγμή.
- `toLocalDateTime()`: Επιστρέφει το τοπικό LocalDateTime χωρίς πληροφορίες ζώνης ώρας.

Η κλάση `ZonedDateTime` είναι εξαιρετικά χρήσιμη για εφαρμογές που απαιτούν ακριβή διαχείριση ημερομηνίας και ώρας με ζώνες ώρας.

5.4.3. Η κλάση Period

Η κλάση `Period` στην Java αποτελεί μέρος του πακέτου `java.time` και χρησιμοποιείται για να αντιπροσωπεύσει μια χρονική περίοδο σε έτη, μήνες και ημέρες. Είναι ιδιαίτερα χρήσιμη για την εργασία με ημερομηνίες και για τον υπολογισμό της διαφοράς μεταξύ δύο ημερομηνιών.

Τα βασικά χαρακτηριστικά της συγκεκριμένης κλάσης είναι:

- `Αμεταβλητότητα`: Η κλάση είναι αμετάβλητη, που σημαίνει ότι κάθε τροποποίηση δημιουργεί ένα νέο αντικείμενο.
- `Μονάδες Μέτρησης`: Υποστηρίζει τις μονάδες έτη, μήνες και ημέρες.
- `ISO-8601`: Συμμορφώνεται με το σύστημα ημερολογίου ISO-8601.

Ακολουθεί παράδειγμα της χρήσης της κλάσης `Period`:

```
import java.time.LocalDate;
import java.time.Period;

public class PeriodExample {
    public static void main(String[] args) {
        // Δημιουργία Period από έτη, μήνες και ημέρες
        Period period = Period.of(2, 3, 5);
        System.out.println("Period: " + period); // P2Y3M5D

        // Υπολογισμός διαφοράς μεταξύ δύο ημερομηνιών
        LocalDate startDate = LocalDate.of(2020, 1, 1);
        LocalDate endDate = LocalDate.of(2023, 4, 30);
        Period difference = Period.between(startDate, endDate);
        System.out.println("Difference: " + difference); // P3Y3M29D
```

```
        // Προσθήκη Period σε ημερομηνία
        LocalDate newDate = startDate.plus(period);
        System.out.println("New Date: " + newDate); // 2022-04-06
    }
}
```

Οι κύριες μέθοδοι που χαρακτηρίζουν την κλάση είναι:

- `of(int years, int months, int days)`: Δημιουργεί ένα Period με συγκεκριμένα έτη, μήνες και ημέρες.
- `between(LocalDate startDate, LocalDate endDate)`: Υπολογίζει τη διαφορά μεταξύ δύο ημερομηνιών και επιστρέφει ένα `Period`.
- plusYears(long yearsToAdd), plusMonths(long monthsToAdd), plusDays(long daysToAdd): Προσθέτει έτη, μήνες ή ημέρες σε ένα Period.
- minusYears(long yearsToSubtract), minusMonths(long monthsToSubtract), minusDays(long daysToSubtract): Αφαιρεί έτη, μήνες ή ημέρες από ένα Period.

Η κλάση `Period` είναι ιδανική για την εργασία με χρονικές περιόδους και την υπολογιστική διαφορά μεταξύ ημερομηνιών.

5.4.4. Η κλάση Duration Κλάση

Η κλάση Duration χρησιμοποιείται για την αναπαράσταση χρονικών διαστημάτων. Είναι μέρος της βιβλιοθήκης java.time στην Java και παρέχει διάφορες μεθόδους για τη διαχείριση και τον υπολογισμό χρονικών διαστημάτων. Με την χρήση της κλάσης αυτής μπορούμε να κάνουμε τα ακόλουθα.

Μπορούμε να δημιουργήσουμε ένα αντικείμενο Duration χρησιμοποιώντας διάφορες στατικές μεθόδους όπως ofDays, ofHours, ofMinutes, κ.λπ για προσθήκη ή αφαίρεση χρονικών διαστημάτων.

```
Duration duration1 = Duration.ofHours(2);
Duration duration2 = Duration.ofMinutes(30);
Duration result = duration1.plus(duration2); // 2 ώρες και 30 λεπτά
```

Επίσης μπορούμε να συγκρίνουμε δύο αντικείμενα Duration ή να ελέγξουμε αν ένα χρονικό διάστημα είναι μηδενικό ή αρνητικό.

```
boolean isNegative = duration.isNegative();
int comparison = duration1.compareTo(duration2);
```

Μπορούμε να μετατρέψουμε το χρονικό διάστημα σε άλλες μονάδες όπως δευτερόλεπτα, λεπτά, ώρες, κ.λπ. όπως πχ

```
long seconds = duration.getSeconds();
```

Η κλάση Duration είναι πολύ χρήσιμη για την εργασία με χρονικά διαστήματα και παρέχει μια πλούσια σειρά εργαλείων για τη διαχείριση και τον υπολογισμό τους. Αν χρειάζεσαι περισσότερες πληροφορίες ή παραδείγματα, μπορείς να ανατρέξεις στην επίσημη τεκμηρίωση της Java.

5.4.5. Μορφοποίηση Ημερομηνιών

Η κλάση DateTimeFormatter είναι μέρος του πακέτου java.time.format και χρησιμοποιείται για τη μορφοποίηση και την ανάλυση (parsing) αντικειμένων ημερομηνίας και ώρας. Παρέχει διάφορους προεπιλεγμένους μορφοποιητές καθώς και τη δυνατότητα δημιουργίας προσαρμοσμένων μορφοποιητών για την κάλυψη συγκεκριμένων αναγκών. Οι βασικές λειτουργίες της κλάσης περιγράφονται παρακάτω.

Μορφοποίηση (Formatting):

```
Μετατρέπει αντικείμενα ημερομηνίας και ώρας σε συμβολοσειρές (strings) σύμφωνα με ένα συγκεκριμένο πρότυπο, όπως στο παρακάτω παράδειγμα.
LocalDate date = LocalDate.now();
DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd/MM/yyyy");
String formattedDate = date.format(formatter);
System.out.println("Formatted Date: " + formattedDate);
```

Ανάλυση (Parsing):

Μετατρέπει συμβολοσειρές (strings) σε αντικείμενα ημερομηνίας και ώρας σύμφωνα με ένα συγκεκριμένο πρότυπο, όπως στο παρακάτω παράδειγμα.

```
String dateString = "29/09/2024";
DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd/MM/yyyy");
LocalDate date = LocalDate.parse(dateString, formatter);
System.out.println("Parsed Date: " + date);
```

Προεπιλεγμένοι Μορφοποιητές:

Παρέχει διάφορους προεπιλεγμένους μορφοποιητές για κοινές μορφές ημερομηνίας και ώρας, όπως `ISO_LOCAL_DATE`, `ISO_LOCAL_TIME`, `ISO_LOCAL_DATE_TIME`, κ.λπ. όπως στο παρακάτω παράδειγμα.

```
LocalDateTime dateTime = LocalDateTime.now();
String isoDateTime =
dateTime.format(DateTimeFormatter.ISO_LOCAL_DATE_TIME);
System.out.println("ISO DateTime: " + isoDateTime);
```

Τοπικοποίηση (Localization):

Υποστηρίζει τη μορφοποίηση και ανάλυση ημερομηνιών και ωρών σύμφωνα με τις τοπικές ρυθμίσεις (locale), όπως στο παρακάτω παράδειγμα.

```
DateTimeFormatter formatter = DateTimeFormatter.ofPattern("d MMM uuuu",
Locale.FRENCH);
String formattedDate = date.format(formatter);
System.out.println("Formatted Date in French: " + formattedDate);
```

5.4.6. Παράδειγμα: Χειρισμός Ημερομηνιών

Ακολουθεί ένα παράδειγμα κώδικα που δείχνει πώς να χρησιμοποιήσετε την κλάση DateTimeFormatter για τη μορφοποίηση και την ανάλυση ημερομηνιών και ωρών:

```
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.util.Locale;

public class DateTimeFormatterExample {
    public static void main(String[] args) {
        // Μορφοποίηση ημερομηνίας
        LocalDate date = LocalDate.now();
        DateTimeFormatter                    formatter                    =
DateTimeFormatter.ofPattern("dd/MM/yyyy");
        String formattedDate = date.format(formatter);
        System.out.println("Formatted Date: " + formattedDate);

        // Ανάλυση ημερομηνίας
        String dateString = "29/09/2024";
        LocalDate parsedDate = LocalDate.parse(dateString, formatter);
        System.out.println("Parsed Date: " + parsedDate);

        // Προεπιλεγμένος μορφοποιητής
        LocalDateTime dateTime = LocalDateTime.now();
        String                       isoDateTime                       =
dateTime.format(DateTimeFormatter.ISO_LOCAL_DATE_TIME);
        System.out.println("ISO DateTime: " + isoDateTime);

        // Τοπικοποίηση
        DateTimeFormatter frenchFormatter = DateTimeFormatter.ofPattern("d
MMM uuuu", Locale.FRENCH);
        String formattedDateInFrench = date.format(frenchFormatter);
        System.out.println("Formatted    Date    in    French:    "    +
formattedDateInFrench);
    }
}
```

5.5. Wrapper Κλάσεις - Παράδειγμα Χειρισμού Wrapper Κλάσεων

Οι κλάσεις τύπου wrapper στην Java χρησιμοποιούνται για να ενθυλακώνουν τις βασικές (primitive) τιμές σε αντικείμενα. Αυτό επιτρέπει τη χρήση των βασικών τύπων δεδομένων σε περιβάλλοντα που απαιτούν αντικείμενα, όπως οι συλλογές (collections). Οι βασικές κλάσεις τύπου wrapper περιλαμβάνουν τις Integer, Double, Boolean, κ.λπ.

Μπορείτε να δημιουργήσετε αντικείμενα τύπου wrapper χρησιμοποιώντας τους κατασκευαστές τους ή τις στατικές μεθόδους valueOf.

```
Integer intObj = Integer.valueOf(10);
Double doubleObj = new Double(10.5);
```

Η Java υποστηρίζει την αυτόματη μετατροπή μεταξύ βασικών τύπων και των αντίστοιχων wrapper κλάσεων.

```
Integer intObj = 5; // Autoboxing
int intValue = intObj; // Unboxing
```

Οι κλάσεις τύπου wrapper επιτρέπουν τη χρήση βασικών τιμών σε συλλογές όπως `ArrayList`, HashMap, κ.λπ.

```
ArrayList<Integer> intList = new ArrayList<>();
intList.add(10);
intList.add(20);
```

Οι κλάσεις τύπου wrapper είναι απαραίτητες για την εργασία με βασικούς τύπους δεδομένων σε περιβάλλοντα που απαιτούν αντικείμενα και παρέχουν πολλές χρήσιμες λειτουργίες και μεθόδους.

5.6. Ερωτήσεις Αυτο-αξιολόγησης

1. **Τι χαρακτηρίζει κυρίως τον διαδικαστικό προγραμματισμό;**

   α) Επικεντρώνεται σε αντικείμενα

   β) Οργανώνει τον κώδικα σε διαδικασίες και συναρτήσεις

   γ) Υλοποιεί κλάσεις και αντικείμενα

   δ) Οργανώνει τον κώδικα με βάση τις ιδιότητες

2. **Στον δομημένο προγραμματισμό, τα δεδομένα:**

   α) Είναι οργανωμένα σε αντικείμενα

   β) Είναι απομονωμένα σε διαφορετικά πακέτα

   γ) Είναι διαθέσιμα σε όλο το πρόγραμμα

   δ) Διαχειρίζονται μέσω διαδικασιών και συναρτήσεων

3. **Ποιο από τα παρακάτω ΔΕΝ είναι χαρακτηριστικό του αντικειμενοστραφούς προγραμματισμού;**

   α) Ενθυλάκωση (Encapsulation)

   β) Κληρονομικότητα (Inheritance)

   γ) Επανάληψη (Iteration)

   δ) Πολυμορφισμός (Polymorphism)

4. **Μια κλάση στη Java είναι:**

   α) Ένα στιγμιότυπο δεδομένων

   β) Ένας ορισμός ή πρότυπο για τη δημιουργία αντικειμένων

   γ) Μια μέθοδος του αντικειμένου

   δ) Μια στατική μεταβλητή

5. **Πώς δημιουργείται ένα αντικείμενο από μια κλάση στη Java;**

   α) MyClass obj;

   β) new MyClass obj;

   γ) MyClass obj = new MyClass();

   δ) MyClass obj = create MyClass();

6. Οι ιδιότητες μιας κλάσης είναι:

α) Μέθοδοι που ανήκουν στην κλάση

β) Μεταβλητές που περιγράφουν την κατάσταση ενός αντικειμένου

γ) Στατικές μεταβλητές

δ) Επανάληψη δεδομένων

7. Οι συμπεριφορές μιας κλάσης αναφέρονται:

α) Στα δεδομένα που αποθηκεύει

β) Στις μεθόδους που εκτελούν ενέργειες

γ) Στα αντικείμενα που δημιουργεί

δ) Στις ιδιότητες που κληρονομεί

8. Τι είναι ένα package στη Java;

α) Ένα λογισμικό που εγκαθιστούμε

β) Ένας τρόπος ομαδοποίησης κλάσεων

γ) Μια βιβλιοθήκη δεδομένων

δ) Ένα framework για προγραμματισμό

9. Γιατί χρησιμοποιούμε packages στη Java;

α) Για βελτιστοποίηση της ταχύτητας εκτέλεσης

β) Για οργάνωση και αποφυγή συγκρούσεων ονομάτων

γ) Για την αποθήκευση δεδομένων

δ) Για την αυτόματη δημιουργία κώδικα

10. Ποια είναι η σωστή σύνταξη για να καλέσετε μια μέθοδο σε ένα αντικείμενο;

α) method(obj);

β) obj.method();

γ) call obj.method();

δ) method.call(obj);

11. Ποιο από τα παρακάτω είναι έγκυρο όνομα κλάσης στη Java;

α) 1Class

β) My Class

γ) _MyClass

δ) class

12. Ποιο είναι το κύριο πλεονέκτημα του αντικειμενοστραφούς προγραμματισμού;

α) Αύξηση ταχύτητας εκτέλεσης

β) Ευκολία κατανόησης και συντήρησης

γ) Μείωση κώδικα

δ) Υποστήριξη περισσότερων γλωσσών

13. Ποια είναι η κύρια διαφορά μεταξύ κλάσης και αντικειμένου;

α) Οι κλάσεις εκτελούν κώδικα, τα αντικείμενα όχι

β) Οι κλάσεις είναι πρότυπα, τα αντικείμενα είναι στιγμιότυπα

γ) Τα αντικείμενα είναι στατικά, οι κλάσεις είναι δυναμικές

δ) Οι κλάσεις ανήκουν στα αντικείμενα

14. Ποια είναι η προεπιλεγμένη ορατότητα (visibility) των μελών μιας κλάσης στη Java;

α) public

β) protected

γ) private

δ) package-private

15. Ποια είναι η σωστή μορφή δήλωσης κατασκευαστή σε μια κλάση;

α) MyClass() {}

β) void MyClass() {}

γ) static MyClass() {}

δ) constructor MyClass() {}

16. Τι είναι ο πολυμορφισμός στον αντικειμενοστραφή προγραμματισμό;

α) Η δυνατότητα ενός αντικειμένου να αλλάζει τύπο

β) Η δυνατότητα μιας μεθόδου να παίρνει πολλές μορφές

γ) Η δημιουργία πολλών κλάσεων από ένα αντικείμενο

δ) Η δημιουργία αντικειμένων από πολλές κλάσεις

17. Ποιο από τα παρακάτω είναι χαρακτηριστικό της κληρονομικότητας;

α) Αναγκάζει όλες τις κλάσεις να είναι ίδιες

β) Επιτρέπει σε μια κλάση να κληρονομήσει τις ιδιότητες μιας άλλης

γ) Εμποδίζει την πρόσβαση σε μέλη της υπερκλάσης

δ) Δημιουργεί αντικείμενα χωρίς κλάση

18. Ποια είναι η σωστή μορφή εισαγωγής ενός package στη Java;

α) import packageName.*;

β) use packageName.*;

γ) include packageName.*;

δ) library packageName.*;

19. Τι κάνει η λέξη-κλειδί super στη Java;

α) Δημιουργεί ένα νέο αντικείμενο

β) Αναφέρεται σε μέλη της υπερκλάσης

γ) Καλεί στατική μέθοδο

δ) Επιστρέφει ένα package

20. Τι ορίζεται ως ενθυλάκωση στον αντικειμενοστραφή προγραμματισμό;

α) Η χρήση κατασκευαστών

β) Η προστασία των δεδομένων μέσω ιδιωτικής πρόσβασης

γ) Η κληρονομικότητα ιδιοτήτων

δ) Η υλοποίηση στατικών μεθόδων

5.7. Απαντήσεις στις ερωτήσεις Αυτο-αξιολόγησης

1. β	2. δ	3. γ	4. β	5. γ
6. β	7. β	8. β	9. β	10. β
11. γ	12. β	13. β	14. δ	15. α
16. β	17. β	18. α	19. β	20. β

ΚΕΦΑΛΑΙΟ 6: Κληρονομικότητα Κλάσεων, Αφηρημένες Κλάσεις και lambda Εκφράσεις

6.1. Εκπαιδευτικοί στόχοι του παρόντος κεφαλαίου

Οι εκπαιδευτικοί στόχοι του παρόντος κεφαλαίου, συνοψίζονται στα κάτωθι σημεία. Οι εκπαιδευόμενοι:

- θα μάθουν για την κληρονομικότητα Κλάσεων και τις σχέσεις "IS-A", "HAS-A",
- θα μάθουν πως γίνεται η δημιουργία Υποκλάσεων,
- θα γνωρίσουν τα πλεονεκτήματα επαναχρησιμοποίησης Κώδικα,
- θα μάθουν πως γίνεται η χρήση του keyword «this»,
- θα μάθουν πως γίνεται η χρήση του keyword «super»,
- θα μάθουν πως γίνεται η Κλήση Δημιουργού (Constructor) Υπερκλάσης,
- θα γνωρίσουν την ορατότητα (Modifiers),
- θα γνωρίσουν την κλάση java.lang.Object,
- θα μάθουν πως γίνεται η δημιουργία αντικειμένου με αναφορά στην υπερκλάση,
- θα μάθουν πως γίνεται το Casting,
- θα μάθουν πως γίνεται η υπερκάλυψη μεθόδων (Method Overriding),
- θα γνωρίσουν τον πολυμορφισμό,
- θα μάθουν πως υλοποιούνται οι αφηρημένες κλάσεις (Abstract Classes),
- θα πραγματοποιήσουν μια εισαγωγή Lambda Εκφράσεις.

6.2. Κληρονομικότητα Κλάσεων

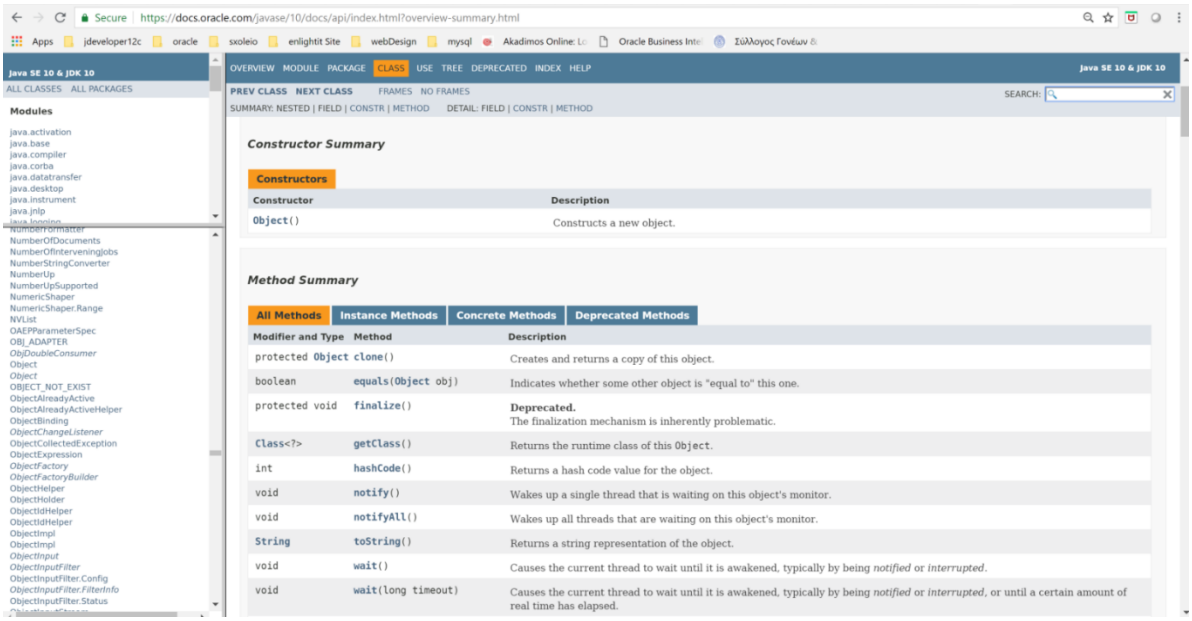
6.2.1. Η έννοια της κληρονομικότητας

Η κληρονομικότητα είναι μια θεμελιώδης τεχνική που χρησιμοποιείται για την οργάνωση και την δημιουργία κλάσεων. Αποτελεί μια από τις βασικότερες έννοιες του αντικειμενοστραφή προγραμματισμού και διευκολύνει στην επαναχρησιμοποίηση του λογισμικού. Μπορούμε να δημιουργήσουμε νέες κλάσεις από τις υπάρχουσες κλάσεις οι οποίες ονομάζονται **υποκλάσεις**. Οι υποκλάσεις κληρονομούν τα χαρακτηριστικά και τις μεθόδους της υπερκλάσης. Με αυτόν τον τρόπο επιτυγχάνεται η επαναχρησιμοποίηση του κώδικα που έχει ήδη γραφτεί για τις υπερκλάσεις. Οι υποκλάσεις αποτελούν εξειδίκευση των υπερκλάσεων και μπορούν εκτός από τα χαρακτιριστικά και μεθόδους των υπερκλάσεων να περιλαμβάνουν και δικά τους χαρακτηριστικά και μεθόδους που ουσιαστικά περιγράφουν αυτή τους την εξειδίκευση.

Ένα πρόγραμμα Java αποτελείται από ιεραρχίες κλάσεων (class hierarchy) όπως η ιεραρχία κλάσεων Όχημα που είδαμε στην Εικόνα 12 Ιεραρχία Κλάσεων . Μία ιεραρχία κλάσεων είναι ένα “δέντρο” που διαβάζεται από επάνω προς τα κάτω. Στην κορυφή βρίσκεται η βασική υπερκλάση

κλάση που περιλαμβάνει τα βασικά χαρακτηριστικά και βασική συμπεριφορά ενός αντικειμένου. Όσο προχωράμε προς τα κάτω οι κλάσεις εξειδικεύονται προσθέτοντας χαρακτηριστικά και συμπεριφορές. Κάθε κλάση στην ιεραρχία κληρονομεί τα χαρακτηριστικά και τις συμπεριφορές των κλάσεων που βρίσκονται επάνω από αυτήν στην ιεραρχία. Ενώ παράλληλα κληροδοτεί τα χαρακτηριστικά και συμπεριφορές της σε αυτές που βρίσκονται κάτω από αυτήν στην ιεραρχία. Για παράδειγμα η κλάση Όχημα αποτελεί την βασική υπερκλάση της ιεραρχίας Οχήματα. Η κλάση Αυτοκίνητο κληρονομεί τα χαρακτηριστικά και την συμπεριφορά της κλάσης Όχημα και παράλληλα κληροδοτεί τα δικά του χαρακτηριστικά και συμπεριφορές του στις κλάσεις Επιβατικό και Φορτηγό.

Όλες οι κλάσεις της Java που είτε δημιουργούμε εμείς είτε είναι προεγγεγραμμένες και περιλαμβάνονται στις libraries του jdk έχουν ως κορυφή βασική κλάση την υπερκλάση java.lang.Object. Κάθε φορά που δημιουργούμε μια κλάση αυτή είναι αυτόματα υποκλάση της java.lang.Object και κληρονομεί τα χαρακτηριστικά και μεθόδους της java.lang.Object.



Εικόνα 14 API της Κλάσης java.lang.Object (https://docs.oracle.com/javase/10/docs/api/index.html?overview-summary.html)

6.2.2. Σχέσεις "IS-A", "HAS-A"

Στην κληρονομικότητα, δύο από τις πιο σημαντικές έννοιες που αφορούν τις σχέσεις των κλάσεων είναι οι **σχέσεις IS-A** και **HAS-A**.

1. **Σχέση IS-A**: Η σχέση **IS-A** σημαίνει ότι μία κλάση είναι μια εξειδικευμένη έκδοση μιας άλλης κλάσης. Για παράδειγμα, εάν έχουμε μια υπερκλάση Vehicle και μια υποκλάση Car, μπορούμε να πούμε ότι **ένα αυτοκίνητο (Car) είναι ένα όχημα (Vehicle)**. Η σχέση IS-A επιτυγχάνεται μέσω της κληρονομικότητας.

2. **Σχέση HAS-A**: Η σχέση **HAS-A** δηλώνει ότι μία κλάση περιέχει ένα άλλο αντικείμενο ως χαρακτηριστικό της. Για παράδειγμα, εάν έχουμε μια κλάση Car και μια κλάση Engine, μπορούμε να πούμε ότι **ένα αυτοκίνητο έχει έναν κινητήρα** (Car HAS-A Engine). Η σχέση HAS-A επιτυγχάνεται μέσω **σύνθεσης** ή **συσχέτισης** (composition/association) και όχι μέσω κληρονομικότητας.

6.2.3. Δημιουργία Υποκλάσεων

Για να δημιουργήσουμε μια υποκλάση κατά τον ορισμό της χρησιμοποιούμε το keyword **extends** και κατόπιν σημειώνουμε το όνομα της κλάσης από την οποία θέλουμε να κληρονομήσουμε.

Για παράδειγμα η κλάση TShirt αποτελεί υποκλάση της Clothing. Για να ορίσουμε την κλάση Dress γράφουμε:

```
public class TShirt extends Clothing {
      //χαρακτηριστικά
      //μέθοδοι
}//end class
```

Στο παράδειγμά μας χρησιμοποιώντας την κληρονομικότητα η κλάση Clothing αποτελεί μια υπερκλάση, ενώ οι κλάσεις Shirt, Dress, TShirt, Trousers αποτελούν υποκλάσεις της Clothing. Η κλάση Clothing θα μπορούσε να είναι:

```
public class Clothing {
      public void setCode(String code) {…}
      public void setColor(String color) {…}
      public void setPrice() {…}
}
```

Η υποκλάση TShirt θα μπορούσε να είναι:

```
public class TShirt extends Clothing {
  public void setGender(String gender) {…}
}
```

Η υποκλάση TShirt κληρονομεί όλες τις μεθόδους της υπερκλάσης Clothing και επομένως μπορεί να χρησιμοποιήσει τις μεθόδους της υπερκλάσης σαν να ήταν δικές της. Για παράδειγμα σε μια main θα μπορούσαμε να γράψουμε:

```
public static void main(String[] args) {
      TShirt myTShirt = new TShirt();
      myTShirt.setSize("S");
            //όπου το 0 αντιστοιχεί σε ανδρικό TShirt και το 1 σε γυναικείο
TShirt
            myTShirt.setGender(0);
}
```

Το αντικείμενο (instance) της κλάσης TShirt μπορεί να χρησιμοποιήσει είτε τις μεθόδους της υπερκλάσης Clothing είτε τις μεθόδους της ιδίας κλάσης TShirt. Μια υποκλάση περιλαμβάνει τις μεθόδους και τα χαρακτηριστικά της υπερκλάσης και παράλληλα διαθέτει και χαρακτηριστικά αλλά και μεθόδους που την διακρίνουν και την χαρακτηρίζουν ως αντικείμενο. Για παράδειγμα η κλάση TShirt έχει χαρακτηριστικά το μέγεθος, το χρώμα, την τιμή που αποτελούν χαρακτηριστικά της υπερκλάσης Clothing αλλά περιλαμβάνει και το επιπλέον χαρακτηριστικό gender που χαρακτηρίζει αν ένα t-shirt είναι ανδρικό, γυναικείο, παιδικό-αγόρι, παιδικό-κορίτσι.

Στον παρακάτω πίνακα έχουν αναλυθεί οι κλάσεις Shirt, Dress, TShirt, Trousers και περιλαμβάνουν το σύνολο των μεθόδων που απαιτούνται για να περιγραφούν ως αντικείμενα. Όπως παρατηρείτε για κάθε χαρακτηριστικό της κλάσης έχει δημιουργηθεί μια get και μια set μέθοδος. Η λογική αυτή

αποτελεί την βασική έννοια του encapsulation όπου κάθε κλάση κρύβει τον κώδικα, που αλλάζει τις τιμές των χαρακτηριστικών της, σε μεθόδους. Το βασικότερο πλεονέκτημα αυτής της λογικής είναι ότι μέσα στις μεθόδους get και set πραγματοποιούνται όλοι οι απαιτούμενοι έλεγχοι ώστε να εμποδίζεται η αποθήκευση μη αποδεκτών τιμών στα χαρακτηριστικά της κλάσης.

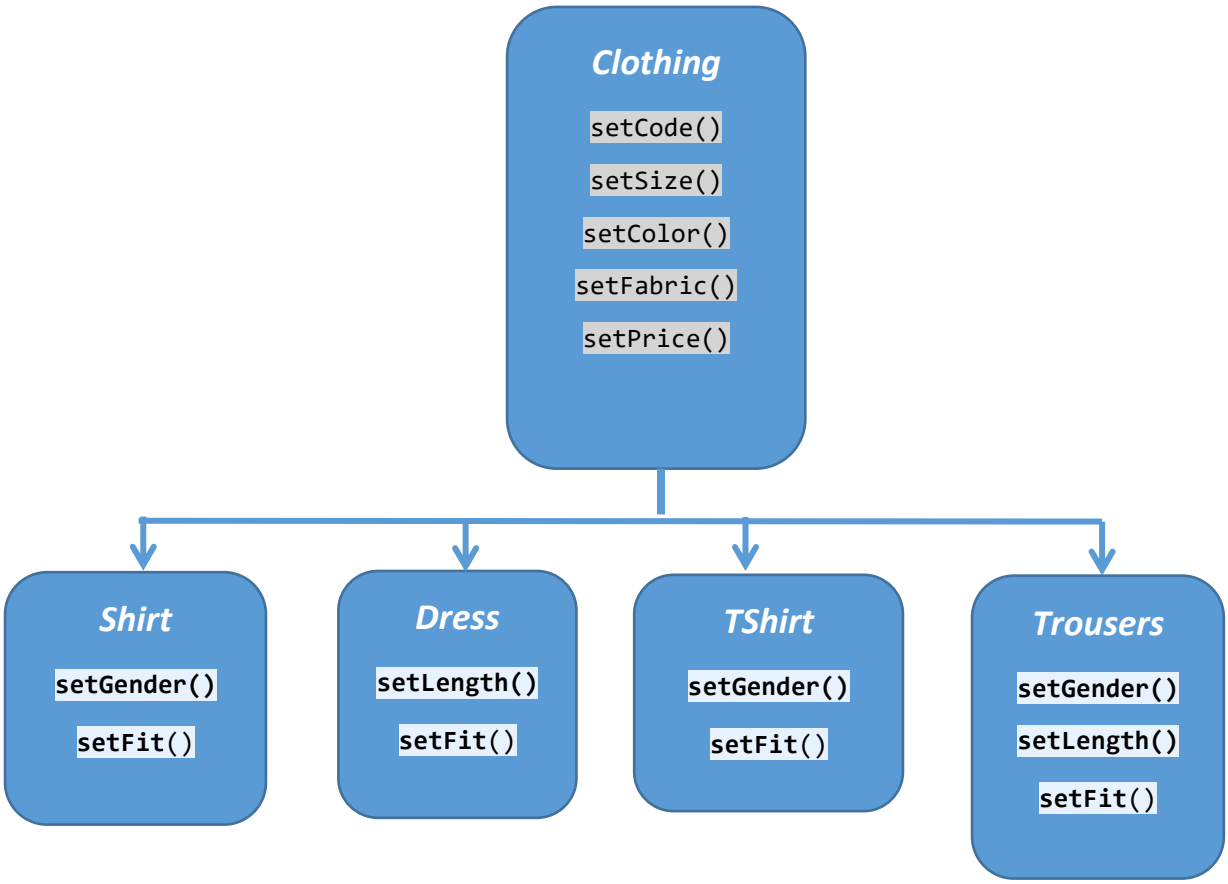
Shirt	*Dress*	*TShirt*	*Trousers*

`getCode()`
`getSize()`
`getColor()`
`getFabric()`
`getPrice()`
`getFit()`
`getGender()`
`displayInfo()`
	`getCode()`
	`getSize()`
	`getColor()`
	`getFabric()`
	`getPrice()`
	`getFit()`
	`getLength()`
	`displayInfo()`
		`getCode()`
		`getSize()`
		`getColor()`
		`getFabric()`
		`getPrice()`
		`getFit()`
		`getGender()`
		`displayInfo()`
			`getCode()`
			`getSize()`
			`getColor()`
			`getFabric()`
			`getPrice()`
			`getFit()`
			`getGender()`
			`getLength()`
			`displayInfo()`

`setCode()`
`setSize()`
`setColor()`
`setFabric()`
`setPrice()`
`setFit()`
`setGender()`
	`setCode()`
	`setSize()`
	`setColor()`
	`setFabric()`
	`setPrice()`
	`setFit()`
	`setLength()`
		`setCode()`
		`setSize()`
		`setColor()`
		`setFabric()`
		`setPrice()`
		`setFit()`
		`setGender()`
			`setCode()`
			`setSize()`
			`setColor()`
			`setFabric()`
			`setPrice()`
			`setFit()`
			`setGender()`
			`setLength()`

Παρατηρώντας καλύτερα τις κλάσεις βλέπουμε ότι περιλαμβάνουν ένα σύνολο από κοινές μεθόδους με την ίδια λειτουργικότητα όπως για παράδειγμα η getSize, setSize, getPrice, setPrice κ.α.. Οι κοινές μέθοδοι για όλες αυτές τις κλάσεις θα μπορούσαν να συγκεντρωθούν σε μια Κλάση και όλες οι υπόλοιπες κλάσεις Shirt, Dress, TShirt, Trousers που τις χρησιμοποιούν να γίνουν υποκλάσεις αυτής της Κλάσης. Αυτός ακριβώς είναι και ο ρόλος της Clothing, η οποία περιλαμβάνει το σύνολο των κοινών μεθόδων. Έτσι επιτυγχάνεται η επαναχρησιμοποίηση του κώδικα στις υποκλάσεις καθώς και η καλύτερη διαχείριση των μεθόδων. Μία πιθανή επιχειρησιακή αλλαγή στην διαχείριση ενός αντικειμένου με την χρήση της κληρονομικότητας θα απαιτούσε την τροποποίηση μιας μεθόδου δηλαδή αλλαγή σε ένα σημείο στο σημείο που αναλύεται ο κώδικας της μεθόδου και όχι σε πολλαπλά σημεία της εφαρμογής που δημιουργούμε. Για παράδειγμα αν το eshop επιθυμούσε να εισάγει ενδύματα για υπέρβαρους η τροποποίηση μόνο της μεθόδου getSize, setSize στην κλάση Clothing ώστε να περιλαμβάνει και άλλα μεγέθη θα μπορούσε να καλύψει αυτήν την επιχειρησιακή αλλαγή. Χωρίς την χρήση της κληρονομικότητας αυτή η επιχειρησιακή αλλαγή θα απαιτούσε αλλαγή σε όλες τις κλάσεις που αναπαριστούσαν ενδύματα.

Η ιεραρχία των κλάσεων φαίνεται στην παρακάτω εικόνα:



Εικόνα 15 Ιεραρχία Κλάσεων Υπερκλάση Clothing

Η κλάση Clothing είναι:

```
package eshop.items;

public class Clothing {
	private String code;
	private String size; //values 'OS','XS','S',M','L','XL', ....
	private String color; //values of color code
	private String fabric;//values of fabric code
	private double price;

	public Clothing(String code, String size, String color, String
fabric, double price) {
		super();
		this.code = code;
		this.size = size;
		this.color = color;
		this.fabric = fabric;
		this.price = price;
	}
	public String getCode() {
		return code;
	}
	public void setCode(String code) {
		this.code = code;
	}
	public String getSize() {
```

```
		return size;
	}
	public void setSize(String size) {
		this.size = size;
	}
	public String getColor() {
		return color;
	}
	public void setColor(String color) {
		this.color = color;
	}
	public String getFabric() {
		return fabric;
	}
	public void setFabric(String fabric) {
		this.fabric = fabric;
	}
	public double getPrice() {
		return price;
	}
	public void setPrice(double price) {
		this.price = price;
	}
	public void displayInfo() {
		System.out.println("Code = ["+this.code+"]");
		System.out.println("Size = ["+this.size+"]");
		System.out.println("Color = ["+this.color+"]");
		System.out.println("Fabric = ["+this.fabric+"]");
		System.out.println("Price = ["+this.price+"]");
	}
}
```

Η κλάση Shirt είναι:

```
package eshop.items;

public class Shirt extends Clothing {
	private String fit; //values "στενό", "φαρδύ", "ίσια γραμμή"
	private int gender; //values 0 -> Man, 1->Woman, 2->Boy, 3->Girl

	public Shirt(String code, String size, String color, String fabric,
double price,
			String fit, int gender) {
		super(code, size, color, fabric, price);

		this.fit = fit;
		this.gender = gender;

	}
	public String getFit() {
		return fit;
	}
	public void setFit(String fit) {
		this.fit = fit;
	}
	public int getGender() {
		return gender;
	}
	public void setGender(int gender) {
		this.gender = gender;
	}
```

```
	public void displayInfo() {
		super.displayInfo();
		System.out.println("Fit = ["+this.fit+"]");
		System.out.println("Gender = ["+this.gender+"]");
	}
}
```

Η κλάση Dress είναι:

```
package eshop.items;

public class Dress extends Clothing {
	private String fit; //values "στενό", "φαρδύ", "ίσια γραμμή"
	private String length; //values "κοντό", "μακρύ", "midi"


	public Dress(String code, String size, String color, String fabric,
double price,
			String fit, String length) {
		super(code, size, color, fabric, price);

		this.fit = fit;
		this.length = length;
	}
	public String getFit() {
		return fit;
	}
	public void setFit(String fit) {
		this.fit = fit;
	}
	public String getLength() {
		return length;
	}
	public void setLength(String length) {
		this.length = length;
	}
	public void displayInfo() {
		super.displayInfo();
		System.out.println("Fit = ["+this.fit+"]");
		System.out.println("Length = ["+this.length+"]");
	}
}
```

Στους Constructors Shirt και Dress υπάρχει ένα νέο keyword **super** αναπαριστά τον constructor της υπερκλάσης. Το keyword **this** αναπαριστά το instance της τρέχουσας κλάσης. Έχουμε πρόσβαση σε κάθε χαρακτηριστικό της κλάσης με το **this.ονομα_χαρακτηριστικού ή** απλά με μόνο το **ονομα_χαρακτηριστικού**. Όταν όμως μέσα σε μία μέθοδο υπάρχει παράμετρος η οποία έχει το ίδιο όνομα με το χαρακτηριστικό (όπως στους παραπάνω constructors) για να αναφερθούμε στο χαρακτηριστικό της κλάσης είναι απαραίτητη η χρήση του **this** όπως:

```
this.fit = fit;
```

`fit` είναι η παράμετρος της μεθόδου και `this.fit` είναι το χαρακτηριστικό fit της κλάσης.

Επίσης στην μέθοδο displayInfo() υπάρχει η γραμμή

```
super.displayInfo();
```

Με την οποία καλείται η μέθοδος displayInfo() της υπερκλάσης. Το super αναφέρεται στην υπερκλάση.

6.2.4. Πλεονέκτημα Επαναχρησιμοποίηση Κώδικα

Η κληρονομικότητα προσφέρει σημαντικά πλεονεκτήματα στην επαναχρησιμοποίηση του κώδικα:

- **Μείωση του Επαναλαμβανόμενου Κώδικα**: Ο κώδικας που είναι κοινός για πολλές κλάσεις μπορεί να τοποθετηθεί στην υπερκλάση και να κληρονομηθεί από τις υποκλάσεις.
- **Διευκόλυνση Συντήρησης**: Ο κώδικας σε μία υπερκλάση μπορεί να συντηρηθεί και να αναβαθμιστεί, επηρεάζοντας άμεσα όλες τις υποκλάσεις που τον κληρονομούν.
- **Διαχείριση Ιεραρχιών και Οργάνωση**: Η κληρονομικότητα βοηθά στη διαχείριση της πολυπλοκότητας οργανώνοντας τον κώδικα σε ιεραρχίες κλάσεων, όπου κάθε υποκλάση εξειδικεύεται περισσότερο από την προηγούμενη.

6.2.5. Η χρήση του keyword «this» και «super»

Η Java παρέχει τα keywords this και super για να αναφερόμαστε σε αντικείμενα και ιδιότητες εντός των κλάσεων, διευκολύνοντας τον χειρισμό και την πρόσβαση σε δεδομένα.

1. **Keyword this**: Χρησιμοποιείται για να αναφερθούμε στο τρέχον αντικείμενο της κλάσης. Είναι χρήσιμο όταν θέλουμε να διακρίνουμε ένα πεδίο από μία παράμετρο ή να κληθεί ο constructor της ίδιας κλάσης.
2. **Keyword super**: Χρησιμοποιείται για να αναφερθούμε στην υπερκλάση και να καλέσουμε τον constructor της υπερκλάσης ή άλλες μεθόδους που κληρονομήθηκαν.

Παράδειγμα Χρήσης του this

```
class Car {
    private String model;

    public Car(String model) {
        this.model = model; // Χρήση του `this` για διάκριση από την παράμετρο
    }

    public void displayModel() {
        System.out.println("Μοντέλο: " + this.model);
    }
}
```

Παράδειγμα Χρήσης του super

```
class Vehicle {
    int speed;

    public Vehicle(int speed) {
```

```
        this.speed = speed;
    }
}

class Car extends Vehicle {
    int numDoors;

    public Car(int speed, int numDoors) {
        super(speed); // Κλήση του constructor της υπερκλάσης
        this.numDoors = numDoors;
    }

    public void displayCarInfo() {
        System.out.println("Ταχύτητα: " + speed);
        System.out.println("Αριθμός πορτών: " + numDoors);
    }
}
```

Στο παράδειγμα:

- Το super(speed); καλεί τον constructor της υπερκλάσης Vehicle με το όρισμα speed, αρχικοποιώντας το χαρακτηριστικό speed της υπερκλάσης.
- Το this.numDoors = numDoors; χρησιμοποιεί το this για να διακρίνει το πεδίο numDoors από την παράμετρο του constructor.

Αυτές οι αρχικές έννοιες περιλαμβάνουν την κατανόηση της κληρονομικότητας, των σχέσεων IS-A και HAS-A, της δημιουργίας υποκλάσεων και της χρήσης των this και super.

6.2.6. Κλήση Δημιουργού (Constructor) Υπερκλάσης

Όπως έχει αναφερθεί σε προηγούμενο κεφάλαιο ο δημιουργός (constructor) είναι μια ειδική μέθοδος σε μια κλάση που εκτελείται αυτόματα όταν δημιουργείται ένα αντικείμενο αυτής της κλάσης. Κύριος σκοπός του είναι να αρχικοποιεί το αντικείμενο.

Βασικά Χαρακτηριστικά Δημιουργών:

- Έχουν το ίδιο όνομα με την κλάση.
- Δεν επιστρέφουν τιμή (ούτε καν void).
- Μπορούν να υπερφορτωθούν (overloading).

Παράδειγμα δημιουργού:

```
class Animal {
    String name;

    Animal(String name) {
        this.name = name;
    }
}
```

Στην ιεραρχία της κληρονομικότητας, μια υποκλάση (subclass) κληρονομεί τα μέλη και τις μεθόδους της υπερκλάσης (superclass). Ωστόσο, η υπερκλάση ενδέχεται να περιλαμβάνει έναν δημιουργό που πρέπει να κληθεί για να αρχικοποιηθούν οι ιδιότητες της υπερκλάσης.

Η Java παρέχει το keyword super για την κλήση:

1. Μεθόδων της υπερκλάσης.
2. Δημιουργών της υπερκλάσης.

Η κλήση του δημιουργού της υπερκλάσης γίνεται πάντα στην **πρώτη γραμμή** του δημιουργού της υποκλάσης.

Παράδειγμα:

```
class Animal {
    String name;

    // Δημιουργός υπερκλάσης
    Animal(String name) {
        this.name = name;
        System.out.println("Animal constructor called.");
    }
}

class Dog extends Animal {
    String breed;

    // Δημιουργός υποκλάσης
    Dog(String name, String breed) {
        super(name); // Κλήση του δημιουργού της υπερκλάσης
        this.breed = breed;
        System.out.println("Dog constructor called.");
    }
}
```

Κύρια Σημεία:

- Ο δημιουργός της υπερκλάσης Animal καλείται πρώτος μέσω του super(name).
- Ακολουθεί η αρχικοποίηση της υποκλάσης Dog.

Εκτέλεση:

```
public class Main {
    public static void main(String[] args) {
        Dog dog = new Dog("Buddy", "Labrador");
    }
}
```

Έξοδος:

Animal constructor called.

Dog constructor called.

Εάν η υπερκλάση δεν περιλαμβάνει κάποιον ρητό δημιουργό, ο μεταγλωττιστής προσθέτει αυτόματα έναν **προεπιλεγμένο δημιουργό** χωρίς παραμέτρους. Η υποκλάση καλεί τον προεπιλεγμένο δημιουργό μέσω του super().

```
class Animal {
    Animal() {
        System.out.println("Default Animal constructor.");
    }
}

class Dog extends Animal {
    Dog() {
        super(); // Κλήση προεπιλεγμένου δημιουργού
        System.out.println("Default Dog constructor.");
    }
}
```

Έξοδος:

Default Animal constructor.

Default Dog constructor.

Η υπερφόρτωση δημιουργών επιτρέπει τη χρήση πολλαπλών δημιουργών με διαφορετικές παραμέτρους. Η επιλογή του κατάλληλου δημιουργού της υπερκλάσης γίνεται μέσω του super() με συγκεκριμένες παραμέτρους.

```
class Animal {
    String name;
    int age;

    // Υπερφορτωμένοι δημιουργοί
    Animal(String name) {
        this.name = name;
    }

    Animal(String name, int age) {
        this.name = name;
        this.age = age;
    }
}

class Dog extends Animal {
    String breed;

    Dog(String name, int age, String breed) {
        super(name, age); // Κλήση υπερφορτωμένου δημιουργού
        this.breed = breed;
    }
}
```

Σφάλματα και Περιορισμοί κατά την Κλήση Δημιουργού Υπερκλάσης

1. **Το super() πρέπει να είναι η πρώτη εντολή:**

Κάθε δημιουργός υποκλάσης πρέπει να καλεί τον δημιουργό της υπερκλάσης πριν από οποιαδήποτε άλλη εντολή.

```
class Animal {
    Animal(String name) {}
}

class Dog extends Animal {
    Dog(String name, String breed) {
        // Σφάλμα: Το super(name) δεν είναι στην πρώτη γραμμή
        System.out.println("Dog is being created.");
        super(name);
    }
}
```

2. **Πρέπει να υπάρχει συμβατός δημιουργός στην υπερκλάση:**

Αν η υπερκλάση δεν έχει δημιουργό χωρίς παραμέτρους, η υποκλάση πρέπει να καλέσει έναν δημιουργό με παραμέτρους.

6.2.7. Ορατότητα και Διαμορφωτές Πρόσβασης (Modifiers)

Η **ορατότητα** αναφέρεται στο επίπεδο πρόσβασης που έχουν άλλα τμήματα του κώδικα στα χαρακτηριστικά και τις μεθόδους μιας κλάσης. Στη Java, χρησιμοποιούνται διαμορφωτές πρόσβασης (access modifiers) για τον έλεγχο της προσβασιμότητας των μελών μιας κλάσης από άλλες κλάσεις.

Οι πιο κοινοί διαμορφωτές πρόσβασης στη Java είναι:

1. **public**: Όταν ένα πεδίο ή μια μέθοδος δηλώνεται ως public, είναι προσβάσιμη από οποιαδήποτε άλλη κλάση, ανεξάρτητα από το πακέτο στο οποίο ανήκει.
2. **protected**: Ο διαμορφωτής protected επιτρέπει την πρόσβαση σε υποκλάσεις και κλάσεις του ίδιου πακέτου. Είναι χρήσιμος όταν θέλουμε να επιτρέψουμε την πρόσβαση σε χαρακτηριστικά από υποκλάσεις αλλά να τον περιορίσουμε για άλλες κλάσεις.
3. **private**: Τα private πεδία ή μέθοδοι είναι προσβάσιμα μόνο εντός της ίδιας κλάσης. Αυτή η ορατότητα χρησιμοποιείται για να προστατεύσει την ακεραιότητα των δεδομένων μιας κλάσης.
4. **χωρίς διαμορφωτή (default)**: Όταν δεν υπάρχει κανένας διαμορφωτής (ονομάζεται "package-private" ή "default"), τα μέλη είναι προσβάσιμα μόνο από κλάσεις του ίδιου πακέτου.

Παράδειγμα Χρήσης Διαμορφωτών Πρόσβασης

```
class Vehicle {
    public int speed;           // Προσβάσιμο από παντού
    protected int capacity;       // Προσβάσιμο μόνο σε υποκλάσεις και στο
ίδιο πακέτο
    private String type;        // Προσβάσιμο μόνο μέσα στην κλάση Vehicle

    // Μέθοδος για να αναθέτει τιμή στο private πεδίο type
    public void setType(String type) {
        this.type = type;
    }
```

```
    // Μέθοδος για να επιστρέφει το private πεδίο type
    public String getType() {
        return type;
    }
}

class Car extends Vehicle {
    public int numDoors;

    public void displayCarInfo() {
        System.out.println("Ταχύτητα: " + speed);      // Προσβάσιμο επειδή
είναι public
        System.out.println("Χωρητικότητα: " + capacity); // Προσβάσιμο
επειδή είναι protected
        // Δεν μπορούμε να έχουμε πρόσβαση απευθείας στο πεδίο type επειδή
είναι private
        System.out.println("Τύπος: " + getType());       // Χρησιμοποιούμε
τον public getter για πρόσβαση
    }
}
```

Στο παραπάνω παράδειγμα:

- Η μεταβλητή speed είναι public, επομένως είναι προσβάσιμη από οποιαδήποτε άλλη κλάση.
- Η μεταβλητή capacity είναι protected, επομένως είναι προσβάσιμη στην υποκλάση Car.
- Η μεταβλητή type είναι private, οπότε η Car δεν μπορεί να έχει άμεση πρόσβαση σε αυτήν. Για αυτόν τον λόγο, η Vehicle παρέχει έναν public getter (getType()) για πρόσβαση στην τιμή της.

Αυτή η διαχείριση ορατότητας βοηθά στην **ασφάλεια και ενθυλάκωση** των δεδομένων, αποτρέποντας την απευθείας πρόσβαση σε ευαίσθητα δεδομένα.

6.2.8. Η κλάση java.lang.Object

Η κλάση Object είναι η **βασική κλάση** για όλες τις κλάσεις στη Java. Κάθε κλάση στη Java είτε άμεσα είτε έμμεσα κληρονομεί από την Object. Αυτό σημαίνει ότι κάθε κλάση έχει πρόσβαση στις μεθόδους της κλάσης Object, ανεξαρτήτως του εάν δηλώνεται extends Object ή όχι.

Κύριες Μέθοδοι της java.lang.Object

Η κλάση Object παρέχει μερικές βασικές μεθόδους που είναι χρήσιμες και κληρονομούνται από όλες τις κλάσεις:

1. **toString()**: Αυτή η μέθοδος επιστρέφει μια String αναπαράσταση του αντικειμένου. Αν δεν την υπερκαλύψουμε, επιστρέφει την προεπιλεγμένη μορφή ClassName@Hashcode.

```
class Car {
    private String model;

    public Car(String model) {
        this.model = model;
    }
```

```
    @Override
    public String toString() {
        return "Car Model: " + model;
    }
}
```

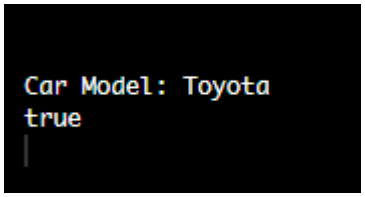
Εδώ η toString() υπερκαλύπτεται ώστε να εμφανίζει πιο κατανοητές πληροφορίες όταν καλείται.

1. **equals(Object obj)**: Αυτή η μέθοδος συγκρίνει δύο αντικείμενα για ισότητα. Από προεπιλογή, συγκρίνει τις διευθύνσεις μνήμης των αντικειμένων. Μπορεί όμως να υπερκαλυφθεί ώστε να συγκρίνει δεδομένα.
2. **hashCode()**: Επιστρέφει έναν αριθμητικό κωδικό (hash code) του αντικειμένου, ο οποίος χρησιμοποιείται σε δομές όπως τα HashMap και HashSet.

Παράδειγμα Χρήσης των Μεθόδων της Object

```
class Car {
    private String model;

    public Car(String model) {
        this.model = model;
    }

    @Override
    public String toString() {
        return "Car Model: " + model;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj) return true;
        if (obj == null || getClass() != obj.getClass()) return false;
        Car car = (Car) obj;
        return model.equals(car.model);
    }

    @Override
    public int hashCode() {
        return model.hashCode();
    }
}

public class Main {
    public static void main(String[] args) {
        Car car1 = new Car("Toyota");
        Car car2 = new Car("Toyota");

        System.out.println(car1.toString()); // Καλέσαμε την υπερκαλυμμένη
toString
        System.out.println(car1.equals(car2));     //     Καλέσαμε     την
υπερκαλυμμένη equals
    }
}
```

Έξοδος:



Εδώ:

- Υπερκαλύψαμε την toString() για να επιστρέψει την πληροφορία του Car.
- Υπερκαλύψαμε την equals() ώστε να συγκρίνει τα δεδομένα των αντικειμένων Car αντί για τις διευθύνσεις μνήμης.
- Υπερκαλύψαμε την hashCode() ώστε να επιστρέφει κωδικό βασισμένο στο μοντέλο του αυτοκινήτου.

6.2.9. Δημιουργία αντικειμένου με αναφορά στην υπερκλάση

Μία συχνή χρήση της κληρονομικότητας είναι η δημιουργία ενός αντικειμένου υποκλάσης και η χρήση μιας αναφοράς (reference) της υπερκλάσης για να το χειριστούμε. Αυτό επιτρέπει στον προγραμματιστή να χρησιμοποιεί τον **πολυμορφισμό** (polymorphism), καθώς μια υπερκλάση μπορεί να αναφέρεται σε οποιοδήποτε αντικείμενο της ίδιας της κλάσης ή οποιασδήποτε υποκλάσης της.

Παράδειγμα Δημιουργίας Αντικειμένου Υποκλάσης με Αναφορά στην Υπερκλάση

```
class Vehicle {
    public void startEngine() {
        System.out.println("Η μηχανή ξεκινά.");
    }
}

class Car extends Vehicle {
    public void playMusic() {
        System.out.println("Αναπαραγωγή μουσικής...");
    }
}

public class Main {
    public static void main(String[] args) {
        // Αναφορά της κλάσης Vehicle που δείχνει σε ένα αντικείμενο της
κλάσης Car
        Vehicle myVehicle = new Car();

        // Κλήση της μεθόδου startEngine που κληρονομήθηκε
        myVehicle.startEngine();

        // Η παρακάτω γραμμή προκαλεί σφάλμα καθώς η αναφορά Vehicle δεν
γνωρίζει τη μέθοδο playMusic
        // myVehicle.playMusic();
    }
}
```

Επεξήγηση:

- Η αναφορά myVehicle είναι τύπου Vehicle, αλλά δείχνει σε ένα αντικείμενο Car.
- Μπορούμε να καλέσουμε την startEngine() επειδή ανήκει στην Vehicle.
- Δεν μπορούμε να καλέσουμε την playMusic() εκτός αν κάνουμε μετατροπή τύπου (casting), καθώς η Vehicle δεν γνωρίζει την playMusic().

6.2.10. Μετατροπή Τύπου Casting

Στη Java, το casting είναι η διαδικασία μετατροπής ενός αντικειμένου από έναν τύπο σε έναν άλλο, συνήθως πιο συγκεκριμένο ή πιο γενικό τύπο. Υπάρχουν δύο είδη μετατροπής τύπου στην κληρονομικότητα:

1. **Ανερχόμενη μετατροπή (Upcasting)**: Μετατρέπει μια υποκλάση σε υπερκλάση της. Αυτό γίνεται αυτόματα και είναι χρήσιμο για την αναφορά υποκλάσεων με γενικές αναφορές (π.χ. Vehicle αντί Car).
2. **Κατιούσα μετατροπή (Downcasting)**: Μετατρέπει μια υπερκλάση σε υποκλάση της. Αυτό δεν γίνεται αυτόματα και απαιτεί να επιβεβαιώσουμε πρώτα ότι το αντικείμενο είναι του συγκεκριμένου τύπου για να αποφύγουμε σφάλματα εκτέλεσης.

Παράδειγμα Ανερχόμενης και Κατιούσας Μετατροπής

```
class Vehicle {
    public void startEngine() {
        System.out.println("Η μηχανή ξεκινά.");
    }
}

class Car extends Vehicle {
    public void playMusic() {
        System.out.println("Αναπαραγωγή μουσικής...");
    }
}

public class Main {
    public static void main(String[] args) {
        // Ανερχόμενη μετατροπή (upcasting)
        Vehicle myVehicle = new Car();

        // Μπορούμε να καλέσουμε τη μέθοδο startEngine, που ανήκει στην
Vehicle
        myVehicle.startEngine();

        // Κατιούσα μετατροπή (downcasting) για να καλέσουμε μεθόδους της
Car
        if (myVehicle instanceof Car) {
            Car myCar = (Car) myVehicle;
            myCar.playMusic();
        }
    }
}
```

Επεξήγηση:

- Με την ανερχόμενη μετατροπή Vehicle myVehicle = new Car();, μπορούμε να χρησιμοποιήσουμε το αντικείμενο Car μέσω της αναφοράς Vehicle.

- Με την κατιούσα μετατροπή (Car) myVehicle, αποκτούμε πρόσβαση σε μεθόδους που είναι μοναδικές στην Car, όπως η playMusic().

6.2.11. Παράδειγμα Δημιουργία Κληρονομικότητας

```
// Η υπερκλάση Vehicle
class Vehicle {
    String brand;  // Μάρκα του οχήματος
    int wheels;    // Αριθμός τροχών

    // Δημιουργός της κλάσης Vehicle
    Vehicle(String brand, int wheels) {
        this.brand = brand;   // Αρχικοποίηση της μάρκας
        this.wheels = wheels; // Αρχικοποίηση του αριθμού τροχών
    }

    // Μέθοδος για την εμφάνιση πληροφοριών του οχήματος
    void displayInfo() {
        System.out.println("Brand: " + brand);
        System.out.println("Wheels: " + wheels);
    }
}

// Η υποκλάση Car που επεκτείνει την Vehicle
class Car extends Vehicle {
    int seats; // Αριθμός καθισμάτων

    // Δημιουργός της κλάσης Car
    Car(String brand, int wheels, int seats) {
        super(brand,  wheels);  //  Κλήση  του  δημιουργού  της  υπερκλάσης
Vehicle
        this.seats = seats;  // Αρχικοποίηση του αριθμού καθισμάτων
    }

    // Υπερκαλυμμένη μέθοδος displayInfo
    @Override
    void displayInfo() {
        super.displayInfo();   //   Κλήση   της   μεθόδου   displayInfo   της
υπερκλάσης
        System.out.println("Seats: " + seats);
    }
}

// Η υποκλάση Truck που επεκτείνει την Vehicle
class Truck extends Vehicle {
    double loadCapacity; // Χωρητικότητα φορτίου σε τόνους

    // Δημιουργός της κλάσης Truck
    Truck(String brand, int wheels, double loadCapacity) {
        super(brand,  wheels);  //  Κλήση  του  δημιουργού  της  υπερκλάσης
Vehicle
        this.loadCapacity = loadCapacity; // Αρχικοποίηση της χωρητικότητας
φορτίου
    }

    // Υπερκαλυμμένη μέθοδος displayInfo
    @Override
    void displayInfo() {
        super.displayInfo();   //   Κλήση   της   μεθόδου   displayInfo   της
```

```
υπερκλάσης
        System.out.println("Load Capacity: " + loadCapacity + " tons");
    }
}

// Κύρια κλάση
public class Main {
    public static void main(String[] args) {
        // Δημιουργία αντικειμένου Car
        Car car = new Car("Toyota", 4, 5);
        System.out.println("Car Information:");
        car.displayInfo();

        System.out.println();

        // Δημιουργία αντικειμένου Truck
        Truck truck = new Truck("Volvo", 6, 18.5);
        System.out.println("Truck Information:");
        truck.displayInfo();
    }
}
```

Έξοδος:

```
Car Information:
Brand: Toyota
Wheels: 4
Seats: 5

Truck Information:
Brand: Volvo
Wheels: 6
Load Capacity: 18.5 tons
```

6.3. Υπερκάλυψη Μεθόδων (Method Overriding)

Η υπερκάλυψη (overriding) επιτρέπει στις υποκλάσεις να προσφέρουν μια **ειδική υλοποίηση** για μια μέθοδο που κληρονόμησαν από την υπερκλάση. Με την υπερκάλυψη, η μέθοδος της υποκλάσης καλείται όταν εκτελείται στο αντικείμενο της υποκλάσης, ακόμη κι αν το αντικείμενο αναφέρεται με τύπο της υπερκλάσης.

Σημειώσεις για την Υπερκάλυψη:

- Η υποκλάση πρέπει να έχει την ίδια υπογραφή μεθόδου (όνομα και παραμέτρους) με την υπερκαλυπτόμενη μέθοδο.
- Η υποκάλυψη είναι απαραίτητη για την επίτευξη **πολυμορφισμού**.

6.3.1. Παράδειγμα Υπερκάλυψης Μεθόδων

```
class Vehicle {
    public void startEngine() {
        System.out.println("Η μηχανή του οχήματος ξεκινά.");
    }
}

class Car extends Vehicle {
    @Override
    public void startEngine() {
        System.out.println("Η μηχανή του αυτοκινήτου ξεκινά με turbo.");
    }
}

public class Main {
    public static void main(String[] args) {
        Vehicle myVehicle = new Car();
        myVehicle.startEngine(); // Θα καλέσει την υπερκαλυμμένη μέθοδο της
Car
    }
}
```

Έξοδος: *Η μηχανή του αυτοκινήτου ξεκινά με turbo.*

6.4. Πολυμορφισμός

Ο πολυμορφισμός επιτρέπει στις κλάσεις να χρησιμοποιούν αντικείμενα διαφορετικών τύπων μέσω μιας κοινής διεπαφής ή υπερκλάσης. Με αυτόν τον τρόπο, ο ίδιος κώδικας μπορεί να δουλεύει με διαφορετικά είδη αντικειμένων, βελτιώνοντας την ευελιξία και την επαναχρησιμοποίηση.

6.4.1. Παράδειγμα Πολυμορφισμός

```
class Vehicle {
    public void drive() {
        System.out.println("Οδήγηση οχήματος.");
    }
}

class Car extends Vehicle {
    @Override
    public void drive() {
        System.out.println("Οδήγηση αυτοκινήτου.");
    }
}

class Bike extends Vehicle {
    @Override
    public void drive() {
        System.out.println("Οδήγηση μοτοσικλέτας.");
    }
}

public class Main {
```

```
    public static void main(String[] args) {
        Vehicle[] vehicles = {new Car(), new Bike()};

        for (Vehicle v : vehicles) {
            v.drive(); // Κάθε αντικείμενο καλεί τη δική του υλοποίηση της
μεθόδου drive
        }
    }
}
```

Έξοδος:

Οδήγηση αυτοκινήτου.

Οδήγηση μοτοσικλέτας.

Στο παραπάνω παράδειγμα:

Το ίδιο κομμάτι κώδικα (v.drive()) καλεί διαφορετικές υλοποιήσεις της drive(), ανάλογα με τον τύπο του αντικειμένου (πολυμορφισμός).

6.5. Αφηρημένες Κλάσεις (Abstract Classes)

Οι αφηρημένες κλάσεις επιτρέπουν στους προγραμματιστές να καθορίζουν κοινές έννοιες και μεθόδους που θα πρέπει να εφαρμόσουν οι υποκλάσεις, χωρίς όμως να απαιτείται πλήρης υλοποίηση. Μία αφηρημένη κλάση μπορεί να περιέχει αφηρημένες (χωρίς υλοποίηση) και μη αφηρημένες (με υλοποίηση) μεθόδους.

- Η αφηρημένη κλάση δεν μπορεί να δημιουργήσει αντικείμενο απευθείας.
- Κάθε υποκλάση της αφηρημένης κλάσης πρέπει να υλοποιεί όλες τις αφηρημένες μεθόδους της.

6.5.1. Παράδειγμα Αφηρημένης Κλάσης

```
abstract class Animal {
    public abstract void makeSound();

    public void sleep() {
        System.out.println("Το ζώο κοιμάται.");
    }
}

class Dog extends Animal {
    @Override
    public void makeSound() {
        System.out.println("Ο σκύλος γαβγίζει.");
    }
}

public class Main {
    public static void main(String[] args) {
        Dog myDog = new Dog();
        myDog.makeSound(); // Καλεί την υπερκαλυμμένη μέθοδο της Dog
        myDog.sleep();      // Καλεί την υλοποιημένη μέθοδο sleep από την
```

```
Animal
    }
}
```

Έξοδος:

Ο σκύλος γαβγίζει.

Το ζώο κοιμάται.

Στο παράδειγμα:

- Η Animal είναι αφηρημένη κλάση και περιέχει την αφηρημένη μέθοδο makeSound().
- Η Dog υλοποιεί τη μέθοδο makeSound(), ενώ η sleep() κληρονομείται απευθείας από την Animal.

6.6. Εισαγωγή στις Lambda Εκφράσεις

Οι Lambda εκφράσεις είναι ένας τρόπος για να ορίσουμε γρήγορα και εύκολα ανώνυμες μεθόδους. Συχνά χρησιμοποιούνται για σύντομες λειτουργίες σε συναρτησιακές διεπαφές (functional interfaces), όπως σε Java Collections και Streams. Οι Lambda εκφράσεις προσφέρουν πιο σύντομο και πιο ευανάγνωστο κώδικα, ειδικά όταν δουλεύουμε με επαναλαμβανόμενες λειτουργίες.

Βασική Σύνταξη Lambda

(παράμετροι) -> { έκφραση ή κώδικας }

Πλεονεκτήματα των Λειτουργιών Lambda

Οι Lambda εκφράσεις προσφέρουν αρκετά πλεονεκτήματα στον προγραμματισμό, ιδιαίτερα στη γλώσσα Java:

1. **Μείωση Κώδικα**: Οι Lambda εκφράσεις μειώνουν την ανάγκη για επιπλέον κώδικα, καθώς μπορούμε να ορίσουμε ανώνυμες συναρτήσεις χωρίς να χρειάζεται να δημιουργούμε κλάσεις.
2. **Ευανάγνωστος Κώδικας**: Ο κώδικας γίνεται πιο ευανάγνωστος, καθώς οι Lambda εκφράσεις επιτρέπουν την απλή και κομψή γραφή συναρτήσεων.

6.6.1. Παράδειγμα Χειρισμού Lambda Εκφράσεων (Lambda Expressions)

```
import java.util.Arrays;
import java.util.List;

public class Main {
    public static void main(String[] args) {
        List<String> names = Arrays.asList("Άννα", "Βασίλης", "Γιώργος");

        names.forEach(name -> System.out.println(name));
    }
}
```

Έξοδος:

```
Αννα
Βασίλης
Γιώργος
```

Στο παράδειγμα:

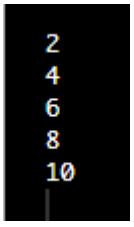
- Χρησιμοποιούμε Lambda για να περάσουμε μια συνάρτηση στο forEach() της λίστας names, η οποία τυπώνει κάθε στοιχείο.

6.6.2. Παράδειγμα Χρήσης Lambda με Streams

Οι Streams API διευκολύνουν τη δουλειά με συλλογές δεδομένων και οι Lambda εκφράσεις είναι συχνά χρήσιμες σε αυτές τις περιπτώσεις.

```
import java.util.Arrays;
import java.util.List;

public class Main {
    public static void main(String[] args) {
        List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9,
10);

        // Φιλτράρουμε τα ζυγά νούμερα και τυπώνουμε
        numbers.stream()
               .filter(n -> n % 2 == 0)  // Φιλτράρουμε για ζυγά νούμερα
               .forEach(n -> System.out.println(n)); // Τυπώνουμε
    }
}
```

Έξοδος:



Στο παραπάνω παράδειγμα:

- Η filter() μέθοδος χρησιμοποιεί μια Lambda έκφραση για να επιλέξει μόνο τα ζυγά νούμερα από τη λίστα.
- Η forEach() μέθοδος τυπώνει κάθε ζυγό νούμερο.

6.7. Πρακτική εξάσκηση

Άσκηση 1: Δημιουργία Κλάσης και Υποκλάσης

Δημιουργήστε μια κλάση Animal με μια μέθοδο makeSound(). Δημιουργήστε δύο υποκλάσεις Dog και Cat που υπερκαλύπτουν τη μέθοδο makeSound() με τις δικές τους υλοποιήσεις.

Άσκηση 2: Αφηρημένη Κλάση

Δημιουργήστε μια αφηρημένη κλάση Shape με μια αφηρημένη μέθοδο area(). Δημιουργήστε υποκλάσεις Circle και Rectangle που υλοποιούν την area().

Άσκηση 3: Πολυμορφισμός

Δημιουργήστε μια κλάση Vehicle με μια μέθοδο drive(). Δημιουργήστε υποκλάσεις Car και Bike που υπερκαλύπτουν τη μέθοδο drive(). Δημιουργήστε μια μέθοδο που δέχεται ένα Vehicle και καλεί τη μέθοδο drive().

Άσκηση 4: Casting

Δημιουργήστε κλάσεις Animal, Dog, και Cat. Δημιουργήστε ένα αντικείμενο Animal που αναφέρεται σε Dog και κάντε casting για να καλέσετε τη μέθοδο makeSound().

Άσκηση 5: Χρήση του super

Δημιουργήστε μια κλάση Person και μια υποκλάση Student. Στην υποκλάση, χρησιμοποιήστε το super για να καλέσετε τον κατασκευαστή της υπερκλάσης.

Άσκηση 6: Lambda Εκφράσεις

Δημιουργήστε μια λίστα από αριθμούς και χρησιμοποιήστε μια Lambda έκφραση για να υπολογίσετε το άθροισμά τους.

Άσκηση 7: Αφηρημένες Μεθόδοι

Δημιουργήστε μια αφηρημένη κλάση Account με αφηρημένες μεθόδους deposit(double amount) και withdraw(double amount). Δημιουργήστε υποκλάσεις SavingsAccount και CheckingAccount.

Άσκηση 8: Υπερκάλυψη

Δημιουργήστε μια κλάση Shape με μια μέθοδο draw(). Δημιουργήστε υποκλάσεις Circle και Square που υπερκαλύπτουν τη μέθοδο draw() και εκτυπώνουν ένα μήνυμα.

Άσκηση 9: Διαχείριση Μεθόδων

Δημιουργήστε μια κλάση Book με μια μέθοδο displayInfo(). Δημιουργήστε δύο υποκλάσεις EBook και PrintedBook και υπερκαλύψτε τη μέθοδο displayInfo().

Άσκηση 10: Collections με Lambda

Δημιουργήστε μια λίστα από κείμενα και χρησιμοποιήστε μια Lambda έκφραση για να φιλτράρετε μόνο τα κείμενα που περιέχουν τη λέξη "Java".

Ενδεικτικές Λύσεις

Άσκηση 1: Δημιουργία Κλάσης και Υποκλάσης

Δημιουργήστε μια κλάση Animal με μια μέθοδο makeSound(). Δημιουργήστε δύο υποκλάσεις Dog και Cat που υπερκαλύπτουν τη μέθοδο makeSound() με τις δικές τους υλοποιήσεις.

```
class Animal {
```

```
    public void makeSound() {
        System.out.println("Το ζώο κάνει ήχο.");
    }
}

class Dog extends Animal {
    @Override
    public void makeSound() {
        System.out.println("Ο σκύλος γαβγίζει.");
    }
}

class Cat extends Animal {
    @Override
    public void makeSound() {
        System.out.println("Η γάτα νιαούριζει.");
    }
}

public class Main {
    public static void main(String[] args) {
        Animal myDog = new Dog();
        Animal myCat = new Cat();

        myDog.makeSound(); // Ο σκύλος γαβγίζει.
        myCat.makeSound(); // Η γάτα νιαούριζει.
    }
}
```

Άσκηση 2: Αφηρημένη Κλάση

Δημιουργήστε μια αφηρημένη κλάση Shape με μια αφηρημένη μέθοδο area(). Δημιουργήστε υποκλάσεις Circle και Rectangle που υλοποιούν την area().

```
abstract class Shape {
    public abstract double area();
}

class Circle extends Shape {
    private double radius;

    public Circle(double radius) {
        this.radius = radius;
    }

    @Override
    public double area() {
        return Math.PI * radius * radius;
    }
}

class Rectangle extends Shape {
    private double width, height;

    public Rectangle(double width, double height) {
        this.width = width;
        this.height = height;
    }
```

```
    @Override
    public double area() {
        return width * height;
    }
}

public class Main {
    public static void main(String[] args) {
        Shape circle = new Circle(5);
        Shape rectangle = new Rectangle(4, 6);

        System.out.println("Area of Circle: " + circle.area());         //
78.53981633974483
        System.out.println("Area of Rectangle: " + rectangle.area()); //
24.0
    }
}
```

Άσκηση 3: Πολυμορφισμός

Δημιουργήστε μια κλάση Vehicle με μια μέθοδο drive(). Δημιουργήστε υποκλάσεις Car και Bike που υπερκαλύπτουν τη μέθοδο drive(). Δημιουργήστε μια μέθοδο που δέχεται ένα Vehicle και καλεί τη μέθοδο drive().

```
class Vehicle {
    public void drive() {
        System.out.println("Οδήγηση οχήματος.");
    }
}

class Car extends Vehicle {
    @Override
    public void drive() {
        System.out.println("Οδήγηση αυτοκινήτου.");
    }
}

class Bike extends Vehicle {
    @Override
    public void drive() {
        System.out.println("Οδήγηση μοτοσικλέτας.");
    }
}

public class Main {
    public static void testDrive(Vehicle vehicle) {
        vehicle.drive();
    }

    public static void main(String[] args) {
        Vehicle myCar = new Car();
        Vehicle myBike = new Bike();

        testDrive(myCar); // Οδήγηση αυτοκινήτου.
        testDrive(myBike); // Οδήγηση μοτοσικλέτας.
    }
}
```

Άσκηση 4: Casting

Δημιουργήστε κλάσεις Animal, Dog, και Cat. Δημιουργήστε ένα αντικείμενο Animal που αναφέρεται σε Dog και κάντε casting για να καλέσετε τη μέθοδο makeSound().

```
class Animal {
    public void makeSound() {
        System.out.println("Το ζώο κάνει ήχο.");
    }
}

class Dog extends Animal {
    @Override
    public void makeSound() {
        System.out.println("Ο σκύλος γαβγίζει.");
    }
}

public class Main {
    public static void main(String[] args) {
        Animal myAnimal = new Dog(); // Upcasting
        myAnimal.makeSound(); // Ο σκύλος γαβγίζει.

        // Downcasting
        Dog myDog = (Dog) myAnimal;
        myDog.makeSound(); // Ο σκύλος γαβγίζει.
    }
}
```

Άσκηση 5: Χρήση του super

Δημιουργήστε μια κλάση Person και μια υποκλάση Student. Στην υποκλάση, χρησιμοποιήστε το super για να καλέσετε τον κατασκευαστή της υπερκλάσης.

```
class Person {
    String name;

    public Person(String name) {
        this.name = name;
    }

    public void display() {
        System.out.println("Όνομα: " + name);
    }
}

class Student extends Person {
    public Student(String name) {
        super(name); // Καλεί τον κατασκευαστή της Person
    }
}

public class Main {
    public static void main(String[] args) {
        Student student = new Student("Γιάννης");
        student.display(); // Όνομα: Γιάννης
```

```
    }
}
```

Άσκηση 6: Lambda Εκφράσεις

Δημιουργήστε μια λίστα από αριθμούς και χρησιμοποιήστε μια Lambda έκφραση για να υπολογίσετε το άθροισμά τους.

```
import java.util.Arrays;
import java.util.List;

public class Main {
    public static void main(String[] args) {
        List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5);

        int sum = numbers.stream()
                         .mapToInt(Integer::intValue) // Μετατροπή σε int
                         .sum();

        System.out.println("Άθροισμα: " + sum); // Άθροισμα: 15
    }
}
```

Άσκηση 7: Αφηρημένες Μεθόδοι

Δημιουργήστε μια αφηρημένη κλάση Account με αφηρημένες μεθόδους deposit(double amount) και withdraw(double amount). Δημιουργήστε υποκλάσεις SavingsAccount και CheckingAccount.

```
abstract class Account {
    protected double balance;

    public abstract void deposit(double amount);
    public abstract void withdraw(double amount);
}

class SavingsAccount extends Account {
    @Override
    public void deposit(double amount) {
        balance += amount;
        System.out.println("Κατάθεση: " + amount);
    }

    @Override
    public void withdraw(double amount) {
        balance -= amount;
        System.out.println("Ανάληψη: " + amount);
    }
}

public class Main {
    public static void main(String[] args) {
        SavingsAccount myAccount = new SavingsAccount();
        myAccount.deposit(1000);
        myAccount.withdraw(200);
        System.out.println("Υπόλοιπο: " + myAccount.balance); // Υπόλοιπο:
800.0
    }
```

```
}
```

Άσκηση 8: Υπερκάλυψη

Δημιουργήστε μια κλάση Shape με μια μέθοδο draw(). Δημιουργήστε υποκλάσεις Circle και Square που υπερκαλύπτουν τη μέθοδο draw() και εκτυπώνουν ένα μήνυμα.

```
class Shape {
    public void draw() {
        System.out.println("Σχεδιάζω ένα σχήμα.");
    }
}

class Circle extends Shape {
    @Override
    public void draw() {
        System.out.println("Σχεδιάζω έναν κύκλο.");
    }
}

class Square extends Shape {
    @Override
    public void draw() {
        System.out.println("Σχεδιάζω ένα τετράγωνο.");
    }
}

public class Main {
    public static void main(String[] args) {
        Shape myCircle = new Circle();
        Shape mySquare = new Square();

        myCircle.draw(); // Σχεδιάζω έναν κύκλο.
        mySquare.draw(); // Σχεδιάζω ένα τετράγωνο.
    }
}
```

Άσκηση 9: Διαχείριση Μεθόδων

Δημιουργήστε μια κλάση Book με μια μέθοδο displayInfo(). Δημιουργήστε δύο υποκλάσεις EBook και PrintedBook και υπερκαλύψτε τη μέθοδο displayInfo().

```
class Book {
    public void displayInfo() {
        System.out.println("Πληροφορίες βιβλίου.");
    }
}

class EBook extends Book {
    @Override
    public void displayInfo() {
        System.out.println("Πληροφορίες ηλεκτρονικού βιβλίου.");
    }
}

class PrintedBook extends Book {
    @Override
```

```
    public void displayInfo() {
        System.out.println("Πληροφορίες έντυπου βιβλίου.");
    }
}

public class Main {
    public static void main(String[] args) {
        Book myEBook = new EBook();
        Book myPrintedBook = new PrintedBook();

        myEBook.displayInfo();      // Πληροφορίες ηλεκτρονικού βιβλίου.
        myPrintedBook.displayInfo(); // Πληροφορίες έντυπου βιβλίου.
    }
}
```

Άσκηση 10: Collections με Lambda

Δημιουργήστε μια λίστα από κείμενα και χρησιμοποιήστε μια Lambda έκφραση για να φιλτράρετε μόνο τα κείμενα που περιέχουν τη λέξη "Java".

```
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;

public class Main {
    public static void main(String[] args) {
        List<String> phrases = Arrays.asList("Αγαπώ την Java", "Η Python
είναι καλή", "Java είναι παντού");

        List<String> javaPhrases = phrases.stream()
                                          .filter(phrase                  ->
phrase.contains("Java"))
                                          .collect(Collectors.toList());

        System.out.println("Φράσεις με 'Java': " + javaPhrases); // Φράσεις
με 'Java': [Αγαπώ την Java, Java είναι παντού]
    }
}
```

6.8. Ερωτήσεις Αυτο-αξιολόγησης

1. **Τι είναι η κληρονομικότητα στην Java;**
   α) Δυνατότητα δημιουργίας αντικειμένων
   β) Μηχανισμός επαναχρησιμοποίησης κώδικα
   γ) Σχέση μεταξύ διαφορετικών κλάσεων
   δ) Όλα τα παραπάνω
2. **Ποιο από τα παρακάτω δεν είναι πλεονέκτημα της κληρονομικότητας;**
   α) Επαναχρησιμοποίηση κώδικα
   β) Αυξημένη πολυπλοκότητα
   γ) Καθαρός και οργανωμένος κώδικας
   δ) Εύκολη συντήρηση
3. **Ποιο είναι το σωστό keyword για τη δήλωση αφηρημένων κλάσεων;**

α) abstract

β) interface

γ) virtual

δ) inherit

4. **Τι συμβαίνει αν δεν υλοποιηθούν οι αφηρημένες μέθοδοι σε μια υποκλάση;**

α) Ο κώδικας θα τρέξει κανονικά

β) Θα προκληθεί σφάλμα κατά την εκτέλεση

γ) Θα προκληθεί σφάλμα κατά την μεταγλώττιση

δ) Θα αγνοηθούν

5. **Ποιες είναι οι υποχρεώσεις ενός κατασκευαστή (constructor) στην Java;**

α) Να έχει πάντα επιστροφή τύπου void

β) Να έχει το ίδιο όνομα με την κλάση

γ) Να είναι δημόσιος

δ) Να καλεί υποχρεωτικά τη μέθοδο finalize

6. **Ποιες είναι οι τύποι ορατότητας (visibility modifiers) στην Java;**

α) private, protected, public

β) private, public, package-private, protected

γ) private, package-private, public, internal

δ) private, public, global

7. **Τι είναι το casting στην Java;**

α) Δημιουργία νέας κλάσης

β) Μετατροπή ενός τύπου δεδομένων σε άλλο

γ) Υπερκάλυψη μεθόδων

δ) Δημιουργία αφηρημένων κλάσεων

8. **Ποια μέθοδος χρησιμοποιείται για τη διαχείριση συλλογών δεδομένων με Streams;**

α) collect()

β) transform()

γ) aggregate()

δ) process()

9. **Ποια είναι η σωστή σύνταξη μιας Lambda έκφρασης;**

α) (arguments) -> expression

β) (expression) -> arguments

γ) arguments -> (expression)

δ) expression -> (arguments)

10. **Ποιο από τα παρακάτω είναι ένα παράδειγμα χρήσης του forEach() με Lambda;**

α) list.forEach(item -> System.out.println(item));

β) forEach(item -> System.out.println(item));

γ) list.forEach(System.out::println);

δ) A και C

11. **Ποιο από τα παρακάτω δεν είναι χαρακτηριστικό των αφηρημένων κλάσεων;**

α) Περιέχουν αφηρημένες μεθόδους

β) Δεν μπορούν να έχουν κατασκευαστές

γ) Μπορούν να περιέχουν κανονικές μεθόδους

δ) Μπορούν να κληρονομηθούν

12. **Ποιο από τα παρακάτω είναι αποτέλεσμα της υπερκάλυψης μεθόδων;**

α) Καλούνται οι μέθοδοι της υπερκλάσης

β) Καλούνται οι μέθοδοι της υποκλάσης

γ) Δημιουργούνται νέες μέθοδοι

δ) Ακυρώνονται οι παλιές μέθοδοι

13. **Ποιες από τις παρακάτω μεθόδους δεν μπορούν να είναι αφηρημένες;**

α) Constructors

β) Static methods

γ) Instance methods

δ) Public methods

14. **Πώς δηλώνουμε μια Lambda έκφραση με πολλαπλές γραμμές;**

α) (params) -> { statements }

β) (params) => { statements }

γ) (params) : { statements }

δ) (params) -> statements

15. **Ποιο από τα παρακάτω είναι απαραίτητο για τη χρήση της κληρονομικότητας;**

α) Να υπάρχει τουλάχιστον μία υποκλάση

β) Να υπάρχει τουλάχιστον μία υπερκλάση

γ) Να είναι οι κλάσεις δημόσιες

δ) Να έχουν κοινά χαρακτηριστικά

6.9. Απαντήσεις στις ερωτήσεις Αυτο-αξιολόγησης

1. δ	**2. β**	**3. α**
4. γ	**5. β**	**6. β**
7. β	**8. α**	**9. α**
10. δ	**11. β**	**12. β**
13. α	**14. α**	**15 β**

ΚΕΦΑΛΑΙΟ 7: Χρήση Interface, Java API Documentation, Collections, Χειρισμός Exceptions (Handling Exceptions) και Logging

7.1. Εκπαιδευτικοί στόχοι του παρόντος κεφαλαίου

Οι εκπαιδευτικοί στόχοι του παρόντος κεφαλαίου, συνοψίζονται στα κάτωθι σημεία. Οι εκπαιδευόμενοι:

- θα γνωρίσουν τη χρήση των Interfaces με παραδείγματα,
- θα πραγματοποιήσουν μια εισαγωγή στις Γενικεύσεις (Generics),
- θα γνωρίσουν το Java API Documentation,
- θα μάθουν τη χρήση των Collections Framework σε ότι αφορά τις λίστες (Lists), τις απεικονίσεις (Maps), τα σύνολα (Sets),
- θα γνωρίσουν το χειρισμό των Exceptions (Handling Exceptions),
- θα μάθουν τη χρήση του try-with-resources Μπλοκ,
- θα μάθουν τον τρόπο διαχείρισης Πολλαπλών Εξαιρέσεων,
- θα μάθουν πως γίνεται η δήλωση Εξαιρέσεων (Exceptions) σε μέθοδο,
- θα μάθουν πως γίνεται η διάδοση εξαιρέσεων (Throwing Exceptions),
- θα μάθουν πως γίνεται η δημιουργία νέων εξαιρέσεων (Exceptions),
- θα γνωρίσουν την έννοια των Assertions,
- θα μάθουν τη χρήση του Java Logging API σε ότι αφορά τα επίπεδα logging, τη παραμετροποίηση και καλές πρακτικές.

7.2. Χρήση Interface

Από την εποχή των πρώτων προγραμματιστών έχουν γίνει προσπάθειες να αυξηθεί η παραγωγικότητα τους με χρήση τμημάτων κώδικα που μπορούν να επαναχρησιμοποιηθούν. Σε αυτή την ενότητα θα παρουσιαστεί μία τεχνική που διαχωρίζει τον επαναχρησιμοποιούμενο κώδικα από εκείνον που διαφέρει σε κάθε περίπτωση χρήσης του. Το επαναχρησιμοποιούμενο τμήμα κώδικα εμπλέκει τις μεθόδους ενός τύπου που ονομάζεται interface και συνδυάζεται με μία κλάση που υλοποιεί τις μεθόδους του. Η λογική της χρήσης του interface βασίζεται στην υλοποίηση μίας διαφορετικής κλάσης για το ίδιο interface που οδηγεί στην αλλαγή συμπεριφοράς του. Αυτό ονομάζεται πολυμορφισμός. Για ποιο λόγο είναι χρήσιμα τα interfaces; Ακολουθεί ένα παράδειγμα με δύο τμήματα κώδικα (δύο μεθόδους) που δείχνει τη χρησιμότητα τους.

```
public void add1(double x)
{
	sum=sum+x;
	if (count==0||max<x) max=x;
	count++;
}
public void add2(BankAccount x)
{
```

```
        sum=sum+x.getBalance();
        if (count==0||max.getBalance()<x.getBalance()) max=x;
        count++;
}
```

Ξεκάθαρα ο αλγόριθμος για την επεξεργασία των δεδομένων και στις δύο μεθόδους, για την κλάση των αριθμών και για την κλάση των τραπεζικών λογαριασμών, είναι ο ίδιος με μόνη διαφορά το είδος των δεδομένων που επεξεργάζεται και η αντίστοιχη μέτρηση.

Έστω πως δημιουργείται μία μέθοδος που θα προσπαθούσε να καλύψει και τις δύο εκδοχές του προηγούμενου κώδικα. Τότε αυτή θα έμοιαζε με το επόμενο τμήμα κώδικα.

```
public void add(ΓενικόςΤύπος x)
{
        sum=sum+x.getMetric();
        if (count==0||max.getMetric()<x.getMetric()) max=x;
        count++;
}
```

Αυτή τη δυνατότητα μας την προσφέρει η υλοποίηση των κλάσεων μέσω ενός Interface το οποίο θα είχε την ακόλουθη μορφή.

```
public interface Metric
{
        double getMetric();
}
```

7.2.1. Το Java Interface

Προφανώς σε μία πραγματική δήλωση ενός Interface θα υπήρχαν περισσότερες μέθοδοι ενώ ο τύπος με Metric είναι δική μας επιλογή. Η γενική σύνταξη ενός Interface είναι προφανής.

```
public interface <Όνομα Interface>
{
        Ονόματα και τύποι μεθόδων
}
```

Πρέπει να σημειωθούν οι ακόλουθες παρατηρήσεις:

- Ένα interface είναι παρόμοιο με μία κλάση αλλά έχει κάποιες σημαντικές διαφορές.
- Όλες οι μέθοδοι του interface είναι abstract και όλες οι απαραίτητες πληροφορίες εμπεριέχονται εντός της υλοποίησης.
- Όλες οι μέθοδοι του είναι αυτομάτως τύπου public.
- Το interface δεν έχει μεταβλητές.

Η υλοποίηση ενός interface είναι μία κλάση που υλοποιεί ένα ή περισσότερα interfaces αν κατά τη δήλωσή της χρησιμοποιηθεί η δεσμευμένη λέξη implements. Στη συνέχεια πρέπει να υλοποιηθούν και οι μέθοδοι των interfaces. Δείτε το ακόλουθο παράδειγμα με ένα σχετικό τμήμα κώδικα.

```
public class ClassName implements InterfaceName1, InterfaceName2
{
        Μεταβλητές
        Μέθοδοι
}
```

Συνοπτικά, στο προηγούμενο παράδειγμα το interface Metric εκφράζει όλα τα μετρήσιμα αντικείμενα που έχουν κοινά στοιχεία. Η ομοιότητα τους μας δίνει την ευελιξία να δημιουργήσουμε έναν κώδικα που μπορεί να επαναχρησιμοποιηθεί. Ας δούμε όμως ένα ολοκληρωμένο παράδειγμα δημιουργίας και υλοποίησης ενός Interface.

Συμπερασματικά, το interface είναι μία συλλογή δηλώσεων μεθόδων χωρίς δεδομένα και χωρίς τα μπλοκ του κώδικα τους. Αυτό σημαίνει πως το interface αποτελείται από κενές μεθόδους (είναι απλώς οι υπογραφές τους). Όταν μία κλάση υλοποιεί το interface τότε πρέπει να υλοποιηθούν όλες οι μέθοδοι που έχουν δηλωθεί σε αυτό. Στην ουσία με αυτό τον τρόπο το interface υποχρεώνει τον προγραμματιστή να υλοποιήσει όλες τις προδιαγραφές της κλάσης.

7.2.2. Παράδειγμα Δημιουργίας Interface

Ακολουθεί ένα πλήρες παράδειγμα δημιουργίας ενός interface και στη συνέχεια της υλοποίησης του με μία κλάση. Το interface με το οποίο αρχίζουμε είναι το Metric που έχει παρουσιαστεί σε προηγούμενη υπο-ενότητα. Το αρχείο όπου αποθηκεύεται είναι το Metric.java.

```
public interface Metric
{
      double metric(Object instObject);
}
```

Στη συνέχεια το interface υλοποιείται ως τάξη RectangleMetric το οποίο θα επιστρέφει το εμβαδό ενός ορθογώνιου παραλληλογράμμου. Το αρχείο όπου αποθηκεύεται είναι το RectangleMetric.java.

```
import java.awt.Rectangle;

public class RectangleMetric implements Metric
{
   public double metric(Object anObject)
   {
      Rectangle instRectangle = (Rectangle) instObject;
      double area = instRectangle.getWidth() * instRectangle.getHeight();
      return area;
   }
}
```

Έπειτα ακολουθεί ένα μεγαλύτερο τμήμα κώδικα που περιλαμβάνει όλους τους υπολογισμούς για όλα τα σύνολα δεδομένων. Πιο συγκεκριμένα υπολογίζει τα πιο απλά περιγραφικά στατιστικά. Το αρχείο αποθηκεύεται με το όνομα CalculateDataSet.java.

```
public class CalculateDataSet
{
   private double sum;
   private Object max;
   private int count;
   private Metric metric;

   public DataSet(Metric insMetric)
   {
      sum = 0;
      count = 0;
      max = null;
      measurer = insMetric;
```

```
    }

    public void add(Object x)
    {
       sum = sum + metric.metric(x);
       if (count == 0 || metric.metric(maximum) < metric.metric(x))
          max = x;
       count++;
    }

    public double getMean()
    {
       if (count == 0) return 0;
       else return sum / count;
    }

    public Object getMax()
    {
       return max;
    }
}
```

Το παράδειγμα ολοκληρώνεται με μία κλάση που περιέχει τη μέθοδο main για να ελεγχθεί το σύνολο του παραπάνω κώδικα. Το αρχείο έχει ως όνομα CalculationTester.java.

```
import java.awt.Rectangle;

public class CalculationTester
{
   public static void main(String[] args)
   {
      Metric m = new RectangleMetric();

      CalculateDataSet dataSet = new CalculateDataSet(m);

      dataSet.add(new Rectangle(5, 10, 20, 30));
      dataSet.add(new Rectangle(10, 20, 30, 40));
      dataSet.add(new Rectangle(20, 30, 5, 15));

      System.out.println("Εμβαδό: " + dataSet.getMean());

      Rectangle max = (Rectangle) dataSet.getMax();
      System.out.println("Μέγιστο εμαβδό: " + max);
   }
}
```

7.3. Εισαγωγή στις Γενικεύσεις (Generics)

Ο προγραμματισμός με γενικεύσεις σχεδιάζει και υλοποιεί τις δομές δεδομένων και τους αλγόριθμους έτσι ώστε να λειτουργούν με πολλαπλούς τύπους δεδομένων. Στην πραγματικότητα, ο προγραμματισμός με γενικεύσεις μπορεί να υλοποιηθεί είτε με κληρονομικότητα είτε με παραμέτρους τύπων. Ο προγραμματιστής Java θα πρέπει να κατανοεί τη λογική του προγραμματισμού με γενικεύσεις, να υλοποιεί γενικεύσιμες κλάσεις και μεθόδους, να γνωρίζει τον τρόπο εκτέλεσης των γενικεύσεων από την εικονική μηχανή της Java καθώς και τους περιορισμούς που έχει αυτή η χρήσιμη δυνατότητα. Μία generic κλάση είναι μία κλάση με μία παράμετρο τύπου που χρειάζεται για να προσδιοριστεί ο τύπος των αντικειμένων που θα αποθηκεύονται. Όταν

δηλώνεται μια generic κλάση, προσδιορίζεται ο τύπος της μεταβλητής για κάθε παράμετρο τύπου. Ακολουθεί ένα σχετικό παράδειγμα δήλωσης μίας τέτοιας κλάσης.

```
public class ArrayQueue <E>
{
   public ArrayQueue(){ …}
   public void addItem(E element){ …}
}
```

Ως παράμετρος χρησιμοποιείται το γράμμα Ε που είναι η μεταβλητή τύπου η οποία προφανώς δεν είναι δεσμευμένη λέξη της γλώσσας. Συμβατικά στις παραμέτρους τύπων χρησιμοποιούνται κεφαλαία λατινικά γράμματα ενώ θα μπορούσαν να χρησιμοποιηθούν οποιαδήποτε συμβολικά ονόματα. Για να χρησιμοποιηθεί μία generic κλάση θα πρέπει να δημιουργηθεί μία πραγματική της υπόσταση με την απόδοση ενός πραγματικού τύπου στην παράμετρο τύπου. Ως παράδειγμα, στην γενίκευση ArrayQueue δίνεται το ArrayQueue<Customer> και ArrayQueue<BusinessCustomer>.

Πρέπει να σημειωθεί πως δεν είναι δυνατή η χρήση των οκτώ πρωτογενών τύπων της Java ως τιμές σε παραμέτρους τύπων. Όπως είναι αναμενόμενο η υποκατάσταση της παραμέτρου με τον πραγματικό τύπο γίνεται αυτομάτως σε όλο το σώμα της κλάσης. Έτσι η μέθοδος addItem() του ανωτέρω παραδείγματος γίνεται public void addItem(Customer element). Η παραπάνω ενέργεια δίνει μεγαλύτερη ασφάλεια στον κώδικα. Ένα άλλο πλεονέκτημα αυτής προσέγγισης είναι πως ο κώδικας είναι πιο ευανάγνωστος.

Κατά αναλογία με την κλάση, μία μέθοδος είναι generic όταν έχει έναν τύπο ως παράμετρο. Μία τέτοια μέθοδος δύναται να εμφανιστεί και ως τμήμα μίας κλάσης που δεν είναι υποχρεωτικά generic. Αυτή η μέθοδος μπορεί να θεωρηθεί ως πρότυπο για μία ομάδα μεθόδων που διαφέρουν ως προς τον τύπο των παραμέτρων τους. Για παράδειγμα, όταν δηλώνεται μία δομή οποιουδήποτε τύπου όπως στο τμήμα κώδικα που ακολουθεί.

```
public class ArrayHandle
{
   public static <T> void display (T[] a)
     {
         for(T t: a)
              System.out.print(t+" ");
         System.out.println();
     }
}
```

Εδώ πρέπει να σημειωθεί πως ο τύπος μεταβλητής <T> τοποθετείται πριν από τον τύπο επιστροφής της τιμής (void). Επίσης στο σώμα της μεθόδου το t είναι μία τοπική μεταβλητή με μεταβλητό τύπο δεδομένων. Όταν καλείται μία generic μέθοδος, τότε πρέπει να προσδιοριστεί ο πραγματικός τύπος για την παράμετρο τύπου (αυτό είναι μία διαφορά από την generic κλάση). Το ταίριασμα των παραμέτρων γίνεται από το μεταφραστή (compiler). Ακολουθεί τμήμα κώδικα που σχετίζεται με την προηγούμενη μέθοδο.

```
Rectangle[] rectangles = …;
ArrayHandle.display(rectangles);
```

Ο τύπος της παραμέτρου της μεταβλητής rectangles είναι Rectangle[] και ο τύπος της παραμέτρου μεταβλητής είναι T[]. Έτσι ο μεταφραστής λαμβάνει το T ως Rectangle. Πρέπει να προστεθεί πως είναι εφικτή η δήλωση μεθόδων generic που δεν είναι static και μεθόδων μέσα σε generic κλάσεις.

Ο μόνος περιορισμός είναι η μη δυνατότητα αντικατάστασης των τύπων με τους οκτώ πρωτογενείς τύπους (όπως και στις κλάσεις). Κάποιες φορές είναι αναγκαίος ο περιορισμός των τύπων που μπορούν να χρησιμοποιηθούν ως παράμετρος μίας generic κλάσης ή μεθόδου. Αυτό επιτυγχάνεται με την υλοποίηση ενός interface όπως φαίνεται στον ακόλουθο κώδικα.

```
public static <E extends Comparable> E max(E[] a)
{
   E largest = a[0];
   for(int i=1; i<a.length; i++)
       if(a[i].compareTo(largest)>0) largest = a[i];
   return largest;
}
```

Όταν πρέπει να ικανοποιούνται δύο περιορισμοί τότε υπάρχει η αντίστοιχη δυνατότητα με χρήση του τελεστή &: <E extends Comparable & Cloneable>. Οι περιορισμοί είναι κλάσεις ή interfaces. Εδώ πρέπει να σημειωθεί πως επειδή τα generics είναι πιο πρόσφατη προσθήκη στην Java, η εικονική μηχανή της δεν δουλεύει με τις αρχικές κλάσεις και μεθόδους. Στην πραγματικότητα «διαγράφει» τους τύπους παραμέτρων και τους αντικαθιστά με τον τύπο object. Αυτός είναι και ο βασικός περιορισμός τους καθώς δεν είναι δυνατή η δημιουργία νέων αντικειμένων ενός generic τύπου. Αυτό ονομάζεται εξάλειψη (erasure).

7.3.1. Παράδειγμα Χειρισμού Γενικεύσεων (Generics)

Ακολουθεί ένα απλό παράδειγμα κώδικα που υλοποιείται με χρήση generics. Ο κώδικας θα αποθηκεύει το ονοματεπώνυμο και το ΑΦΜ ενός φορολογουμένου και θα διαθέτει δύο συναρτήσεις get για να επιστρέφει τις δύο πληροφορίες. Η κλάση Pair είναι generic.

```
public class Pair <N, A> // Η κλάση συλλέγει ένα ζεύγος στοιχείων με
διαφορετικό τύπο.
{
   private N first;
   private A second;

   public Pair(N firstItem, A secondItem){ // Constructor
          first = firstItem;
          second = secondItem;
    }

   public getFirst(){
          return first;
    }
   public getSecond(){
          return second;
    }
   public toString(){
          return "{"+first+","+second+"}";
    }
}
```

```
public class PairTest
{
   public static void main(String[] args)
   {
      String[] names = {"Γιάννης","Γιώργος","Λένα"};

      Pair <String, Integer> result =firstContaining(names,1234);

      System.out.println(result.getFirst());
      System.out.println(result.getSecond());
   }
}

public static Pair <String, Integer> firstContaining(String[] strings,
String sub)
{
   for (int i=0; i<strings.length; i++)
   {
      if string[i].contains(sub)
             return new Pair<String, Integer> (String[i],i);
      return new Pair<String, Integer> (null,-1);
   }
}
```

Η δεύτερη κλάση PairTest απλώς χρησιμοποιείται για τον έλεγχο της γενικεύσιμης κλάσης. Τα δεδομένα που προστίθενται είναι μια σειρά από ονόματα και ένας ακέραιος αριθμός.

7.4. Java API Documentation

Η Java όπως και η πλειοψηφία των σύγχρονων γλωσσών προγραμματισμού υψηλού επιπέδου παρέχει ένα σύνολο από βιβλιοθήκες και εργαλεία που συμπληρώνουν τις βασικές δομές και λειτουργίες. Προφανώς οι προγραμματιστές μπορούν να χρησιμοποιήσουν τις έτοιμες βιβλιοθήκες για τη σύνταξη του δικού τους κώδικα. Φυσικά, ο προγραμματιστής- χρήστης μίας βιβλιοθήκης δεν απαιτείται να γνωρίζει τον τρόπο υλοποίησης της βιβλιοθήκης αλλά μόνο τον τρόπο χρήσης της. Πιο συγκεκριμένα, ο προγραμματιστής πρέπει να γνωρίζει τις εισόδους και την έξοδο της βιβλιοθήκης, δηλαδή να γνωρίζει το interface (διεπαφή).

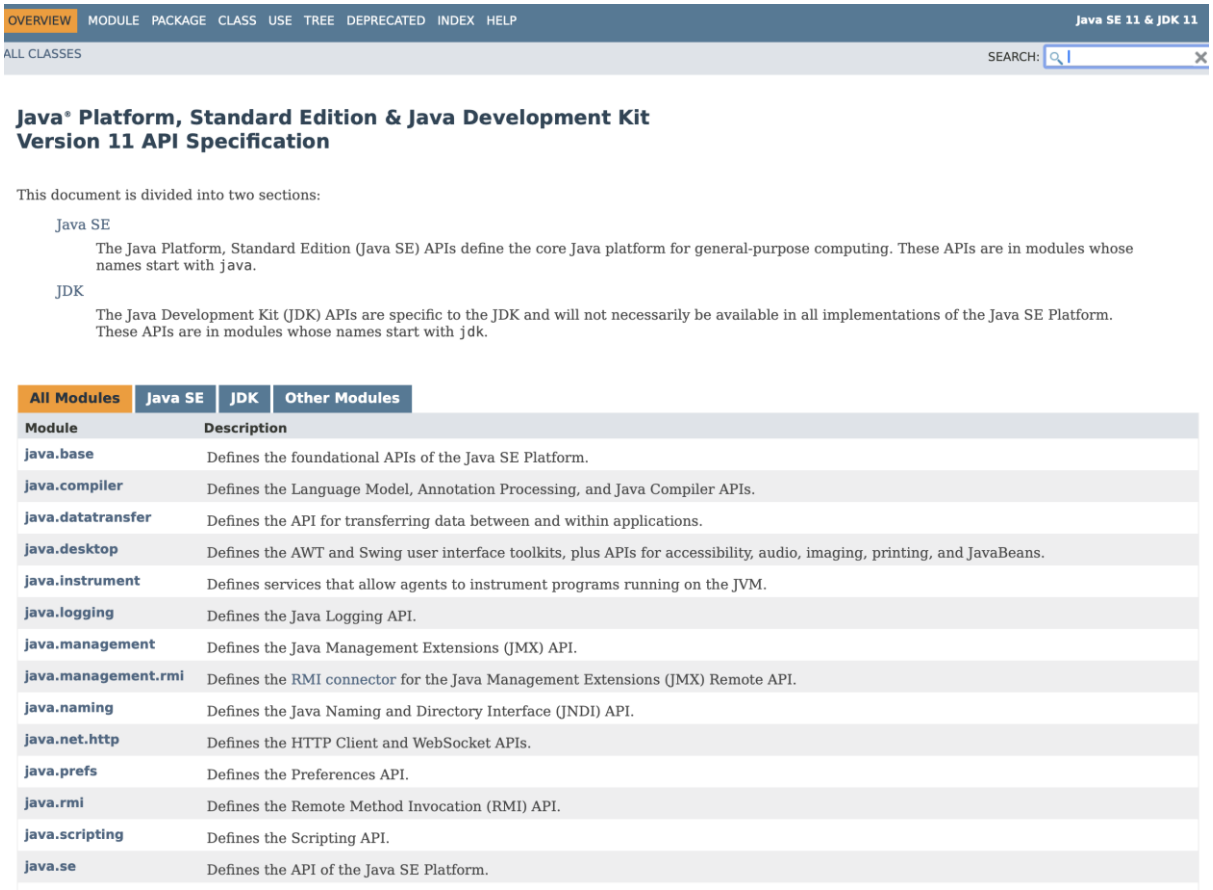
Ο όρος Java API (Application Programming Interface) περιλαμβάνει το σύνολο των interfaces των βιβλιοθηκών που έρχονται μαζί με την τυπική εγκατάσταση της Java. Συνεπώς, ο προγραμματιστής Java γνωρίζοντας αυτά τα API μπορεί να κάνει χρήση των έτοιμων βιβλιοθηκών και πλαισίων χωρίς να υπεισέρχεται στις λεπτομέρειες της υλοποίηση τους. Βέβαια η Java συνοδεύεται από μία τεκμηρίωση (documentation) των API, το API documentation που περιέχει την περιγραφή τους. Η κάθε έκδοση της Java συνοδεύεται από μία τέτοια τεκμηρίωση η οποία μπορεί εύκολα να βρεθεί στο διαδικτυακό τόπο της Oracle. Στον παρόντα εκπαιδευτικό οδηγό μπορείτε να βρείτε το Java API Documentation για την έκδοση 11.

7.4.1. Java Platform SE and JDK Version 11 API Specification

Το Java API Documentation έχει συνταχθεί με μία τυποποιημένη μορφή για να τηρηθεί η συνέπεια και η ομοιόμορφη ποιότητα ενώ η Java προσφέρει και ένα εξειδικευμένο εργαλείο το Javadoc το

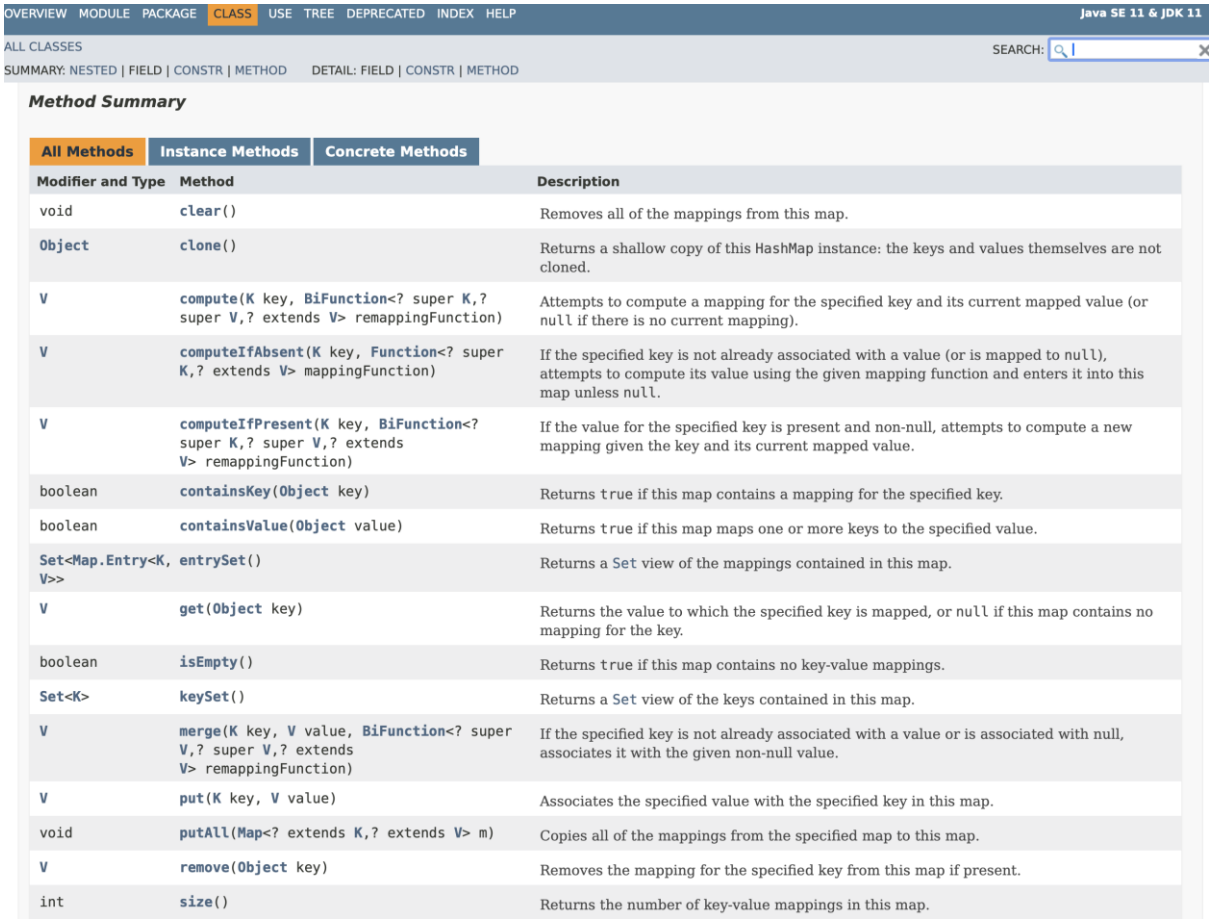
οποίο σκανάρει τον κώδικα που γράφεται από τον προγραμματιστή και εμπεριέχει σχόλια ειδικής διαμόρφωσης για δημιουργηθεί ένα API Documentation για αυτόν. Το εργαλείο Javadoc βασίζεται στην ύπαρξη σχολίων που τοποθετούνται ακριβώς πριν τον κώδικα μίας κλάσης, μεθόδου, πεδίου ή constructor. Η πρώτη πρόταση πρέπει να είναι μία σύνοψη (σύντομη περιγραφή της οντότητας για την οποία δημιουργείται το documentation). Η σύντομη περιγραφή τελειώνει με την πρώτη τελεία η οποία ακολουθείται το κενό, tab, enter ή κάποιο tag. Τα tags προσφέρουν τη δυνατότητα προσδιορισμού συγκεκριμένων πληροφοριών και σε αυτά συμπεριλαμβάνονται τα εξής: @author, @param, @throw, @code και @literal.

Στην εικόνα που ακολουθεί παρουσιάζεται η μορφή του τυπικού Java Api Specification για την έκδοση 11.



7.4.2. Java Platform SE 11: Method Summary

Η τυπική μορφή του Method Summary, όπως προκύπτει από το Java API Documentation, είναι ένας πίνακας με τρεις στήλες που εμφανίζουν τον τροποποιητή και τον τύπο (πρώτη στήλη), τη μέθοδο και τις παραμέτρους της (δεύτερη στήλη) και τη συνοπτική περιγραφή της μεθόδου (τρίτη στήλη). Ακολουθεί μία εικόνα για το περιεχόμενο της Method Summary της κλάσης HashMap <K, V> όπως εμφανίζεται στην τεκμηρίωση της έκδοσης 11.



OVERVIEW MODULE PACKAGE CLASS USE TREE DEPRECATED INDEX HELP — Java SE 11 & JDK 11

ALL CLASSES — SEARCH:

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

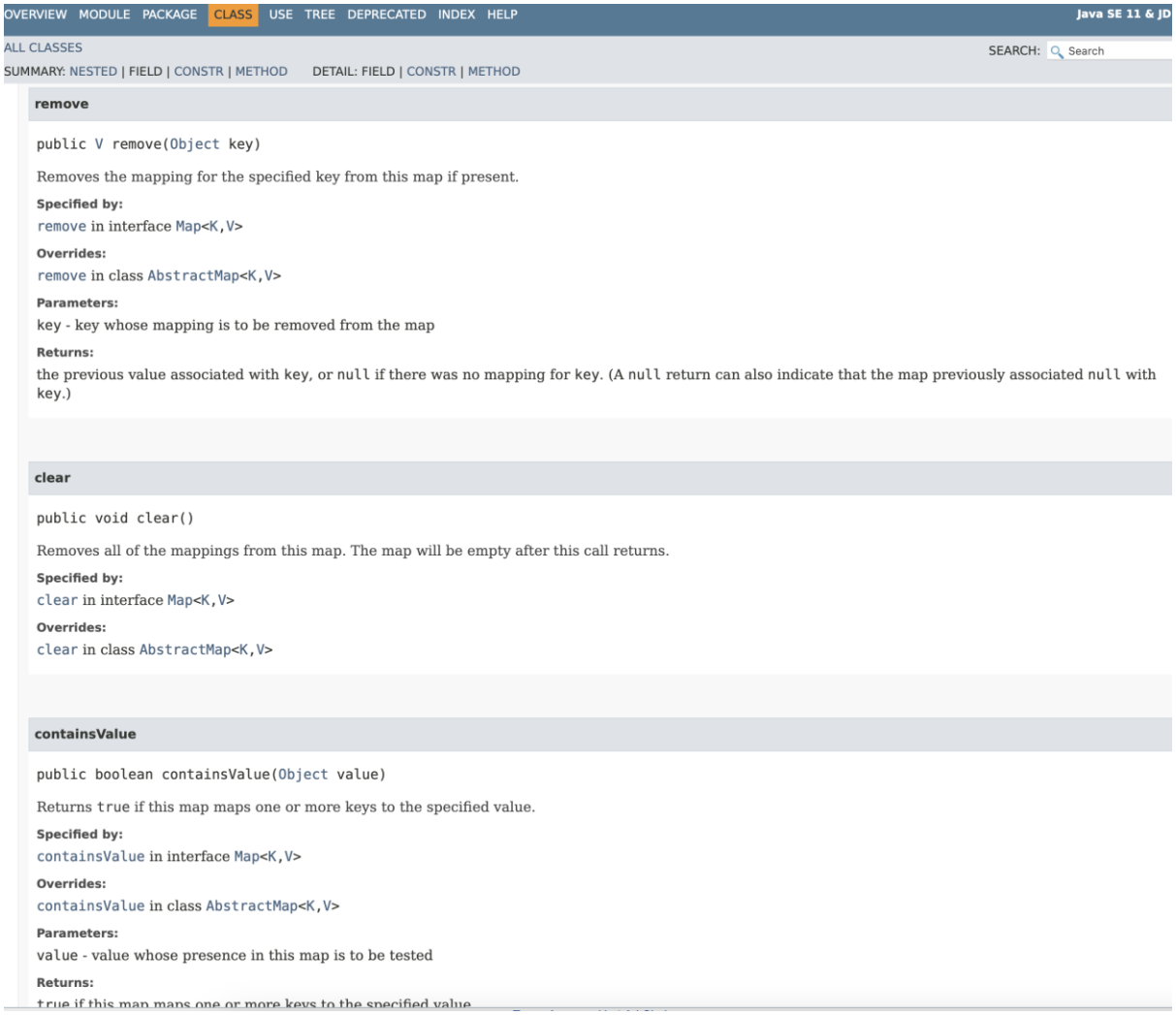
Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method	Description
void	clear()	Removes all of the mappings from this map.
Object	clone()	Returns a shallow copy of this HashMap instance: the keys and values themselves are not cloned.
V	compute(K key, BiFunction<? super K,? super V,? extends V> remappingFunction)	Attempts to compute a mapping for the specified key and its current mapped value (or null if there is no current mapping).
V	computeIfAbsent(K key, Function<? super K,? extends V> mappingFunction)	If the specified key is not already associated with a value (or is mapped to null), attempts to compute its value using the given mapping function and enters it into this map unless null.
V	computeIfPresent(K key, BiFunction<? super K,? super V,? extends V> remappingFunction)	If the value for the specified key is present and non-null, attempts to compute a new mapping given the key and its current mapped value.
boolean	containsKey(Object key)	Returns true if this map contains a mapping for the specified key.
boolean	containsValue(Object value)	Returns true if this map maps one or more keys to the specified value.
Set<Map.Entry<K, V>>	entrySet()	Returns a Set view of the mappings contained in this map.
V	get(Object key)	Returns the value to which the specified key is mapped, or null if this map contains no mapping for the key.
boolean	isEmpty()	Returns true if this map contains no key-value mappings.
Set<K>	keySet()	Returns a Set view of the keys contained in this map.
V	merge(K key, V value, BiFunction<? super V,? super V,? extends V> remappingFunction)	If the specified key is not already associated with a value or is associated with null, associates it with the given non-null value.
V	put(K key, V value)	Associates the specified value with the specified key in this map.
void	putAll(Map<? extends K,? extends V> m)	Copies all of the mappings from the specified map to this map.
V	remove(Object key)	Removes the mapping for the specified key from this map if present.
int	size()	Returns the number of key-value mappings in this map.

7.4.3. Java Platform SE 11: Method Detail

Κάνοντας κλικ στο όνομα μίας μεθόδου που φαίνεται στο προηγούμενο στιγμιότυπο της JAVA Platform SE 11: Method Summary θα προκύψει μία εικόνα όπως η επόμενη που μας δείχνει τις λεπτομέρειες της μεθόδου (σύντομη περιγραφή, παράμετροι, επιστρεφόμενη τιμή και αν γίνεται κάποιο override.)



7.5. Collections Framework

Κάθε ομάδα ξεχωριστών αντικειμένων που εκπροσωπούν μία μοναδική οντότητα ονομάζεται συλλογή αντικειμένων της Java (Java Collection). Από το JDK1.2 έχει δημιουργηθεί ένα ξεχωριστό πλαίσιο (framework) που ονομάζεται Collections Framework και περιέχει όλε τις σχετικές κλάσεις και interfaces. Γενικά το πλαίσιο (framework) είναι ένα σύνολο κλάσεων και interfaces που δίνουν στον προγραμματιστή μία έτοιμη αρχιτεκτονική έτσι ώστε να υλοποιεί μία κλάση χωρίς να πρέπει να ορίσει το ίδιο το πλαίσιο της.

Το πλαίσιο είναι μία βέλτιστη αντικειμενοστραφής τεχνική σχεδίαση γιατί όλες οι κλάσεις της συλλογής εκτελούν το ίδιο είδος ενέργειας. Οι συλλογές θεωρούνται σημαντικό συστατικό στοιχείο τους προγραμματισμού σε Java καθώς προσφέρουν έτοιμες λύσεις σε συνήθεις δομές δεδομένων όπως οι λίστες, απεικονίσεις και τα σύνολα. Τα πλεονεκτήματα της χρήσης του πλαισίου collections framework είναι τα ακόλουθα:

- Συνέπεια για το κάθε API που έχει ένα βασικό σύνολο interfaces όπως Collection, Set, List και Map και όλες οι κλάσεις που τις υλοποιούν και έχουν κοινό σύνολο μεθόδων.

- Μείωση του προγραμματιστικού φόρτου γιατί ο προγραμματιστής εστιάζει στο κύριο αντικείμενο του κώδικα και δεν ασχολείται με τη σχεδίαση της συλλογής υλοποιώντας έτσι τη βασική ιδέα του αντικειμενοστραφούς προγραμματισμού.
- Αύξηση της απόδοσης και της ποιότητας του κώδικα καθώς ο προγραμματιστής δεν ασχολείται με τη σχεδίαση και υλοποίηση μίας δομής δεδομένων. Απλά χρησιμοποιεί την καλύτερη, κατά την κρίση του, υλοποίηση της για να αυξήσει την απόδοση του κώδικα του.

Το πακέτο που περιέχει όλες τις κλάσεις και τα interfaces που απαιτούνται από το πλαίσιο Collections είναι το java.util. Εκεί υπάρχει το interface iterator που επεκτείνεται από το Collections το οποίο και δίνει με επέκταση όλες τις επιμέρους συλλογές όπως η List και η Set.

Το Collection interface διαθέτει διάφορες μεθόδους που μπορούν να χρησιμοποιηθούν από όλες τις συλλογές που την υλοποιούν. Σε αυτές συμπεριλαμβάνονται οι ακόλουθες: addObject(), isEmpty(), remove(Object), size(), iterator(), clear() και άλλες. Συμπερασματικά μπορεί να επισημανθεί πως αυτό το interface είναι το θεμέλιο πάνω στο οποίο υλοποιούνται όλες οι κλάσεις των υπολοίπων συλλογών.

7.5.1. Λίστες (Lists)

Το interface List είναι ίσως η πιο δεδομένη επέκταση του Collection και αποτελεί τον τύπο δεδομένων όπου αποθηκεύονται τα αντικείμενα ως διατεταγμένες σειριακές συλλογές. Το List υλοποιείται με διάφορες κλάσεις όπως οι ακόλουθες: ArrayList, Vector, Stack, Queue και άλλες. Η πιο συνήθης περίπτωση είναι η ArralyList που δίνει τη δυνατότητα για χειρισμό δυναμικών πινάκων στην Java. Το μέγεθος της μεταβάλλεται αυτομάτως με την προσθήκη ή την αφαίρεση δεδομένων. Η υλοποίηση ενός παραδείγματος για την ArrayList δίνεται στο τμήμα κώδικα που παρουσιάζεται στην ενότητα 7.5.4 ως παράδειγμα χρήση Collections.

7.5.2. Απεικονίσεις (Maps)

Στην Java, το interface απεικόνιση (Map) είναι μέρος της συλλογής java.util και δίνει τη δυνατότητα της αντιστοίχισης μεταξύ ενός κλειδιού και μίας τιμής. Η απεικόνιση δεν είναι υπο-τύπος του interface collection και συμπεριφέρεται με διαφορετικό τρόπο. Πιο συγκεκριμένα, α) δεν επιτρέπει διπλές τιμές κλειδιών (πρέπει να είναι μοναδικά) ενώ οι τιμές μπορούν να εμφανίζονται πολλές φορές, β) επιτρέπει σε κάποιες υλοποιήσεις της την ύπαρξη ενός null κλειδιού και γ) επιτρέπει τις εναλλακτικές τεχνικές thread-safe για συγχρονισμένη πρόσβαση στα δεδομένα.

Με μαθηματικούς όρους, η απεικόνιση είναι μία συνάρτηση από ένα σύνολο (σύνολο κλειδιών) σε ένα άλλο σύνολο (το σύνολο των τιμών). Η βιβλιοθήκη της Java διαθέτει δύο υλοποιήσεις για την απεικόνιση, την HashMap και την TreeMap. Γενικά είναι προτιμότερη η HashMap εκτός εάν πρέπει η διάσχιση των κλειδιών να γίνεται με διατεταγμένη σειρά. Συνοπτικά, για να βρεθούν όλα τα κλειδιά και οι αντίστοιχες τιμές τους σε μία απεικόνιση γίνεται η διάσχιση των κλειδιών τους με κάποιο επαναληπτικό τρόπο.

Κάποιες από τις μεθόδους του interface Map είναι οι ακόλουθες: put (Object, Object), remove(Object, equals(Object), size() και values(). Ακολουθεί ένα τμήμα κώδικα που δημιουργεί μία υλοποίηση της απεικόνισης.

```
import java.util.*;

public class MapDemo
{
   public static void main(String[] args)
   {
      Map <Integer, String> MP1 = new Map <Integer, String>();

      MP1.put ("Μία", new Integer(100));
      MP1.put ("Ωραία", new Integer(200)););
      MP1.put ("Πεταλούδα" new Integer(500)););
      MP1.put ("Πετάει" new Integer(500)););

      System.out.println("Η τιμή για το 1 είναι"+ HM1.get());

      for(Map.Entry<String, Integer) MapE: MP1.entrySet())
            System.out.println(MapE.getKey()+ ":"+ MapE.getValue());
   }
}
```

7.5.3. Σύνολα (Sets)

Το interface Σύνολο είναι μία διατεταγμένη συλλογή αντικειμένων στην οποία δεν μπορούν να αποθηκευτούν διπλότυπα δεδομένα. Συνεπώς χρησιμοποιείται όταν είναι επιθυμητή η αποθήκευση μοναδικών δεδομένων.

Στα μαθηματικά, το σύνολο απορρίπτει τις διπλότυπες τιμές. Εάν ένα αντικείμενο υπάρχει ήδη στο σύνολο τότε η οποιαδήποτε προσπάθεια να προστεθεί το ίδιο ξανά απλώς αγνοείται χωρίς να υπάρχει μήνυμα λάθους. Αυτό είναι πολύ χρήσιμο σε πολλές προγραμματιστικές περιπτώσεις. Υπάρχουν δύο διαφορετικές δομές που υλοποιούν το σύνολο, το hash table και το δένδρο. Η πρώτη απαιτεί έναν κωδικό και γενικώς είναι η προτιμότερη επιλογή εκτός εάν απαιτείται η διάσχιση των δεδομένων να γίνεται με κάποια διατεταγμένη (ταξινομημένη) σειρά.

Η υλοποίηση του interface Set γίνεται μέσω διαφόρων κλάσεων όπως το HashSet, το TreeSet και άλλες. Ακολουθεί μία υλοποίηση του HashSet που στην πραγματικότητα είναι μία υλοποίηση της δομής hash table. Τα αντικείμενα που αποθηκεύονται, τοποθετούνται σύμφωνα με κάποιο hash κωδικό.

```
import java.util.*;

public class HashDemo
{
   public static void main(String[] args)
   {
      HashMap <Integer, String> HM1 = new HashMap <Integer, String>();

      HM1.put (1, "Μία");
      HM1.put (2, "Ωραία");
      HM1.put (3, "Πεταλούδα");

      System.out.println("Η τιμή για το 1 είναι"+ HM1.get());

      for(Map.Entry<Integer, String) h: HM1.entrySet())
            System.out.println(h.getKey()+ " "+h.getValue());
   }
```

```
}
```

Ακολουθεί μία άλλη κλάση με μία διαφορετική εκδοχή του παραδείγματος.

```
import java.util.*;

public class HashDemo2
{
   public static void main(String[] args)
   {
      HashSet <String> HS2 = new HashSet <String>();

      HS2.put ("Μία");
      HS2.put ("Ωραία");
      HS2.put ("Πεταλούδα");
      HS2.put ("Πετάει");

      Iterator<String> itr = HS2.iterator();

      while(itr.hasNext()
            System.out.println(itr.next);
   }
}
```

7.5.4. Παράδειγμα Χρήσης Collections

Το παράδειγμα που ακολουθεί δείχνει τη χρήση μίας ArrayList η οποία απλώς αποθηκεύει συνεχόμενους περιττούς αριθμούς από το 1 μέχρι το 11 και μετά εμφανίζεται το περιεχόμενο της μετά την αφαίρεση του τρίτου της στοιχείου.

```
import java.io.*;
import java.util.*;

public class ArrayListDemo
{
   public static void main(String[] args)
   {
      ArrayList <Integer> aList = new ArrayList <Integer>();

      for(int i=1; i<=11; i=i+2)
            aList.add(i);

      System.out.println(aList);
      aList.remove(3);
      for(int i=0; i<=aList.size(); i++)
            System.out.println(aList.get(i)+ " ");
   }
}
```

7.6. Χειρισμός Exceptions (Handling Exceptions)

Ο χειρισμός των εξαιρέσεων είναι ένα από τα δύο μέσα της Java για το χειρισμό των runtime λαθών έτσι ώστε να διατηρηθεί η κανονική ροή εκτέλεσης μίας εφαρμογής. Ο μηχανισμός χειρισμού Exceptions βοηθά στην αντιμετώπιση λαθών όπως το ClassNotFoundException, IOException,

RemoteException και άλλα. Τι είναι όμως εξαίρεση; Είναι ένα μη επιθυμητό ή μη αναμενόμενο συμβάν που εμφανίζεται αναπάντεχα κατά την εκτέλεση του προγράμματος και διαταράσσει την ομαλή ροή των εντολών του.

Οι εξαιρέσεις μπορούν να ανιχνευθούν και να αντιμετωπιστούν από το ίδιο το πρόγραμμα. Πιο συγκεκριμένα όταν μία εξαίρεση εμφανιστεί μέσα στο σώμα μίας μεθόδου τότε ένα αντικείμενο δημιουργείται, ένα αντικείμενο εξαίρεσης. Αυτό περιέχει πληροφορία για την εξαίρεση όπως το όνομα και την περιγραφή της καθώς και την κατάσταση του προγράμματος κατά την εμφάνιση της.

Μία λίστα πιθανών αιτιών εμφάνισης εξαιρέσεων είναι η ακόλουθη: λανθασμένη είσοδος δεδομένων από το χρήστη, λάθος σε μία συσκευή, απώλεια δικτυακής σύνδεση, εξάντληση αποθηκευτικού χώρου, προσπάθεια για άνοιγμα ανύπαρκτου αρχείο, προσπάθεια πρόσβασης εκτός των ορίων ενός πίνακα και άλλες.

7.6.1. Η σημασία του Χειρισμού Εξαιρέσεων (Handling Exceptions) (Επανάληψη)

Η σημασία του χειρισμού των εξαιρέσεων έγκειται στο ότι αποτελεί τον πιο ευέλικτο τρόπο για να περνάει ο έλεγχος από το σημείο αναφοράς του λάθους σε ένα χειρισμό αποκατάσταση του. Εδώ πρέπει να σημειωθεί πως υπάρχει μία σημαντική διαφορά μεταξύ του λάθους και της εξαίρεσης. Το λάθος υποδεικνύει ένα σοβαρό πρόβλημα το οποίο μία εφαρμογή δεν προσπαθεί να ανιχνεύσει ενώ η εξαίρεση υποδεικνύει μία κατάσταση που μπορεί να ανιχνευθεί.

Οι εξαιρέσεις μπορούν να διακριθούν σε δύο κατηγορίες: στις εξαιρέσεις που είναι ήδη υλοποιημένες από την τυπική Java και τις εξαιρέσεις που δημιουργούνται από τον προγραμματιστή. Με άλλα λόγια, η πρώτη κατηγορία περιλαμβάνει τις εξαιρέσεις που είναι διαθέσιμες στις βιβλιοθήκες της Java. Μία άλλη διάκριση των εξαιρέσεων ο διαχωρισμός τους σε μη ελεγχόμενες που επεκτείνουν την κλάση RuntimeException ή error και σε ελεγχόμενες που οφείλονται σε εξωτερικές καταστάσεις όπου ο προγραμματιστής ελέγχει αν το πρόγραμμα του χειρίζεται αυτές τις εξαιρέσεις.

7.6.2. Το try-with-resources Μπλοκ

Κάθε εξαίρεση θα πρέπει κάπου να γίνεται διαχειρίσιμη μέσα στο πρόγραμμα. Εάν μία εξαίρεση δεν αντιμετωπιστεί τότε θα εμφανιστεί ένα μήνυμα λάθους καθώς το πρόγραμμα θα τερματίζεται. Αυτό μπορεί να είναι ικανοποιητικό για μία φοιτητική εργασία αλλά δεν αποτελεί αποδεκτή κατάσταση για το επαγγελματικό λογισμικό. Για αυτό απαιτείται η εγκατάσταση ενός χειρισμού των εξαιρέσεων μέσα στον κώδικα. Αυτό επιτυγχάνεται με την εντολή try και catch. Το κάθε τμήμα κώδικα περιέχει μία ή περισσότερες εντολές που μπορεί να πυροδοτήσουν μία εξαίρεση. Κάθε πρόταση (clause) catch περιέχει το χειρισμό για έναν τύπο εξαίρεσης.

7.6.3. Διαχείριση Πολλαπλών Εξαιρέσεων

Στον παρακάτω τμήμα κώδικα παρουσιάζεται η δυνατότητα αντιμετώπισης πολλαπλών εξαιρέσεων με χρήση πολλών μπλοκ της εντολής catch.

```
try
   {
      String fname= …;
```

```
      File InFile = new File(fname);
      Scanner inp = new Scanner(InFile);
      ..
   }
    catch (IOException exception)
    {
     exception.printStachTrace();
    }
    catch (NumberFormatException exception)
    {
     System.out.println(("Η είσοδος δεν είναι αριθμός");
    }
```

Σύμφωνα με τη λογική της εντολής try και catch αν κάποιο exception εμφανιστεί τότε η εκτέλεση του κώδικα μεταφέρεται στον μπλοκ εντολών του αντίστοιχου exception. Ένας καλός τρόπος αντίδρασης σε μία εξαίρεση είναι το να δίνεται στο χρήστη η δυνατότητα για διόρθωση της εισόδου δεδομένων. Είναι σημαντική να τονιστεί ότι ο προγραμματιστής θα πρέπει να τοποθετεί catch μπλοκ στις μεθόδους που μπορεί να αντιμετωπίσουν προβλήματα με εξαιρέσεις. Γενικά, μια καλή πρακτική είναι να γίνεται throw μία εξαίρεση όσο το δυνατό πιο γρήγορα γίνεται, όταν εντοπίζεται το πρόβλημα, ενώ το catch μπλοκ να τοποθετείται όταν μπορεί να αντιμετωπιστεί το πρόβλημα. Επιπλέον, όταν σε ένα σχήμα try και catch τοποθετηθεί και ένα finally μπλοκ τότε εκεί θα εκτελεστούν οι ενέργειες για οποιαδήποτε εξαίρεση δεν έχει ήδη ένα αντίστοιχο catch μπλοκ για να τη χειριστεί.

7.6.4. Δήλωση Εξαιρέσεων (Exceptions) σε μέθοδο

Για να δηλωθεί ότι μία μέθοδος θα τερματιστεί όταν μία ελεγχόμενη εξαίρεση εμφανιστεί στο σώμα της μεθόδου, χρησιμοποιείται η throws clause όπως παρουσιάζεται στο ακόλουθο τμήμα κώδικα.

```
public void read(String filename)
   {
      throws FileNotFoundException, NoSuchElemementExceeds;
   }
```

Εδώ πρέπει να σημειωθεί πως πρέπει να προσδιοριστούν όλες οι ελεγχόμενες εξαιρέσεις που μπορεί να διαδώσει η μέθοδος και ότι μπορεί να δεχτεί και ως όρισμα μη ελεγχόμενες εξαιρέσεις. Φυσικά αυτός ο τρόπος χειρισμού είναι ο πιο ανεπαρκής γιατί απλώς δηλώνεται στο μεταφραστή (compiler) ότι ο προγραμματιστής γνωρίζει για αυτή την εξαίρεση και αφήνει το πρόγραμμα να τερματίσει τη λειτουργία του όταν εμφανιστεί. Προφανώς, φαίνεται αυτός ο τρόπος χειρισμού είναι κάπως ανεύθυνος και πρέπει να γίνεται η αντιμετώπιση της εξαίρεσης πιο αποτελεσματικά.

7.6.5. Διάδοση Εξαιρέσεων (Throwing Exceptions)

Ένας απλός τρόπος με τον οποίο δομείται το αντικείμενο μία εξαίρεσης με ένα σύνηθες μήνυμα λάθος είναι η χρήση της εντολής throw. Αυτός ο τρόπος παρουσιάζεται στο ακόλουθο τμήμα κώδικα.

```
public class CreditAccount
{
   …
```

```
   public void purchase(double amount)
   {
      if(amount>balance)
            throw new IllegalArgumentException("το ποσό ξεπερνά το
υπόλοιπο λογαριασμού");
      balance = balance - amount;
   }
   …
}
```

Όταν μία εξαίρεση γίνεται ενεργή με την throw τότε η εκτέλεση του κώδικα δεν προχωρά στην επόμενη εντολή αλλά συνεχίζει με έναν χειριστή της εξαίρεσης. Πιο συγκεκριμένα, με την εντολή throw new ένα νέο αντικείμενο εξαίρεσης δομείται και στη συνέχεια διαδίδεται ενώ η επόμενη γραμμή κώδικα δεν εκτελείται όταν συμβεί αυτό το γεγονός.

7.6.6. Δημιουργία Νέων Εξαιρέσεων (Exceptions)

Κάποιες φορές απαιτείται η δημιουργία νέων εξαιρέσεων που δεν περιλαμβάνονται στις βιβλιοθήκες της γλώσσας Java. Τότε ο προγραμματιστής μπορεί να δημιουργήσει το νέο Exception με χρήση του συνδυασμού των δεσμευμένων λέξεων throw και new. Πρέπει να επισημανθεί ότι αυτός ο συνδυασμός έχει ήδη χρησιμοποιηθεί στο παράδειγμα της προηγούμενης υπο-ενότητας. Ένα τέτοιο παράδειγμα παρουσιάζεται στο ακόλουθο τμήμα κώδικα.

```
public class WordContainsException extends Exception
{
   public WordContainsException(){}
   public WordContainsException(String message)
         {
          super(message);
         }
}
…
try
{
      if(word.contains((" "))
           {
            throw new WordContainsException();
           }
catch (WordContainsException)
{
      System.out.println(("Η δική μου εξαίρεση!");
}
```

Στο πρώτο τμήμα του κώδικα δημιουργείται η νέα εξαίρεση ως μία επέκταση της γενικής εξαίρεσης και ταυτόχρονα δηλώνονται δύο constructors. Στο δεύτερο τμήμα του κώδικα απλώς γίνεται η χρήση της νέας εξαίρεσης που δημιουργήθηκε.

7.6.7. Η έννοια των Assertions

Μία άλλη επιλογή που προσφέρει η Java για την αντιμετώπιση των λαθών είναι τα Assertions. Σύμφωνα με την Java τεκμηρίωση της Oracle ο ορισμός του Assertion είναι: μία πρόταση που

επιτρέπει στον προγραμματιστή να δοκιμάσει τις παραδοχές για το πρόγραμμα του. Για παράδειγμα ακόλουθη γραμμή κώδικα παρουσιάζει τη δήλωση ενός Assertion.

```
assert something == true;
```

Όταν αυτός ο κώδικας εκτελεστεί και η συνθήκη έχει τιμή False τότε ένα Assertion διαδίδεται. Μία προσπάθεια συνδυασμού ενός Assertion με ένα Exception παρουσιάζεται στο τμήμα κώδικα που ακολουθεί.

```
try
{
   assert x != 0;
   return 1/x;
}
catch (Throwable t)
{
   return null;
}
```

Πρέπει να τονιστεί οι δύο επιλογές, Assertion και Exception, δεν θεωρούνται ισότιμες. Είναι δύο εντελώς διαφορετικά πράγματα.

7.6.8. Παραδείγματα Exceptions

Ακολουθεί ένα ολοκληρωμένο παράδειγμα χειρισμού εξαιρέσεων.

```
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.Scanner;

/** Αυτό το πρόγραμμα διαβάζει ένα αρχείο που περιέχει αριθμούς και
αναλύει το περιεχόμενό του. Εάν το αρχείο δεν υπάρχει ή περιέχει
συμβολοσειρές που δεν είναι αριθμοί τότε εμφανίζεται μήνυμα σφάλματος.*/

public class DataAnalyzer
{
   public static void main(String[] args)
   {
      Scanner in = new Scanner(System.in);
      DataSetReader reader = new DataSetReader();

      boolean done = false;
      while (!done)
      {
         try
         {
            System.out.println("Δώσε το όνομα αρχείου: ");
            String fname = in.next();

            double[] data = reader.readFile(fname);
            double sum = 0;
            for (double d : data) sum = sum + d;
            System.out.println("Το άθροισμα είναι: " + sum);
            done = true;
         }
         catch (FileNotFoundException exception)
         {
```

```
            System.out.println("Ανύπαρκτο αρχείο");
         }
         catch (BadDataException exception)
         {
            System.out.println("Μη ορθά δεδομένα: " +
exception.getMessage());
         }
         catch (IOException exception)
         {
            exception.printStackTrace();
         }
      }
   }
}
```

7.7. Java Logging API

Η καταγραφή (logging) είναι μία διεργασία με την οποία καταγράφονται τα μηνύματα που είναι σχετικά με την εκτέλεση ενός προγράμματος σε ένα κεντρικό σημείο. Αυτή η δυνατότητα επιτρέπει τη δημιουργία μίας αναφοράς μηνυμάτων των λαθών και των προειδοποιήσεων όπως και άλλων πληροφοριών (χρόνος εκτέλεσης και στατιστικά). Όλα αυτά είναι χρήσιμα για τη μετέπειτα ανάλυση.

7.7.1. Χρήση μηχανισμού JAVA Logging

Στην Java ο μηχανισμός καταγραφής υλοποιείται με το αντικείμενο Logger. Πιο συγκεκριμένα, διατίθεται το Java logging API που επιτρέπει τη ρύθμιση των μηνυμάτων που θα εμφανιστούν ενώ και μεμονωμένες κλάσεις μπορούν να χρησιμοποιήσουν το Logger για να εγγράψουν μηνύματα στα αρχεία καταγραφής. Έτσι το java.util.logging παρέχει αυτές τις δυνατότητες μέσω της κλάσης Logger.

Ο παρακάτω κώδικας δημιουργεί ένα αντικείμενο Logger με την παραδοχή πως η τρέχουσα κλάση ονομάζεται TestLogger.

```
import java.util.Logger;

private final static Logger LOGGER=
Logger.getLogger(TestLogger.class.getName());
```

7.7.2. Επίπεδα logging

Η κλάση Logger είναι στην πραγματικότητα μία ιεραρχία Loggers και το σύμβολο της τελείας υποδεικνύει ένα επίπεδο στην ιεραρχία. Για παράδειγμα, εάν λαμβάνεται ένας Logger για το com.example τότε αυτός είναι παιδί του κενού String. Κάθε ρύθμιση ενός γονικού Logger επιδρά και στα παιδιά του. Το επίπεδο όμως του logging καθορίζει και τη σοβαρότητα του μηνύματος. Η κλάση Level δίνει την πληροφορία για τα μηνύματα που θα καταγραφούν. Μία φθίνουσα λίστα (ως προς τη σοβαρότητα) είναι η εξής: SEVER, WARNING, INFO, CONFIG, FINE, FINER και FINEST. Επιπλέον, υπάρχουν και οι τιμές OFF και ALL που αποκλείουν ή συμπεριλαμβάνουν το σύνολο των

επιπέδων. Δίνοντας την εντολή LOGGER.setLevel(Level.CONFIG) σημαίνει πως θα καταγραφούν όλα τα μηνύματα των επιπέδων SEVER, WARNING, INFO και CONFIG.

7.7.3. Παραμετροποίηση Logging

Κάθε Logger έχει πρόσβαση σε διάφορους χειριστές (handlers). Ο χειριστής λαμβάνει ως είσοδο μηνύματα από το Logger και τα προωθεί σε κάποιο προορισμό. Η μέθοδος setLevel(Level.OFF) σταματάει τη λειτουργία του χειριστή. Οι τυπικοί χειριστές περιλαμβάνουν τους ConsoleHandler (για τα μηνύματα που καταγράφονται στην κονσόλα) και FileHandler (για τα μηνύματα που καταγράφονται στο αρχείο).

Η παραμετροποίηση ενός χειριστή μπορεί να γίνει με τη χρήση ενός μορφοποιητή (formatter). Οι διαθέσιμοι μορφοποιητές είναι ο SimpleFormatter και ο XMLFormatter αλλά ο προγραμματιστής μπορεί να δημιουργήσει και τους δικούς του.

7.7.4. Καλές πρακτικές logging

Στις καλές πρακτικές συμπεριλαμβάνεται η κοινή πρακτική της χρήσης ονόματος του Logger για κάθε κλάση της οποίας οι ενέργειες καταγράφονται ως μία κατηγορία μηνυμάτων καθώς επιτρέπει στον προγραμματιστή τη διακριτή ρύθμιση της καταγραφής της κάθε κλάσης. Προφανώς αυτή η πρακτική είναι προτεινόμενη από την τεκμηρίωση του Logging API.

7.7.5. Παράδειγμα logging

Ακολουθεί ένα παράδειγμα κώδικα για τη διαχείριση ενός χειριστή Logger. Το πρώτο μέρος δημιουργεί το χειριστή MyHandler.

```
import java.util.LogRecord;
import java.util.StreamHandler;

public class MyHandler extends StreamHandler
{
   public void publish(LogRecord record)
        {
            super.publish(record);
        }
   public void flush()
        {
            super.flush();
        }
   public void close()
        {
            super.close();
        }
}
```

Στη συνέχεια παρατίθεται το τμήμα κώδικα που καθορίζει τη μορφοποίηση του χειριστή.

```
import java.util.LogRecord;
import java.util.Formatter;

public class MyFormatter extends Formatter
```

```
{
   public String format(LogRecord record)
        {
            return record.getThreadID()+"::"+record.getSourceClassName()()
+"::"+record.getMessage()+"\n";
        }
}
```

7.8. Πρακτική εξάσκηση

Άσκηση 1: Δημιουργία Interface

Δημιουργήστε ένα interface με το όνομα MyInterface το οποίο θα περιέχει τρεις μεθόδους: α) μία για την προσθήκη ενός αντικειμένου, β) μία την αφαίρεση ενός αντικείμενου και γ) μία που θα επιστρέφει true αν το αντικείμενο υπάρχει αλλιώς false. Τα αντικείμενα θεωρείστε ότι είναι String.

Άσκηση 2: Δημιουργία Generic κλάσης

Δημιουργήστε μία generic κλάση με όνομα Box<T>, όπου T θα είναι ο τύπος των αντικειμένων που αποθηκεύει. Η κλάση θα πρέπει να έχει μία μέθοδο για την αποθήκευση στοιχείων και μία για την πρόσβαση στοιχείων ενώ θα υπάρχει και μία μέθοδος που θα ελέγχει αν είναι το αντικείμενο της κλάσης είναι κενό.

Άσκηση 3: Χειρισμός Συνόλου (Set)

Δημιουργήστε μία κλάση που θα υλοποιεί το interface της πρώτης άσκησης. Θεωρείστε ότι τα στοιχεία που θα αποθηκεύονται είναι στοιχεία ενός συνόλου (HashSet). Επιπλέον θα εμφανίζονται τα μηνύματα επιτυχούς ή ανεπιτυχούς προσθήκης ή διαγραφής.

Άσκηση 4: Δημιουργία Exception

Δημιουργήστε μία νέα exception για την περίπτωση που γίνεται προσπάθεια διαίρεσης με το 0. Φροντίστε να εμφανίζονται τα κατάλληλα μηνύματα. Στη συνέχεια γράψτε μία κλάση με την οποία θα γίνεται έλεγχος της νέας εξαίρεση δίνοντας μία αποδεκτή διαίρεση και μία μη αποδεκτή.

Άσκηση 5: Δημιουργία Logger

Δημιουργείστε ένα Logger για τον προηγούμενο κώδικα.

Ενδεικτικές Λύσεις

Άσκηση 1: Δημιουργία Interface

```
public interface MyInterface {
  void addItem(String item);

    void removeItem(String item);

    boolean containsItem(String item);
}
```

Άσκηση 2: Δημιουργία Generic κλάσης

```
public class Box<T> {
    private T item;
    public void setItem(T item) {
        this.item = item;
    }

    public T getItem() {
        return item;
    }

    public boolean isEmpty() {
        return item == null;
    }
}
```

Άσκηση 3: Χειρισμός Συνόλου

```
import java.util.HashSet;
import java.util.Set;

public class MyInterfaceImpl implements MyInterface {
    private Set<String> items;

    public MyInterfaceImpl() {
        this.items = new HashSet<>();
    }

    public void addItem(String item) {
        if (items.add(item)) {
            System.out.println("Το αντικείμενο \"" + item + "\"
προστέθηκε επιτυχώς.");
        } else {
            System.out.println("Το αντικείμενο \"" + item + "\" υπάρχει
ήδη.");
        }
    }
    public void removeItem(String item) {
        if (items.remove(item)) {
            System.out.println("Το αντικείμενο \"" + item + "\"
αφαιρέθηκε επιτυχώς.");
        } else {
            System.out.println("Το αντικείμενο \"" + item + "\" δεν
βρέθηκε στη συλλογή.");
```

```
        }
}
```

Άσκηση 4: Δημιουργία Exception

```
public class DivisionByZeroException extends Exception {
    public DivisionByZeroException() {
        super("Δεν επιτρέπεται η διαίρεση με το μηδέν.");
    }

    public DivisionByZeroException(String message) {
        super(message);
    }
}


public class DivisionTest {
    public static double divide(int numerator, int denominator) throws
DivisionByZeroException {
        if (denominator == 0) {
            throw new DivisionByZeroException();
        }
        return (double) numerator / denominator;
    }

    public static void main(String[] args) {
        try {
            System.out.println("5 / 2 = " + divide(5, 2));
            System.out.println("5 / 0 = " + divide(5, 0));
        } catch (DivisionByZeroException e) {
            System.out.println("Σφάλμα: " + e.getMessage());
        }
    }
}
```

Άσκηση 5: Δημιουργία Logger

```
import java.util.logging.*;

public class DivisionTest {
    private       static       final       Logger       logger       =
Logger.getLogger(DivisionTest.class.getName());

    public static double divide(int numerator, int denominator) throws
DivisionByZeroException {
        if (denominator == 0) {
            logger.warning("Προσπάθεια διαίρεσης με το μηδέν!");
            throw new DivisionByZeroException("Ο παρονομαστής δεν μπορεί
να είναι μηδέν.");
        }
        double result = (double) numerator;
        logger.info("Διαίρεση: " + numerator + " / " + denominator + " =
" + result);
        return result;
    }

    public static void main(String[] args) {
        try {
```

```
            System.out.println("5 / 2 = " + divide(5, 2));

            System.out.println("5 / 0 = " + divide(5, 0));
        } catch (DivisionByZeroException e) {
            System.out.println("Σφάλμα: " + e.getMessage());
            logger.severe("Σφάλμα: " + e.getMessage());
        }
    }
}
```

7.9. Ερωτήσεις αυτο-αξιολόγησης

1. **Ποιο από τα παρακάτω είναι σωστό σχετικά με τα Interfaces στην Java;**
   α) Μπορούν να έχουν κατασκευαστές
   β) Όλες οι μέθοδοι τους είναι public και abstract
   γ) Μπορούν να περιέχουν private static μεθόδους
   δ) Μόνο οι public μέθοδοι μπορούν να είναι abstract
2. **Τι συμβαίνει αν μια κλάση υλοποιεί ένα Interface αλλά δεν υλοποιεί όλες τις μεθόδους του;**
   α) Η κλάση πρέπει να δηλωθεί ως abstract
   β) Η κλάση μπορεί να παραμείνει concrete
   γ) Ο μεταγλωττιστής αγνοεί τις μη υλοποιημένες μεθόδους
   δ) Εμφανίζεται σφάλμα χρόνου εκτέλεσης
3. **Ποιο είναι το κύριο πλεονέκτημα της χρήσης Generics;**
   α) Υποστηρίζουν μόνο τύπους String
   β) Παρέχουν έλεγχο τύπου κατά τη μεταγλώττιση
   γ) Μειώνουν το μέγεθος του κώδικα
   δ) Δεν έχουν πλεονεκτήματα σε σχέση με τις κλασικές δομές
4. **Ποια από τις παρακάτω είναι έγκυρη δήλωση γενικευμένης κλάσης;**
   α) class MyClass<T> {}
   β) class MyClass<T extends Number> {}
   γ) class MyClass<T super String> {}
   δ) Και οι δύο πρώτες
5. **Ποια από τις παρακάτω ετικέτες χρησιμοποιείται για την περιγραφή των παραμέτρων μιας μεθόδου στο Javadoc;**
   α) @param
   β) @parameter
   γ) @arg
   δ) @argument
6. **Ποια κλάση χρησιμοποιείται για τη δημιουργία μιας μη συγχρονισμένης λίστας;**
   α) Vector
   β) LinkedList
   γ) ArrayList
   δ) ConcurrentLinkedList

7. **Ποια είναι η βασική κλάση όλων των εξαιρέσεων στην Java;**
   α) Error
   β) Exception
   γ) Throwable
   δ) RuntimeException
8. **Ποια από τις παρακάτω δηλώσεις χρησιμοποιείται για την διάδοση μιας εξαίρεσης;**
   α) throw
   β) throws
   γ) catch
   δ) try
9. **Ποιος είναι ο βασικός σκοπός των assertions στην Java;**
   α) Διαχείριση εξαιρέσεων
   β) Έλεγχος συνθηκών κατά την εκτέλεση
   γ) Εμφάνιση μηνυμάτων στη κονσόλα
   δ) Παροχή σχολίων
10. **Ποιο επίπεδο καταγραφής (logging) χρησιμοποιείται για πληροφορίες ανάπτυξης;**
    α) SEVERE
    β) INFO
    γ) DEBUG
    δ) FINE
11. **Ποια μέθοδος μπορεί να περιλαμβάνει προεπιλεγμένη υλοποίηση σε ένα Interface από την Java 8 και μετά;**
    α) Static
    β) Default
    γ) Abstract
    δ) Public
12. **Ποια από τις παρακάτω δηλώσεις για Generics είναι λανθασμένη;**
    α) Η γενικευμένη κλάση επιτρέπει τον έλεγχο τύπου κατά τη μεταγλώττιση
    β) Τα Generics μπορούν να χρησιμοποιηθούν με Collections
    γ) Τα Generics δεν υποστηρίζουν τύπους Wrapper
    δ) Μπορούμε να περιορίσουμε το τύπο των Generics με extends
13. **Ποια από τις παρακάτω κλάσεις δεν ανήκει στο Collections Framework;**
    α) HashSet
    β) ArrayList
    γ) HashMap
    δ) Array
14. **Ποια μέθοδος χρησιμοποιείται για την αφαίρεση ενός στοιχείου από ένα List;**
    α) remove()
    β) delete()
    γ) discard()
    δ) exclude()
15. **Ποια μέθοδος μπορεί να χρησιμοποιηθεί για την πρόσβαση στο μήνυμα μιας εξαίρεσης;**

α) getErrorMessage()

β) getMessage()

γ) printStackTrace()

δ) getDescription()

16. Ποια είναι η βασική διαφορά μεταξύ exceptions και assertions;

α) Οι exceptions είναι για debugging, οι assertions για runtime errors

β) Οι exceptions είναι για runtime errors, οι assertions για debugging

γ) Και οι δύο χρησιμοποιούνται για runtime errors

δ) Οι exceptions δεν υποστηρίζονται από τη Java

17. Ποια κλάση χρησιμοποιείται για την παραμετροποίηση των logs;

α) LogManager

β) LogConfigurator

γ) LogHandler

δ) LogWriter

18. Ποια είναι η κύρια διαφορά μεταξύ Interface και Abstract Class στη Java;

α) Ένα Interface μπορεί να έχει constructor, ενώ μια Abstract Class όχι

β) Ένα Interface μπορεί να περιέχει σταθερές μόνο, ενώ μια Abstract Class μπορεί να έχει πεδία

γ) Ένα Interface μπορεί να έχει πολλαπλή κληρονομικότητα, ενώ μια Abstract Class όχι

δ) Μια Abstract Class μπορεί να περιέχει μόνο static μεθόδους

19. Ποια είναι η προεπιλεγμένη σημασία του T στις γενικευμένες κλάσεις;

α) Τύπος χαρακτήρων

β) Τύπος αριθμών

γ) Τύπος αντικειμένων

δ) Συγκεκριμένος τύπος που καθορίζεται κατά την υλοποίηση

20. Ποια κλάση από το Collections Framework είναι fail-fast;

α) HashMap

β) ArrayList

γ) LinkedList

δ) Όλες οι παραπάνω

21. Ποια μέθοδος χρησιμοποιείται για την ταξινόμηση μιας λίστας;

α) arrange()

β) sort()

γ) order()

δ) reorder()

22. Ποιο είναι το αποτέλεσμα της χρήσης throw χωρίς try ή catch;

α) Το πρόγραμμα μεταγλωττίζεται αλλά εμφανίζει σφάλμα κατά την εκτέλεση

β) Εμφανίζεται σφάλμα κατά τη μεταγλώττιση

γ) Το throw αγνοείται

δ) Το πρόγραμμα τερματίζεται χωρίς μήνυμα σφάλματος

7.10. Απαντήσεις στις ερωτήσεις αυτο-αξιολόγησης

1. β	2. α	3. β	4. δ	5. α
6. γ	7. γ	8. β	9. β	10. δ
11. β	12. γ	13. δ	14. α	15. β
16. β	17. α	18. γ	19. δ	20. δ
21. β	22. β			

ΚΕΦΑΛΑΙΟ 8: Java IO API και Serialization

8.1. Εκπαιδευτικοί στόχοι του παρόντος κεφαλαίου

Οι εκπαιδευτικοί στόχοι του παρόντος κεφαλαίου, συνοψίζονται στα κάτωθι σημεία. Οι εκπαιδευόμενοι:

- θα μάθουν να αναγνωρίζουν τύπους ροών εισόδου/εξόδου (I/O Streams),
- θα μάθουν να διακρίνουν τις δυαδικές ροές (Byte Streams) από τις ροές χαρακτήρων (Character Streams),
- θα μάθουν να περιγράφουν τις λειτουργίες της κλάσης System,
- θα μάθουν να περιγράφουν τη λειτουργικότητα της κλάσης File,
- θα κατανοήσουν τις βασικές αρχές του Serialization στην Java,
- θα μάθουν να εξηγούν τη χρήση της τυπικής εισόδου και εξόδου (STDIN, STDOUT, STDERR),
- θα μάθουν να χρησιμοποιούν το Java IO API για διαχείριση αρχείων,
- θα μάθουν να αξιοποιούν το πακέτο java.io για λειτουργίες εισόδου/εξόδου,
- θα μάθουν να χρησιμοποιούν την κλάση Console για είσοδο/έξοδο,
- θα μάθουν να εφαρμόζουν τη μετατροπή από δυαδικές ροές σε ροές χαρακτήρων (InputStreamReader),
- θα μάθουν να εφαρμόζουν τη μετατροπή από ροές χαρακτήρων σε δυαδικές ροές (OutputStreamWriter),
- θα μάθουν να διαχειρίζονται το FileSystem, τους καταλόγους (Paths) και τα αρχεία (Files) με χρήση του πακέτου java.nio και nio.2,
- θα μάθουν να υλοποιούν ροές εισόδου και εξόδου (Input/Output Streams).

8.2. Java I/O API – Εισαγωγή

Το java.io API είναι σημαντικό εργαλείο για την ανάπτυξη εφαρμογών Java με σύνθετες ανάγκες I/O που απαιτούν αλληλεπίδραση με εξωτερικές πηγές ή αποδέκτες δεδομένων. Το API αυτό, επιτρέπει τη διαχείριση εισόδου και εξόδου δεδομένων από και προς αρχεία, κονσόλα, δίκτυα, κ.λπ. καθώς στηρίζει τη λειτουργία του στην αφαίρεση (abstraction) των ρευμάτων (ή ροών) δεδομένων (data streams).

Τα ρεύματα δεδομένων (streams) είναι ένας τρόπος για να μετακινούνται πληροφορίες μέσα σε μια εφαρμογή, από μια πηγή προς έναν προορισμό, σαν ένα "ποτάμι" δεδομένων, ανεξάρτητα από το είδος της πληροφορίας (π.χ. κείμενο, εικόνα, ήχος). Μπορούμε να τα σκεφτούμε ως κανάλια που επιτρέπουν τη συνεχή ροή δεδομένων, είτε πρόκειται για ανάγνωση (είσοδο) είτε για εγγραφή (έξοδο). Για παράδειγμα, ένα ρεύμα δεδομένων μπορεί να μεταφέρει πληροφορίες από ένα αρχείο δίσκου σε μια εφαρμογή, ή από την εφαρμογή σε έναν εκτυπωτή. Τα ρεύματα κάνουν δυνατή αυτή

τη διαδικασία με πολύ αποδοτικό τρόπο, αφού τα δεδομένα "κυλούν" σταδιακά χωρίς να χρειάζεται να φορτώνονται όλα μαζί στη μνήμη.

Οι βασικές κλάσεις και διεπαφές του API περιέχονται στο πακέτο java.io, του οποίου τα κυριότερα χαρακτηριστικά περιλαμβάνουν:

1. **Stream-based I/O**: Το java.io είναι βασισμένο στην αφαίρεση των ρευμάτων δεδομένων (streams). Τα streams είναι σειριακά ρεύματα δεδομένων (συνήθως ροές bytes ή χαρακτήρων), μέσω των οποίων οι εφαρμογές διαβάζουν και γράφουν δεδομένα.
    - Input Streams: Για ανάγνωση δεδομένων (π.χ. κλάση `FileInputStream` για ανάγνωση από αρχεία).
    - Output Streams: Για εγγραφή δεδομένων (π.χ. κλάση `FileOutputStream` για εγγραφή σε αρχεία).
2. **Character και Byte Streams**: Το java.io API υποστηρίζει τόσο byte streams (για δυαδικά δεδομένα) όσο και character streams (για χαρακτήρες).
    - Byte Streams: Οι κλάσεις `InputStream` και `OutputStream` είναι οι βασικές κλάσεις για διαχείριση byte streams.
    - Character Streams: Οι κλάσεις `Reader` και `Writer` είναι σχεδιασμένες για διαχείριση δεδομένων χαρακτήρων, διευκολύνοντας την ανάγνωση και εγγραφή κειμένου.
3. **Object Serialization**: Η σειριοποίηση στην Java είναι η αναπαράσταση της κατάστασης ενός αντικειμένου ως ροή byte, η οποία περιέχει όλες τις πληροφορίες γι' αυτό. Η σειριοποίηση στη Java βοηθά στη μετακίνηση του κώδικα από μία JVM σε άλλη και στη συνέχεια την αποσειριοποίηση του εκεί. Η κλάση ObjectOutputStream επιτρέπει την εγγραφή ενός αντικειμένου σε ροή δεδομένων, ενώ η ObjectInputStream την ανάκτησή του.
4. **File I/O**: Παρέχεται με την κλάση File, η οποία επιτρέπει την αναπαράσταση και διαχείριση αρχείων και καταλόγων στο σύστημα αρχείων. Μπορεί να χρησιμοποιηθεί για την ανάκτηση πληροφοριών για τα αρχεία (π.χ. μέγεθος, όνομα, μονοπάτι).

Συνοψίζοντας, οι κυριότερες Κλάσεις και Διασυνδέσεις του πακέτου java.io περιλαμβάνουν:

- `InputStream` / `OutputStream`: Βασικές κλάσεις για byte streams.
- `Reader` / `Writer`: Βασικές κλάσεις για character streams.
- `File`: Χρησιμοποιείται για την αναπαράσταση αρχείων και φακέλων στο σύστημα αρχείων.
- `Serializable`: Μια διασύνδεση που δηλώνει ότι ένα αντικείμενο μπορεί να σειριοποιηθεί.
- `BufferedReader` / `BufferedWriter`: Παρέχουν ενδιάμεση μνήμη για βελτίωση της απόδοσης κατά την ανάγνωση/εγγραφή δεδομένων.

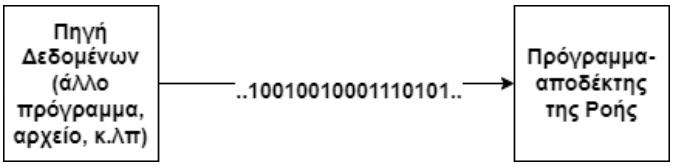
8.3. Διαχείριση Ρευμάτων Δεδομένων (Streams)

Στα επόμενα θα διευκρινίσουμε τις σχετικές έννοιες που προαναφέρθηκαν και θα δούμε πώς μπορούμε να χρησιμοποιούμε τις κλάσεις του πακέτου java.io για να διαχειριζόμαστε περιπτώσεις χρήσης όπως αυτές που αναφέραμε παραπάνω.

8.3.1. Ρεύματα και τύποι ρευμάτων

Η ανάγκη εισόδου ή/και εξόδου δεδομένων υπάρχει στα περισσότερα προγράμματα. Όμως, ένας υπολογιστής μπορεί να συνδέεται με πολλών, διαφορετικών ειδών συσκευές εισόδου ή/και εξόδου. Για παράδειγμα, μπορεί να πρόκειται για την τυπική είσοδο/έξοδο που είναι συνδυασμός πληκτρολογίου/οθόνης, όμως είσοδο και έξοδο μπορούμε να έχουμε από/σε έναν τοπικό δίσκο ή και μέσω δικτύου. Όπως καταλαβαίνουμε, είναι πολύ δύσκολο αν όχι αδύνατο να φτιάξουμε διαφορετικά εργαλεία που να αντιμετωπίζουν ξεχωριστά κάθε δυνατή περίπτωση. Η κεντρική λύση εδώ είναι η προσέγγιση μέσα από την αφαίρεση που επιτυγχάνεται με τις ροές ή ρεύματα (streams).

Με βάση αυτή τη λογική, ένα ρεύμα συσχετίζεται με μια είσοδο ή μια έξοδο δεδομένων, ασχέτως των λεπτομερειών που την χαρακτηρίζουν, π.χ. άλλο πρόγραμμα, αρχείο, εξυπηρετητής κ.λπ. Το πρόγραμμα μας διαβάζει (λαμβάνει) από το ρεύμα οπότε μιλάμε για *Ρεύμα Εισόδου* (input stream) ή γράφει (στέλνει) δεδομένα σε ένα ρεύμα, οπότε μιλάμε για *Ρεύμα Εξόδου* (output stream). Δείτε μια σχηματική απεικόνιση ρεύματος εισόδου στην Εικόνα 16.



Εικόνα 16 - ρεύμα δεδομένων εισόδου

Οι κλάσεις των ρευμάτων επεξεργάζονται δεδομένα σε σειριακή μορφή, ένα byte ή χαρακτήρα τη φορά, πράγμα που μειώνει τις απαιτήσεις μνήμης και επιτρέπει την σταδιακή επεξεργασία μεγάλων όγκων δεδομένων. Από την άλλη, η επεξεργασία ενός byte/χαρακτήρα τη φορά έχει ως αποτέλεσμα να καθυστερεί ο κώδικας, πράγμα που, όπως θα δούμε παρακάτω, μπορούμε να το αντιμετωπίσουμε με χρήση κλάσεων του java.io που προσφέρουν buffering (ενδιάμεση αποθήκευση των δεδομένων και λιγότερες προσβάσεις στο μέσο (π.χ. δίσκο).

Σε μια γενική προσέγγιση, τα βασικά είδη ρευμάτων δεδομένων για την Java είναι δύο: τα byte streams και τα character streams:

- Τα δυαδικά ρεύματα (byte streams) διαχειρίζεται τα δεδομένα σε επίπεδο byte, επομένως 8 bit κάθε φορά, ενώ,
- Τα ρεύματα χαρακτήρων (character streams) σε επίπεδο χαρακτήρα που, για τον Unicode, είναι τυπικά τα 16 bits.

Συνήθως χρησιμοποιούμε τις ροές byte όταν πρόκειται να επεξεργαστούμε δυαδικά αρχεία όπως εικόνες, ήχο, βίντεο, κ.λπ. και τις ροές character για επεξεργασία αρχείων κειμένου.

Το πακέτο java.io περιέχει κλάσεις τόσο για είσοδο και έξοδο δυαδικών ρευμάτων όσο και για είσοδο και έξοδο ρευμάτων χαρακτήρων. Για το σκοπό αυτό περιλαμβάνει (μεταξύ άλλων, όπως θα δούμε αργότερα) τις παρακάτω αφηρημένες (abstract) γονικές κλάσεις (πού όλες επεκτείνουν την *java.lang.Object*):

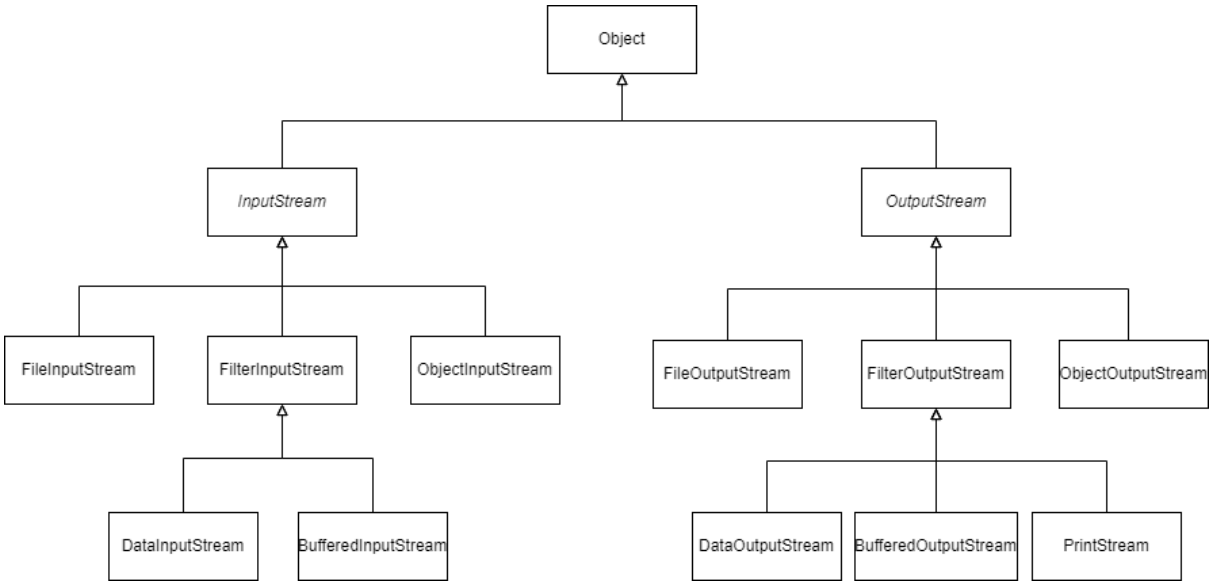
- `InputStream` για είσοδο ρευμάτων byte
- `OutputStream` για έξοδο ρευμάτων byte
- `Reader` για είσοδο ρευμάτων character
- `Writer` για έξοδο ρευμάτων character

Όπως γνωρίζουμε, δεν μπορούμε άμεσα να χρησιμοποιήσουμε μια αφηρημένη κλάση χωρίς πρώτα να υλοποιήσουμε τα αφηρημένα μέλη της. Έτσι, οι παραπάνω κλάσεις δεν αρκούν για να μπορέσουμε να διαχειριστούμε ροή εισόδου ή εξόδου. Όμως, το πακέτο περιλαμβάνει υποκλάσεις τους οι οποίες είναι πλήρως υλοποιημένες (concrete).

8.3.2. Δυαδικά Ρεύματα (Byte Streams)

Τα Δυαδικά Ρεύματα (Byte Streams) στη Java χρησιμοποιούνται για διαχείριση δεδομένων που δεν είναι σε μορφή κειμένου όπως, για παράδειγμα, εικόνες, ήχοι ή άλλου τύπου δυαδικά (binary) αρχεία. Διαχειρίζονται δεδομένα με τη μορφή bytes (8-bit), επιτρέποντας την αποστολή και λήψη τους byte προς byte. Επειδή διαβάζουν και γράφουν δεδομένα σε δυαδική μορφή, μπορούν να χειριστούν αρχεία **κάθε μορφής,** πράγμα που τα κάνει ιδανικά για την ανάγνωση και εγγραφή δεδομένων από οποιαδήποτε πηγή που δεν είναι συμβολοσειρά, όπως εικόνες ή άλλα δυαδικά αρχεία.

Στην Εικόνα 17 βλέπουμε μια απλοποιημένη ιεραρχία κλάσεων που αφορά στη διαχείριση δυαδικών ρευμάτων.



Εικόνα 17 - ιεραρχία κλάσεων δυαδικών ρευμάτων (απλοποιημένη)

Η ιεραρχία `InputStream` χρησιμοποιείται για ανάγνωση (δυαδικών) δεδομένων από πηγές εισόδου ενώ η ιεραρχία `OutputStream` για εγγραφή (δυαδικών) δεδομένων σε προορισμούς εξόδου.

Και οι δύο αυτές κλάσεις έχουν διάφορες εξειδικευμένες υλοποιήσεις (υποκλάσεις), οι οποίες καλύπτουν διαφορετικές ανάγκες. Ενδεικτικά, μπορούμε να δούμε σε αντιστοιχία (Πίνακας 1) μερικές από τις κυριότερες, για είσοδο και έξοδο.

Πίνακας 1 – μερικές από τις κυριότερες κλάσεις byte stream

Κλάσεις Εισόδου (*InputStream*)	**Κλάσεις Εξόδου (*OutputStream*)**
`FileInputStream` Διαβάζει δεδομένα από αρχεία. Είναι η κύρια κλάση για την ανάγνωση byte από ένα αρχείο.	`FileOutputStream` Χρησιμοποιείται για εγγραφή byte σε αρχεία.
`BufferedInputStream` Βελτιώνει την αποδοτικότητα προσθέτοντας προσωρινή αποθήκευση (buffering), *ώστε* να μειώνεται ο αριθμός των προσβάσεων στον δίσκο.	`BufferedOutputStream` Παρέχει buffering κατά την *εγγραφή*, βελτιώνοντας την απόδοση.
`DataInputStream` Χρησιμοποιείται για την ανάγνωση πρωτογενών τύπων δεδομένων, όπως int, float, double κ.λπ., σε δυαδική μορφή.	`DataOutputStream` Επιτρέπει την εγγραφή πρωτογενών τύπων δεδομένων σε δυαδική μορφή, διατηρώντας τη μορφή τους.
	`PrintStream` Για εγγραφή δεδομένων κυρίως σε κονσόλα (αλλά και όχι μόνο). Υποστηρίζει μορφοποίηση διαφόρων τύπων δεδομένων, κωδικοποίηση, χειρίζεται bytes. Η βασική της χρήση είναι για έξοδο σε κονσόλα ή για logging, ενώ για εγγραφή κειμένου γενικά, συνήθως προτιμάται η `PrintWriter` που θα δούμε παρακάτω.

Θα ξεκινήσουμε με ένα παράδειγμα ανάγνωσης και εγγραφής σε δυαδικό αρχείο.

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
public class Stream1 {
      public static void main(String [] args) {
            try        (FileInputStream        fileIn        =        new
FileInputStream("source.jpg");
```

```
                FileOutputStream           fileOut           =           new
FileOutputStream("duplicate.jpg")){
                   int byteData;
                   while ((byteData = fileIn.read()) != -1) {
                         fileOut.write(byteData);
                   }
             } catch (IOException e) {
                   e.printStackTrace();
               }
      }
}
```

Μετά τα κατάλληλα imports, χρησιμοποιούμε μια εντολή try-with-resources για να ανοίξουμε ένα ρεύμα εισόδου προς ένα δυαδικό αρχείο με όνομα source.jpg που περιέχει μια εικόνα και ένα ρεύμα εξόδου, που θα δημιουργήσει ένα νέο αρχείο με όνομα duplicate.jpg. Η try-with-resources μας εξασφαλίζει ότι μετά την εκτέλεση του τμήματος κώδικα οι πόροι (τα αρχεία) θα κλείσουν. Αν πάτε στην τεκμηρίωση του java API για τις κλάσεις `FileInputStream` και *`FileOutputStream`*, θα δείτε ότι υλοποιούν τη διασύνδεση (interface) *`AutoCloseable`*. Οι δύο αυτές κλάσεις έχουν κατασκευαστές που δέχονται ως όρισμα ένα αλφαριθμητικό με το μονοπάτι του αρχείου στο σύστημά μας. Στη συνέχεια, όσο το ρεύμα εισόδου έχει δεδομένα τα διαβάζουμε και τα γράφουμε στο ρεύμα εξόδου. Σε περίπτωση εξαίρεσης το πρόγραμμα εμφανίζει τη στοίβα των κλήσεων. Η μέθοδος *`read()`* της *`FileInputStream`* διαβάζει το επόμενο byte από τη ροή εισόδου και σε περίπτωση που φθάσει στο τέλος του αρχείου, επιστρέφει -1. Αντίστοιχα, η μέθοδος *`write()`* της *`FileOutputStream`* γράφει στο ρεύμα εξόδου το byte που δέχεται ως όρισμα (καθώς ένας ακέραιος της Java είναι 4 bytes, στην πράξη η write γράφει τα τελευταία οκτώ bits του ακέραιου που δέχεται ως όρισμα).

Αν εκτελέσετε τον παραπάνω κώδικα με είσοδο π.χ. ένα αρχείο μεγέθους 4 MB θα δείτε ότι παίρνει αρκετά δευτερόλεπτα χρόνο μέχρι να ολοκληρωθεί. Ο λόγος είναι ότι η διαδικασία γίνεται ένα byte κάθε φορά. Ας ξαναγράψουμε τον κώδικα, αυτή τη φορά χρησιμοποιώντας τις κλάσεις *`BufferedInputStream`* και *`BufferedOutputStream`*. Οι κατασκευαστές τους δέχονται ως όρισμα ένα αντικείμενο τύπου *`InputStream`* ή *`OutputStream`* αντίστοιχα (άρα και *`FileInputStream`*, *`FileOutputStream`* αντίστοιχα, λόγω της ιεραρχίας - Εικόνα 17). Λόγω της αντικειμενοστρέφειας που υποστηρίζει η Java, μπορούμε να χρησιμοποιήσουμε τη λειτουργία της ενθυλάκωσης (encapsulation) που ονομάζεται wrapping με την οποία σε ένα τύπο ρεύματος μπορούν να προστεθούν πρόσθετες λειτουργίες.

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.IOException;
public class Stream2 {
      public static void main(String [] args) {
            try (BufferedInputStream fileIn = new BufferedInputStream(new
FileInputStream("source.jpg"));
               BufferedOutputStream fileOut = new BufferedOutputStream(new
FileOutputStream("duplicate.jpg"))){
                   int byteData;
                   while ((byteData = fileIn.read()) != -1) {
                         fileOut.write(byteData);
```

```
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
}
```

Η μόνη διαφορά στον κώδικα είναι η χρήση των κλάσεων μέσα στο try. Αν εκτελέσετε τον κώδικα, θα παράξει το ίδιο αποτέλεσμα (δημιουργία αντιγράφου του δυαδικού αρχείου εικόνας), όμως η διαφορά στην ταχύτητα είναι πολύ μεγάλη, ο κώδικας ολοκληρώνεται σχεδόν άμεσα (για το ίδιο αρχείο μεγέθους 4 MB). Ο λόγος είναι ότι η χρήση των `BufferedInputStream` και `BufferedOutputStream` βελτιώνει την απόδοση των byte streams, ειδικά όταν διαχειριζόμαστε μεγάλα αρχεία, καθώς οι κλάσεις χρησιμοποιούν buffer και προσθήκη buffer σημαίνει ότι τα ρεύματα θα διαβάσουν/γράψουν δεδομένα σε μεγάλες "δόσεις", μειώνοντας τις συχνές προσβάσεις στον δίσκο.

Οι κλάσεις `DataInputStream` και `DataOutputStream` χρησιμοποιούνται για ανάγνωση και εγγραφή πρωτογενών (primitive) τύπων δεδομένων (π.χ., `int`, `double`, `boolean`) με τρόπο που είναι ανεξάρτητος του συστήματος που τρέχει ο κώδικας. Αυτές οι κλάσεις επιτρέπουν την απευθείας εγγραφή πρωτογενών τιμών σε δυαδική μορφή, χωρίς μετατροπή σε αλφαριθμητικό. Ένα παράδειγμα χρήσης τους βλέπουμε στον κώδικα που ακολουθεί.

```
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;

public class Stream3 {
    public static void main(String[] args) {
        try (DataOutputStream fileOut = new DataOutputStream(new
FileOutputStream("data.dat"))) {
            fileOut.writeInt(34);
            fileOut.writeDouble(3.14);
            fileOut.writeBoolean(true);
        } catch (IOException e) {
            e.printStackTrace();
        }

        try (DataInputStream fileIn = new DataInputStream(new
FileInputStream("data.dat"))) {
            System.out.println(fileIn.readInt());
            System.out.println(fileIn.readDouble());
            System.out.println(fileIn.readBoolean());
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Στο πρώτο μέρος του κώδικα, ανοίγουμε το ρεύμα σε ένα αρχείο με όνομα data.dat και χρησιμοποιούμε τις μεθόδους `writeInt()`, `writeDouble()` και `writeBoolean()` της κλάσης `DataOutputStream` για να γράψουμε μια τιμή τύπου `int`, μια `double` και μια τύπου `boolean`, αντίστοιχα.

Στο δεύτερο μέρος ανοίγουμε το αρχείο που δημιουργήσαμε (θυμηθείτε, τα αρχεία κλείνουν χωρίς δική μας παρέμβαση, λόγω της try-with-resources) και με χρήση των μεθόδων `readInt()`, `readDouble()` και `readBoolean()` διαβάζουμε τις αντίστοιχες τιμές και τις εμφανίζουμε στην οθόνη.

Το αρχείο data.dat που δημιουργείται είναι δυαδικό και όχι αρχείο κειμένου. Δοκιμάστε να το ανοίξετε με έναν text editor και θα το διαπιστώσετε!

Οι κλάσεις `DataInputStream` και `DataOutputStream` χρησιμοποιούνται συνδυαστικά και είναι χρήσιμες σε κάποιες περιπτώσεις που χρειάζεται να διαβάσουμε ή να γράψουμε βασικούς τύπους δεδομένων της Java (π.χ., `int`, `float`, `double`, `boolean`) σε δυαδική μορφή, αντί για απλό κείμενο. Εξασφαλίζουν μικρό μέγεθος αρχείου και ανεξαρτησία από την πλατφόρμα.

Σε επόμενες ενότητες θα δούμε τη λειτουργία των κλάσεων `ObjectInputStream` και `ObjectOutputStream`.

Παρακάτω παρουσιάζονται μερικά από τα κυριότερα χαρακτηριστικά των τεσσάρων πιο σημαντικών κλάσεων από αυτές που είδαμε. Η κλάση `File`, που αναφέρεται, θα παρουσιαστεί παρακάτω.

FileInputStream

Κατασκευαστές

`FileInputStream(String name)` Δημιουργεί ένα ρεύμα `FileInputStream` ανοίγοντας μια σύνδεση σε ένα αρχείο με όνομα `name` στο σύστημα αρχείων.

`FileInputStream(File file)` Δημιουργεί ένα ρεύμα `FileInputStream` ανοίγοντας μια σύνδεση σε ένα αρχείο που παριστάνεται ως ένα αντικείμενο της κλάσης `File`(*).

(*) Θα την δούμε παρακάτω

Μέθοδοι

`void close()` Κλείνει το ρεύμα και απελευθερώνει τους σχετικούς πόρους του συστήματος.

`int read()` Διαβάζει από το ρεύμα και επιστρέφει το επόμενο byte ή -1 αν φθάσει στο τέλος αρχείου.

`int read(byte[] b)` Διαβάζει από το ρεύμα μέχρι `b.length` bytes και αποθηκεύει στο `b`. Επιστρέφει πλήθος bytes που διάβασε ή -1 για τέλος αρχείου.

FileOutputStream

Κατασκευαστές

`FileOutputStream(String name)` Δημιουργεί ένα ρεύμα `FileOutputStream` για εγγραφή σε ένα αρχείο με όνομα `name` στο σύστημα αρχείων.

`FileOutputStream(String name, boolean append)` Δημιουργεί ένα ρεύμα `FileOutputStream` για εγγραφή σε ένα αρχείο με όνομα `name` στο σύστημα αρχείων. Αν το `append` έχει τιμή `true`, τα bytes γράφονται στο τέλος του αρχείου (προσθήκη).

`FileOutputStream(File file)` Δημιουργεί ένα ρεύμα `FileOutputStream` για εγγραφή σε ένα αρχείο που παριστάνεται ως ένα αντικείμενο της κλάσης `File`.

`FileOutputStream(File file, boolean append)` Δημιουργεί ένα ρεύμα `FileOutputStream` για εγγραφή σε ένα αρχείο που παριστάνεται ως ένα αντικείμενο της κλάσης `File`. Αν το `append` έχει τιμή `true`, τα bytes γράφονται στο τέλος του αρχείου (προσθήκη).

Μέθοδοι
`void write(int b)` Γράφει το byte στο ρεύμα εξόδου (το τελευταίο byte του int).
`void write(byte[ ] b)` Γράφει στο ρεύμα εξόδου τα bytes του πίνακα `b`.

BufferedInputStream

Κατασκευαστές
`BufferedInputStream(InputStream in)` Δημιουργεί ένα αντικείμενο `BufferedInputStream` ενθυλακώνοντας ένα αντικείμενο `InputStream (π.χ. ένα FileInputStream)` και δίνοντας δυνατότητα buffering στο ρεύμα.
`BufferedInputStream(InputStream in, int size)` Παρόμοια, αλλά εδώ ορίζουμε το μέγεθος `size` του buffer.

Μέθοδοι
Παρόμοια με την `FileInputStream`

BufferedOutputStream

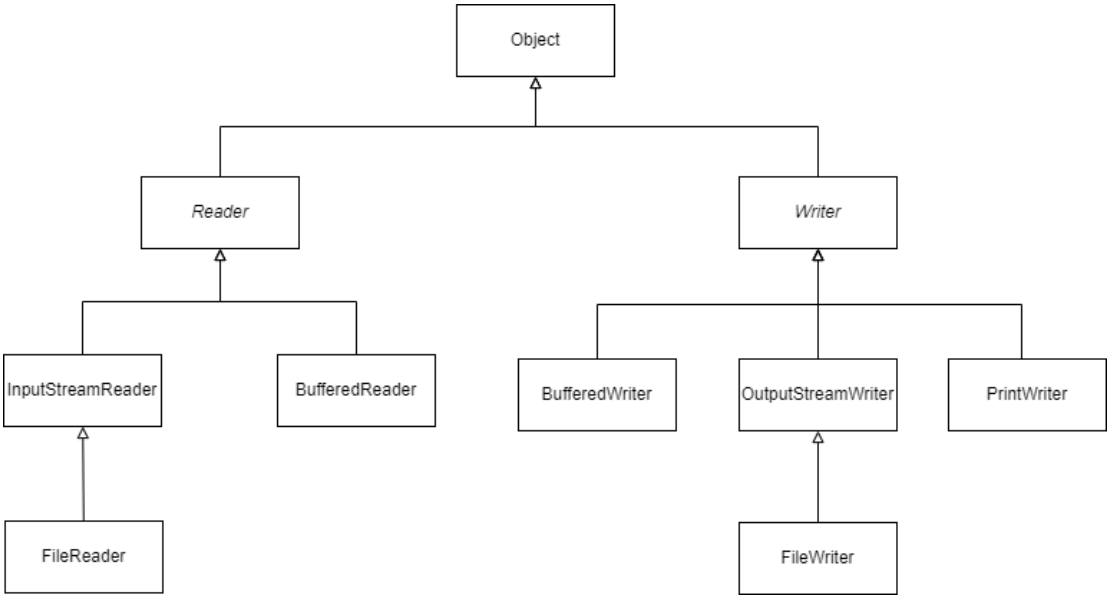
`BufferedOutputStream(OutputStream out)` Δημιουργεί ένα αντικείμενο `BufferedOutputStream` ενθυλακώνοντας ένα αντικείμενο `OutputStream` (π.χ. ένα `FileOutputStream`) και δίνοντας δυνατότητα buffering στο ρεύμα.
`BufferedOutputStream(OutputStream out, int size)` Παρόμοια, αλλά εδώ ορίζουμε το μέγεθος `size` του buffer.

Μέθοδοι
Παρόμοια με την `FileOutputStream`

8.3.3. Ρεύματα Χαρακτήρων (Character Streams)

Τα Ρεύματα Χαρακτήρων (Character Streams) είναι ένα σύνολο εργαλείων για τη διαχείριση δεδομένων κειμένου, τα οποία αναπαρίστανται ως χαρακτήρες (16-bit Unicode). Αυτό τα κάνει ιδανικά για αρχεία κειμένου και άλλες πηγές δεδομένων που αποτελούνται από ακολουθίες χαρακτήρων, καθώς διευκολύνουν την ορθή διαχείριση διαφορετικών συνόλων χαρακτήρων (character encoding).

Στην Εικόνα 18 βλέπουμε μια (απλοποιημένη) ιεραρχία κλάσεων που αφορά στη διαχείριση ρευμάτων χαρακτήρων.



Εικόνα 18 - ιεραρχία κλάσεων ρευμάτων χαρακτήρων (απλοποιημένη)

Η ιεραρχία `Reader` χρησιμοποιείται για ανάγνωση χαρακτήρων από πηγές εισόδου ενώ η ιεραρχία `Writer` για εγγραφή χαρακτήρων σε προορισμούς εξόδου.

Και οι δύο αυτές κλάσεις έχουν διάφορες εξειδικευμένες υλοποιήσεις (υποκλάσεις), οι οποίες καλύπτουν διαφορετικές ανάγκες. Ενδεικτικά, μπορούμε να δούμε σε αντιστοιχία (Πίνακας 2) μερικές από τις κυριότερες, για είσοδο και έξοδο.

Πίνακας 2 - μερικές από τις κυριότερες κλάσεις character stream

Κλάσεις Εισόδου (*Reader*)	**Κλάσεις Εξόδου (*Writer*)**
`FileReader` Ανάγνωση χαρακτήρων από αρχεία. Είναι η βασική κλάση για τη διαχείριση δεδομένων κειμένου από αρχείο.	`FileWriter` Χρησιμοποιείται για την εγγραφή χαρακτήρων σε αρχεία.
`BufferedReader` Παρέχει προσωρινή αποθήκευση (buffering) για πιο αποδοτική ανάγνωση. Επιπλέον, προσφέρει χρήσιμες μεθόδους, όπως η `readLine()`, για ανάγνωση ολόκληρων γραμμών κειμένου.	`BufferedWriter` Παρέχει προσωρινή αποθήκευση κατά την εγγραφή, βελτιώνοντας την απόδοση και προσφέροντας τη μέθοδο `newLine()` για αλλαγή (εισαγωγή νέας) γραμμής.
	`PrintWriter` Διευκολύνει την εγγραφή μορφοποιημένων δεδομένων κειμένου σε ρεύματα χαρακτήρων (π.χ. αρχεία, κονσόλα). Υποστηρίζει κωδικοποίηση χαρακτήρων. Η `PrintWriter` προσφέρει χρήσιμες μεθόδους για την

	εγγραφή δεδομένων. Τέτοιες είναι οι μέθοδοι `print()`, `println()`, `printf()` για μορφοποιημένη εγγραφή δεδομένων κειμένου, αριθμών, κ.λπ. Προσοχή, η `PrintWriter` αγνοεί τυχόν εξαιρέσεις κατά την εγγραφή. Ο κώδικας θα πρέπει να τις ελέγχει ξεχωριστά με τη μέθοδο `checkError()`.

Στο επόμενο παράδειγμα που δείχνει μια χρήση εγγραφής σε αρχείο κειμένου, θα ακολουθήσουμε διαφορετική πολιτική στο θέμα των εξαιρέσεων. Στα προηγούμενα χρησιμοποιήσαμε (σωστότερα) μια εντολή try-with-resources, εδώ επιλέγουμε να κάνουμε throw την περίπτωση εξαίρεσης, απλά και μόνο για λόγους επίδειξης δυνατοτήτων χειρισμού εξαιρέσεων.

Η κλάση `FileWriter` περιέχει μεθόδους που γράφουν σε ροές χαρακτήρων. Έχει διάφορους κατασκευαστές, ένας εκ των οποίων δέχεται αλφαριθμητικό με το όνομα του αρχείου (δείτε Εικόνα 19, από το Java API Documentation).

`FileWriter(String fileName)`	Constructs a `FileWriter` given a file name, using the default charset

Εικόνα 19 - ένας κατασκευαστής της FileWriter (Java API documentation)

Επίσης από το Java API documentation, βλέπουμε ότι η `FileWriter` περιλαμβάνει μεθόδους κληρονομημένες από την ιεραρχία κλάσεων (δείτε και Εικόνα 18) στην οποία ανήκει, μια από τις οποίες είναι η (υπερφορτωμένη) `write()`. Μια εκδοχή της γράφει ένα αλφαριθμητικό.

Method Summary

Methods declared in class java.io.OutputStreamWriter

close, flush, getEncoding, write, write, write

Methods declared in class java.io.Writer

append, append, append, nullWriter, write, write

Methods declared in class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Εικόνα 20 - μέθοδοι που κληρονομεί η FileWriter (Java API documentation)

Ο παρακάτω κώδικας θα δημιουργήσει ένα αρχείο κειμένου με όνομα `textfile1.txt` και θα γράψει μέσα σε αυτό, χρησιμοποιώντας την κλάση `FileWriter`.

```
import java.io.FileWriter;
import java.io.IOException;
public class Stream4 {
      public static void main(String[] args) throws IOException{
            FileWriter myFile = new FileWriter("textfile1.txt");
```

```
            myFile.write("This is a test. My first text written to a
file!");
            myFile.close();
            System.out.println("Done!");
      }
}
```

Στις δύο πρώτες γραμμές κάνουμε εισαγωγή των κλάσεων. Στη γραμμή της `main()` γίνεται διαχείριση της `IOException`, που πρέπει υποχρεωτικά να αντιμετωπίσουμε. Εδώ επιλέγουμε την «εύκολη» λύση του throw για να επικεντρώσουμε στα βασικά των κλάσεων `Writer`. Στη επόμενη γραμμή δημιουργούμε ένα αντικείμενο της κλάσης `FileWriter` περνώντας ως παράμετρο το πλήρες όνομα του αρχείου που θα δημιουργηθεί. Στη συνέχεια χρησιμοποιούμε την μέθοδο `write` (κληρονομημένη από την υπερκλάση της `FileWriter`, την `OutputStreamWriter`) και γράφουμε το κείμενο. Τέλος, κλείνουμε το ρεύμα με τη μέθοδο `close()`. Η `close()`, αν υπάρχουν δεδομένα που δεν έχουν γραφτεί, τα γράφει και μετά ελευθερώνει τους πόρους του συστήματος.

Το περιεχόμενο του αρχείου textfile1.txt θα είναι μόνο μια γραμμή και το βλέπουμε στην Εικόνα 21.

```
1  This is a test. My first text written to a file!
```

Εικόνα 21 - περιεχόμενο του αρχείου κειμένου textfile1.txt

Η κλάση `BufferedWriter`, είναι παρόμοια με την `FileWriter`, αλλά παρέχει buffering. Αυτό σημαίνει ότι αν ο κώδικας εκτελεί πολλαπλές εγγραφές, τότε κρατάει σε buffer τα δεδομένα και τα γράφει μαζικά, μειώνοντας έτσι τις προσβάσεις στο περιφερειακό.

```
import java.io.FileWriter;
import java.io.BufferedWriter;
import java.io.IOException;

public class Stream5 {
      public static void main(String[] args) throws IOException {
BufferedWriter        myFile        =        new        BufferedWriter(new
      FileWriter("textfile2.txt"));
            myFile.write("This is a test. My second text written to a
file!");
            myFile.newLine();
            myFile.write("\nUsing class BufferedWriter");
            myFile.close();
            System.out.println("Done!");
      }
}
```

Δείτε στην Εικόνα 22 το περιεχόμενο του αρχείου κειμένου που θα δημιουργηθεί .

```
1  This is a test. My second text written to a file!
2
3  Using class BufferedWriter
```

Εικόνα 22 - περιεχόμενο του αρχείου κειμένου textfile2.txt

Μετά τα imports των κλάσεων, δημιουργούμε ένα αντικείμενο `BufferedWriter` στο οποίο περνάμε ένα αντικείμενο `FileWriter` (wrapping). Η εγγραφή στο ρεύμα γίνεται με τη μέθοδο

write. Παρατηρήστε στο δημιουργημένο αρχείο κειμένου τη συμπεριφορά της μεθόδου στις διαδοχικές εντολές write, καθώς και την εισαγωγή αλλαγής γραμμής που γίνεται με τη μέθοδο newLine() της BufferedWriter. Δείτε επίσης ότι μέσα στο κείμενο που γράφουμε μπορούμε να συμπεριλάβουμε ειδικούς χαρακτήρες όπως π.χ. η αλλαγή γραμμής (\n).

Έχοντας δημιουργήσει με τους προηγούμενους κώδικες τα αρχεία textfile1.txt και textfile2.txt, στο επόμενο παράδειγμα ο κώδικας ανοίγει το ρεύμα εισόδου από το δεύτερο, διαβάζει το περιεχόμενό του και το προσθέτει στο πρώτο που το ανοίγει για έξοδο. Προσέξτε, δεν διαγράφουμε τα περιεχόμενα, αλλά προσθέτουμε τις νέες γραμμές (append). Για να γίνει αυτό, βάζουμε μια ακόμα παράμετρο με τιμή true στη δημιουργία του αντικειμένου FileWriter που σηματοδοτεί προσθήκη και όχι διαγραφή του υπάρχοντος. Διαβάζουμε από το ρεύμα εισόδου μια-μια τις γραμμές χρησιμοποιώντας τη μέθοδο readLine(). Η μέθοδος αυτή επιστρέφει την επόμενη γραμμή και αν φθάσει στο τέλος αρχείου επιστρέφει την τιμή null, την οποία ελέγχουμε στο while. Πριν το while, για να γράψει στη δεύτερη γραμμή του textfile1.txt και όχι «κολλητά» στο υπάρχον κείμενο, εισάγουμε πρώτα μια νέα γραμμή με τη newLine(). Δεν χρειάζεται να κάνουμε κάτι για να κλείσουμε τα αρχεία, αφού χρησιμοποιούμε try-with-resources.

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

public class Stream6 {
	public static void main(String[] args) {
		String inputFile = "textfile2.txt";
		String outputFile = "textfile1.txt";
		// Διαβάζει από το input και προσθέτει το περιεχόμενο στο
output
		try (BufferedReader reader = new BufferedReader(new
FileReader(inputFile));
		BufferedWriter writer = new BufferedWriter(new
FileWriter(outputFile, true))) { // true για append
			String line;
			writer.newLine(); // αλλαγή γραμμής
			while ((line = reader.readLine()) != null) {
				writer.write(line); // γράφει στο output αρχείο
				writer.newLine(); // αλλαγή γραμμής
			}
			System.out.println("Η αντιγραφή και προσθήκη ολοκληρώθηκε
με επιτυχία.");
		} catch (IOException e) {
			e.printStackTrace();
		}
	}
}
```

Αν θέλουμε περισσότερες ευκολίες διαχείρισης των δεδομένων μας στην εγγραφή κειμένου, μπορούμε να ενθυλακώσουμε το αντικείμενο BufferedWriter σε ένα νέο αντικείμενο της κλάσης PrintWriter που επίσης συμπεριλαμβάνεται στο πακέτο java.io. Ανατρέχοντας και πάλι στο Java API documentation μπορούμε να δούμε μια σειρά μεθόδων όπως print, println

και `printf` που διαθέτει η `PrintWriter` για το σκοπό αυτό. Δείτε μέρος της τεκμηρίωσης στην Εικόνα 23.

void	**print**(float f) Prints a floating-point number.
void	**print**(int i) Prints an integer.
void	**print**(long l) Prints a long integer.
void	**print**(**Object** obj) Prints an object.
void	**print**(**String** s) Prints a string.
PrintWriter	**printf**(**Locale** l, **String** format, **Object**... args) A convenience method to write a formatted string to this writer using the specified format string and arguments.

Εικόνα 23 - τμήμα τεκμηρίωσης Java API για την PrintWriter

Στον κώδικα που ακολουθεί, δείτε πώς έχουμε δημιουργήσει το αντικείμενο `PrintWriter` μέσα στο οποίο περνάμε ως όρισμα ένα αντικείμενο `BufferedWriter`, το οποίο δέχεται ως όρισμα ένα αντικείμενο `FileWriter`. Με αυτό τον τρόπο η διαχείριση της ροής θα γίνεται με χρήση buffering και επιπλέον έχουμε και τις πρόσθετες δυνατότητες εκτύπωσης της `PrintWriter`, που ενδεικτικά παρουσιάζονται με τις μεθόδους `printf`, `print` και `println`.

```
import java.io.*;

public class Stream7 {
	public static void main(String[] args) throws IOException {
		String filename = "myText.txt";
		double[] numbers = { 12.3, 1.44, 4.22 };
		PrintWriter myStream = new PrintWriter(new BufferedWriter(new
FileWriter(filename)));
		myStream.printf("The numbers in file %s are:\n", filename);
		for (int i = 0; i < 3; i++)
			if (i == 2)
				myStream.print(numbers[i]);
			else
				myStream.println(numbers[i]);
		myStream.close();
	}
}
```

Δείτε στην Εικόνα 24 το περιεχόμενο του αρχείου που θα δημιουργήσει ο παραπάνω κώδικας.

```
1  The numbers in file C:\data\myText.txt are:
2  12.3
3  1.44
4  4.22
```

Εικόνα 24 - αρχείο με χρήση μεθόδων της PrintWriter

Παρακάτω παρουσιάζονται μερικά από τα κυριότερα χαρακτηριστικά των πιο σημαντικών κλάσεων από αυτές που είδαμε. Η κλάση File θα παρουσιαστεί παρακάτω.

FileReader

Κατασκευαστές
`FileReader(String fileName)` Δημιουργεί ένα αντικείμενο `FileReader` για το αρχείο που ορίζεται από το όνομά του `filename`.
`FileReader(File file)` Δημιουργεί ένα αντικείμενο `FileReader` για το αρχείο που ορίζεται από ένα αντικείμενο `File`.

Μέθοδοι
`int read()` Διαβάζει έναν χαρακτήρα από το αρχείο και επιστρέφει τον ακέραιο κωδικό του, ή -1 αν έχει φτάσει στο τέλος του αρχείου.
`void close()` Κλείνει το `FileReader` και απελευθερώνει τους πόρους του συστήματος.

FileWriter

Κατασκευαστές
`FileWriter(String fileName)` Δημιουργεί ένα αντικείμενο `FileWriter` που γράφει σε ένα αρχείο με το όνομα `fileName`. Εάν το αρχείο δεν υπάρχει, δημιουργείται νέο. Αν υπάρχει ήδη, τα δεδομένα του αρχείου θα αντικατασταθούν από αυτά που θα γράψουμε.
`FileWriter(String fileName, boolean append)` Δημιουργεί ένα αντικείμενο FileWriter για το αρχείο `fileName`, όπου η παράμετρος `append` με τιμή `true` ορίζει ότι θέλουμε να προσθέτουμε δεδομένα στο τέλος του αρχείου (αντί να αντικαθιστούμε το περιεχόμενό του).
`FileWriter(File file)` και `FileWriter(File file, boolean append)` Αντίστοιχη λειτουργία με τις δύο παραπάνω, όπου αντί για το όνομα του αρχείου περνάμε ένα αντικείμενο της κλάσης `File`.

Μέθοδοι
`void write(int c)` Γράφει έναν χαρακτήρα στο αρχείο (τα τελευταία 16 bits του ακεραίου `c`).
`void write(String str)` Γράφει ένα αλφαριθμητικό στο αρχείο.
`void close()` Κλείνει το `FileWriter` και απελευθερώνει τους πόρους του. Κλείνοντας εξασφαλίζουμε ότι τα δεδομένα έχουν γραφτεί στο αρχείο.

BufferedReader

Κατασκευαστές
`BufferedReader(Reader in)` Δημιουργεί έναν `BufferedReader` που διαβάζει δεδομένα από τον `Reader (π.χ. FileReader)` παρέχοντας buffering.
`BufferedReader(Reader in, int sz)` Δημιουργεί έναν BufferedReader με `sz` μέγεθος buffer.

Μέθοδοι
`int read()` Διαβάζει έναν χαρακτήρα από το ρεύμα και επιστρέφει τον ακέραιο κωδικό του χαρακτήρα ή -1 εάν φτάσει στο τέλος του αρχείου.

`String readLine()` Διαβάζει μία γραμμή κειμένου και επιστρέφει το αλφαριθμητικό, χωρίς το χαρακτήρα νέας γραμμής στο τέλος. Επιστρέφει `null` αν έχει φτάσει στο τέλος του αρχείου.
`void close()` Κλείνει τον `BufferedReader` και απελευθερώνει τους πόρους του.

BufferedWriter

Κατασκευαστές
`BufferedWriter(Writer out)` Δημιουργεί έναν `BufferedWriter` που γράφει δεδομένα στον `Writer` (π.χ. `FileWriter`) με χρήση buffer.
`BufferedWriter(Writer out, int size)` Δημιουργεί έναν `BufferedWriter` με καθορισμένο μέγεθος buffer `size`.

Μέθοδοι
`void write(int c)` Γράφει έναν χαρακτήρα (ως ακέραιο `c`) στον buffer.
`void write(String str)` Γράφει ένα αλφαριθμητικό στον buffer.
`void newLine()` Εισάγει μια νέα γραμμή στο αρχείο, χρησιμοποιώντας την κατάλληλη κωδικοποίηση για την πλατφόρμα.
`void close()` Κλείνει τον `BufferedWriter` και απελευθερώνει τους πόρους του, εξασφαλίζοντας ότι όλα τα δεδομένα έχουν γραφτεί στο αρχείο πριν το κλείσιμο.

PrintWriter

Κατασκευαστές
`PrintWriter(Writer out)` Δημιουργεί έναν `PrintWriter` που χρησιμοποιεί το `Writer` (π.χ. ένα `FileWriter`) ως προορισμό εξόδου.
`PrintWriter(String fileName)` Δημιουργεί έναν `PrintWriter` που γράφει σε ένα αρχείο με όνομα `fileName`. Αν το αρχείο δεν υπάρχει, θα δημιουργηθεί.

Μέθοδοι
`void print(Object obj)` και `void print(String s)` Εκτυπώνει αντικείμενα ή συμβολοσειρές (αλλά και άλλους τύπους δεδομένων όπως `int`, `double` κ.λπ.) χωρίς αλλαγή γραμμής.
`void println(Object obj)` και `void println(String s)` Εκτυπώνει ένα αντικείμενο ή ένα αλφαριθμητικό (ή άλλους απλούς τύπους δεδομένων), προσθέτοντας νέα γραμμή στο τέλος.
`void printf(String format, args)` Εκτυπώνει μορφοποιημένο κείμενο με βάση μια συμβολοσειρά `format` (όπως στην `System.out.printf`).
`void format(String format, args)` Παρόμοια με την `printf`, μορφοποιεί και εκτυπώνει κείμενο με βάση τη συμβολοσειρά `format`.
`void close()` Κλείνει τον `PrintWriter` και απελευθερώνει τους πόρους του, διασφαλίζοντας ότι όλα τα δεδομένα έχουν γραφτεί.

8.3.3.1 Παράδειγμα εγγραφής και ανάγνωσης κλάσης σε αρχείο κειμένου

Υποθέστε ότι σε μια εφαρμογή έχουμε μια κλάση `Student` με πεδία το όνομα, την ηλικία και τα μαθήματα που ο μαθητής παρακολουθεί. Κάποια στιγμή έχουμε ένα στιγμιότυπο (αντικείμενο της κλάσης για έναν συγκεκριμένο μαθητή) και θέλουμε να αποθηκεύσουμε τα δεδομένα που περιέχει

σε ένα αρχείο κειμένου, προκειμένου σε επόμενη φάση να μπορούμε να τα ανακτήσουμε. Ο κώδικας μπορεί να είναι σε κάποια μορφή όπως αυτή που ακολουθεί.

```
// Η κλάση Student
import java.io.*;
import java.util.Arrays;

public class Student {
	private String name;
	private int age;
	private String[] subjects;

	public Student(String name, int age, String[] subjects) {
		this.name = name;
		this.age = age;
		this.subjects = subjects;
	}

	// Αποθήκευση των δεδομένων σε αρχείο κειμένου
	public void saveToTextFile(String fileName)  {
		try	(PrintWriter	writer	=	new	PrintWriter(new
FileWriter(fileName))) {
			writer.println(name);
			writer.println(age);
			writer.println(String.join(",", subjects)); // συνδυάζει
τα μαθήματα με κόμμα
		} catch (IOException e) {
			System.err.println("Could not create the file");
			System.err.println(e.getMessage());
		}
	}

	// Ανάγνωση δεδομένων από αρχείο κειμένου
	public static Student readFromTextFile(String fileName)  {
		try	(BufferedReader	reader	=	new	BufferedReader(new
FileReader(fileName))) {
			String name = reader.readLine();
			int age = Integer.parseInt(reader.readLine());
			// διαχωρισμός μαθημάτων με κόμμα
			String[] subjects = reader.readLine().split(",");
			return new Student(name, age, subjects);
		} catch (FileNotFoundException e) { //FileReader Exception
			System.err.println("Could not open the file");
			System.err.println(e.getMessage());
		} catch (IOException e) { //BufferedReader.readLine() Exception
			System.err.println("Could not read the file");
			System.err.println(e.getMessage());
		}	catch	(NumberFormatException	e)	{	//Integer.parseInt
Exception
			System.err.println("Error parsing age");
			System.err.println(e.getMessage());
		}catch (Exception e) { // Any other Exception
			System.err.println("Unexpected error");
			System.err.println(e.getMessage());
		}
		return null;
	}
	@Override
```

```
	public String toString() {
		return "Student{name=" + name + ", age=" + age + ", subjects="
+ Arrays.toString(subjects) + "}";
	}
}
```

Η κλάση `Student` περιέχει τρία πεδία, όνομα (`String name`), ηλικία (`int age`) και μαθήματα που παρακολουθεί σε μορφή πίνακα από αλφαριθμητικά (`String[] subjects`). Ο κατασκευαστής αναθέτει τις τιμές που του παρέχονται ως ορίσματα.

Για να εμφανίζουμε τα πεδία των αντικειμένων της κλάσης κάνουμε override τη μέθοδο `toString()` που, ως γνωστόν, μια εκδοχή της περιλαμβάνεται σε κάθε αντικείμενο, καθώς έρχεται από την κλάση `Object`.

Η μέθοδος `saveToTextFile()` δέχεται ως όρισμα το όνομα του αρχείου και χρησιμοποιεί την `PrintWriter` για να γράψει τα δεδομένα. Για να γράψουμε τα μαθήματα του μαθητή, χρησιμοποιούμε τη μέθοδο `String.join()` που τα συνδυάζει σε ένα αλφαριθμητικό με ενδιάμεσα κόμμα.

Η μέθοδος `readFromTextFile()` δέχεται ως όρισμα το όνομα του αρχείου και χρησιμοποιεί την `BufferedReader()` για να διαβάσει γραμμή-γραμμή τα περιεχόμενα του αρχείου. Για κάθε γραμμή ανάλογα με τον τύπο των δεδομένων κάνει την σχετική επεξεργασία, καθώς οι αναγνώσεις με την `readLine()` επιστρέφουν αλφαριθμητικό: μετατροπή του `age` σε ακέραιο με την `Integer.ParseInt()`, διαχωρισμός της γραμμής με τα μαθήματα με τη μέθοδο `String.split()` και δημιουργία πίνακα αλφαριθμητικών. Η μέθοδος έχει δηλωθεί `static` για να μπορούμε να την καλέσουμε χωρίς πριν να δημιουργήσουμε αντικείμενο, αφού η δουλειά της είναι να μας δημιουργήσει αυτή ένα αντικείμενο `Student` και να το επιστρέψει.

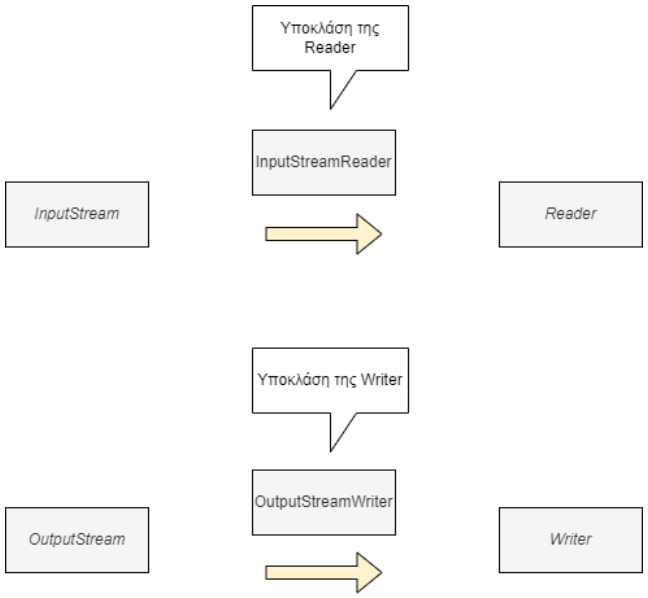
Και στις δύο μεθόδους χρησιμοποιείται δομή try-with-resources οπότε τα ρεύματα κλείνουν αυτόματα. Επίσης και στις δύο μεθόδους γίνεται παγίδευση των λαθών, στα σχόλια είναι σημειωμένο κάθε `Exception` ποια λειτουργία αφορά. Στη μέθοδο `readFromTextFile()` εφόσον προκύψει λάθος επιστρέφεται η τιμή `null`.

```
public class Main {
	public static void main(String[] args)  {
		String fileName = "student.txt";
		Student student = new Student("John Smith", 20, new String[] {
"Math", "Science", "History" });
		// Αποθήκευση σε αρχείο κειμένου
		student.saveToTextFile(fileName);
		// Ανάκτηση από αρχείο κειμένου
		Student loadedStudent = Student.readFromTextFile(fileName);
		if (loadedStudent != null) System.out.println("Loaded Student:
" + loadedStudent);
	}
}
```

Η κλάση `Main` περιέχει τη μέθοδο `main()` και κάνει μια ενδεικτική χρήση των μεθόδων της `Student`. Δημιουργεί και αρχικοποιεί ένα αντικείμενο `Student` και στη συνέχεια με χρήση των μεθόδων της `Student` γράφει στο αρχείο κειμένου τα δεδομένα και μετά τα διαβάζει και τα εμφανίζει στην οθόνη (εφόσον δεν επιστραφεί `null`).

8.3.4. Μετατροπές ρευμάτων

Οι κλάσεις `InputStreamReader` και `OutputStreamWriter` χρησιμοποιούνται για τη μετατροπή ρευμάτων δεδομένων από bytes σε χαρακτήρες και αντίστροφα, διασφαλίζοντας τη σωστή κωδικοποίηση και αποκωδικοποίηση του κειμένου. Ειδικότερα (Εικόνα 25), ένα `InputStream` μπορεί να μετατραπεί σε `Reader` μέσα από την κλάση `InputStreamReader`. Αντίστροφα, ένα `OutputStream` μπορεί να μετατραπεί σε `Writer` μέσα από την κλάση `OutputStreamWriter`.



Εικόνα 25 - μετατροπή ρευμάτων

Στα επόμενα, θα δούμε περιπτώσεις όπου μπορεί να χρειαστούν οι κλάσεις αυτές καθώς και τον τρόπο με τον οποίο θα τις χρησιμοποιήσουμε.

8.3.4.1 Μετατροπή δυαδικού ρεύματος σε ρεύμα χαρακτήρα – InputStreamReader

Σκεφτείτε, ενδεικτικά, τις παρακάτω περιπτώσεις χρήσης, που παρουσιάζονται συχνά:

- Ένας περιηγητής ιστού στέλνει αιτήματα HTTP και λαμβάνει την απόκριση (π.χ. HTML ή JSON) σε μορφή bytes. Τα bytes πρέπει να μετατραπούν σε κείμενο με συγκεκριμένη κωδικοποίηση (π.χ. UTF-8).
- Ένας αισθητήρας στέλνει τα δεδομένα σε bytes. Η εφαρμογή πρέπει να τα μετατρέψει σε κείμενο με τη σωστή κωδικοποίηση.
- Μια εφαρμογή θέλει να διαβάσει δεδομένα από ένα αρχείο κειμένου που δεν είναι στην προεπιλεγμένη κωδικοποίηση του συστήματος (π.χ. UTF-16). Θα πρέπει ο κώδικας να εφαρμόσει την κατάλληλη κωδικοποίηση για να μπορέσει να το διαβάσει σωστά.

Η `InputStreamReader` ενθυλακώνει ένα `InputStream` (ροή bytes) και το μετατρέπει σε `Reader` (ροή χαρακτήρων), δηλαδή, ερμηνεύει τη ροή των bytes του ρεύματος ως κείμενο με συγκεκριμένη κωδικοποίηση. Η διαδικασία φαίνεται στον επόμενο κώδικα. Ως πηγή δεδομένων dataSource χρησιμοποιεί ένα αρχείο, αλλά μπορεί να είναι κάτι άλλο, π.χ. ένα ρεύμα που έρχεται

από το διαδίκτυο. Επιλέγουμε να μην χειριστούμε τυχόν εξαιρέσεις (throws), για λόγους επικέντρωσης στο αντικείμενο που θέλουμε να δείξουμε.

```
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.Reader;
import java.io.IOException;

public class Stream8 {
	public static void main(String[] args) throws IOException {
		// το αρχείο με τα δεδομένα
		String dataSource = "c:\\data\\data.dat";
		// 1. το δυαδικό ρεύμα
		InputStream inStream = new FileInputStream(dataSource);
		// 2. δυαδικό ρεύμα -> ρεύμα χαρακτήρων με κωδικοποίηση UTF-8
		Reader inStreamReader = new InputStreamReader(inStream, "UTF-
8");
		// 3. για buffering
		BufferedReader bufferedReader = new
BufferedReader(inStreamReader);
		String line;
		while ((line = bufferedReader.readLine()) != null)
			System.out.println(line);
		// απελευθέρωση πόρων συστήματος
		bufferedReader.close();
	}
}
```

Κοιτώντας τον παραπάνω κώδικα, μπορούμε να παρατηρήσουμε ότι για να μετατρέψουμε (ερμηνεύσουμε) ένα δυαδικό αρχείο ως αρχείο κειμένου μπορούμε να ακολουθήσουμε τα παρακάτω βήματα:

1. Δημιουργία `InputStream`
2. Πέρασμα (ενθυλάκωση) του `InputStream` στον `InputStreamReader` για δημιουργία `Reader`
3. Πέρασμα (ενθυλάκωση) του `Reader` σε `BufferedReader` για καλύτερη απόδοση

Σχηματικά μπορούμε να πούμε ότι στο βήμα 1 βρισκόμαστε στον κόσμο των bytes, στο βήμα 2 περνάμε από τον κόσμο των bytes στον κόσμο των χαρακτήρων και στο βήμα 3 βρισκόμαστε στον κόσμο των χαρακτήρων!

Κατά τ' άλλα η συνέχεια του κώδικα είναι γνωστή, χρησιμοποιούμε την μέθοδο `readLine()` της κλάσης `BufferedReader` για να διαβάσουμε μια-μια τη γραμμή και να την εμφανίσουμε στην οθόνη.

8.3.4.2 Μετατροπή ρεύματος χαρακτήρων σε δυαδικό ρεύμα – OutputStreamWriter

Σκεφτείτε, ενδεικτικά, τις παρακάτω περιπτώσεις χρήσης, που παρουσιάζονται συχνά:

- Μια εφαρμογή θέλει να στείλει δεδομένα κειμένου μέσω δικτύου. Για να γίνει αυτό, θα πρέπει να μετατραπούν σε bytes και με τη σωστή κωδικοποίηση (π.χ. UTF-8).

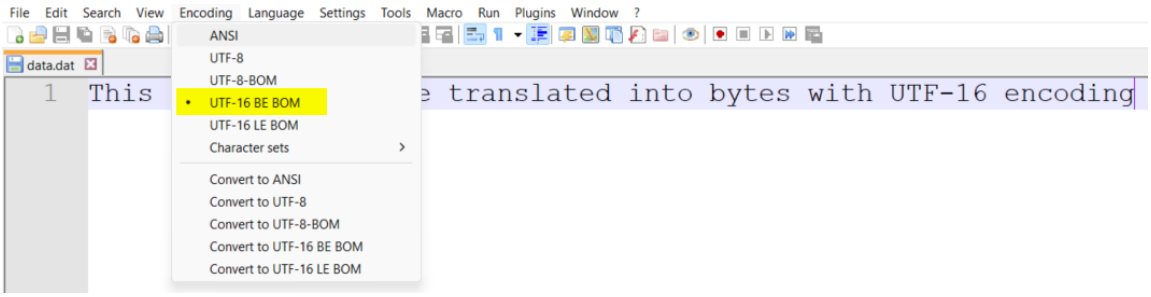
- Μια εφαρμογή θέλει να δημιουργήσει από κείμενο, αρχείο με συγκεκριμένη κωδικοποίηση, π.χ. UTF-16.
- Ένας Web server θέλει να απαντήσει σε αίτημα HTTP, οπότε πρέπει να στείλει δεδομένα (π.χ. HTML ή JSON) με συγκεκριμένη κωδικοποίηση (και πάλι, συνήθως UTF-8).

Η `OutputStreamWriter` ενθυλακώνει ένα `OutputStream` (ροή bytes) και το μετατρέπει σε `Writer` (ροή χαρακτήρων), δηλαδή, ερμηνεύει τη ροή των χαρακτήρων του ρεύματος ως bytes, όπως μπορούμε να δούμε στον επόμενο κώδικα.

```
import java.io.OutputStream;
import java.io.OutputStreamWriter;
import java.io.BufferedWriter;
import java.io.FileOutputStream;
import java.io.Writer;
import java.io.IOException;

public class Stream9 {
	public static void main (String [] args) throws IOException{
		String data = "This text needs to be translated into bytes with
UTF-16 encoding";
		String dataTarget = "data.dat";
		OutputStream oStream = new FileOutputStream(dataTarget);
		Writer oStreamWriter = new OutputStreamWriter(oStream, "UTF-
16");
		BufferedWriter bufferedWriter = new
BufferedWriter(oStreamWriter);
		bufferedWriter.write(data);
		bufferedWriter.close();
	}
}
```

Αρχικά δημιουργούμε ένα `OutputStream`. Ως αποδέκτη της ροής εξόδου ορίζουμε ένα αρχείο (dataTarget), αλλά μπορεί να είναι και κάτι άλλο, π.χ. διαδίκτυο. Στη συνέχεια χρησιμοποιούμε την `OutputStreamWriter` στην οποία ενθυλακώνουμε το `OutputStream` για να δημιουργήσουμε ένα αντικείμενο `Writer` (ουσιαστικά ένα αντικείμενο `OutputStreamWriter`, αλλά θυμηθείτε, η `OutputStreamWriter` είναι υποκλάση της `Writer` οπότε έχουν σχέση «is-a»).

Για να έχουμε buffering, περνάμε το αντικείμενο `Writer` σε μια `BufferedWriter`. Όλα τα byte που γράφονται στο `BufferedWriter` θα αποθηκεύονται πρώτα σε προσωρινή μνήμη μέσα σε έναν εσωτερικό πίνακα byte του `BufferedWriter`. Όταν το buffer γεμίσει, τότε αδειάζει όλο το περιεχόμενό του (flush) στο υποκείμενο `OutputStreamWriter` μεμιάς. Τέλος, κλείνουμε το `BufferedWriter` (αρκεί μόνο αυτό) για να απελευθερώσουμε τους πόρους του συστήματος. Δείτε στην Εικόνα 26 όπου έχουμε ανοίξει το δημιουργημένο αρχείο με τον κειμενογράφο Notepad++.



Εικόνα 26 - αρχείο με κωδικοποίηση UTF-16

8.3.5. Standard Είσοδος και Έξοδος (Standard I/O)

Όταν εκτελούμε ένα πρόγραμμα, γίνεται αυτόματα το import του πακέτου `java.lang`. Το πακέτο αυτό περιλαμβάνει την κλάση `System` η οποία, μεταξύ άλλων, περιέχει τρεις `public`, `static` και `final` μεταβλητές ρεύματος, τις `in`, `out` και `err` με τις οποίες δημιουργούνται αυτόματα τρία ρεύματα που συσχετίζονται με συσκευές του συστήματος.

Εξ ορισμού, το ρεύμα `System.in` δίνει τη δυνατότητα στο πρόγραμμα να εισαγάγει bytes από το πληκτρολόγιο. Το ρεύμα `System.out` δίνει τη δυνατότητα στο πρόγραμμα να εξαγάγει χαρακτήρες στην οθόνη. Τέλος το ρεύμα `System.err` δίνει τη δυνατότητα εξαγωγής μηνυμάτων σφάλματος (επίσης χαρακτήρες) στην οθόνη. Όλα τα ρεύματα μπορούν να ανακατευθυνθούν σε άλλο ή από άλλο μέσο με τη βοήθεια μεθόδων που παρέχει η κλάση `System`.

Το `System.in` είναι αντικείμενο της κλάσης `InputStream`, ενώ τα `System.out` και `System.err` είναι αντικείμενα της κλάσης `PrintStream` (δείτε και Εικόνα 17).

8.3.5.1 Η Κλάση System

Η κλάση `System` είναι μια τελική (final) κλάση που παρέχει στατικά πεδία και μεθόδους για αλληλεπίδραση με το λειτουργικό σύστημα και διαχείριση πόρων της Java εφαρμογής. Όπως είδαμε, περιλαμβάνει ρεύματα για την τυπική είσοδο, έξοδο και έξοδο σφαλμάτων, αλλά και άλλες λειτουργίες, όπως π.χ. για διαχείριση των μεταβλητών περιβάλλοντος, των ιδιοτήτων του συστήματος, κ.λπ. Μερικά παραδείγματα μπορούμε να δούμε στον κώδικα που ακολουθεί και τμήμα της εξόδου εκτέλεσης στην Εικόνα 27 (η αρίθμηση για λόγους σύγκρισης κώδικα με έξοδο).

```
public class Stream10 {
      public static void main(String[] args) {
            // τρέχων χρόνος σε χιλιοστά δευτερολέπτου με εκκίνηση την 1-1-
1970
            long currentTime1 = System.currentTimeMillis();
            // όλες οι ιδιότητες (ρυθμίσεις) του συστήματος σε μορφή
κλειδί=τιμή
            System.out.println("1 "+System.getProperties());
            // τιμή για το κλειδί java.vendor
            System.out.println( "2 "+System.getProperty("java.vendor"));
            // τιμή για το κλειδί java.version
            System.out.println("3 "+System.getProperty("java.version"));
            // τιμή για το κλειδί os.name
            System.out.println("4 "+ System.getProperty("os.name"));
            // τι περιέχει η μεταβλητή περιβάλλοντος PATH
            System.out.println("5 "+System.getenv("PATH"));
```

```
            // τρέχων χρόνος σε χιλιοστά δευτερολέπτου με εκκίνηση την 1-1-
1970
            long currentTime2 = System.currentTimeMillis();
            // χρόνος που διήρκεσε η εκτέλεση του κώδικα
            System.out.printf("Program                took                %d
milliseconds".formatted(currentTime2-currentTime1));
      }
}
```

```
1 {java.specification.version=21, sun.cpu.isalist=amd64, sun.jnu.encoding=UTF-8
, java.vm.specification.vendor=Oracle Corporation, java.specification.name=Java
2 Oracle Corporation
3 21
4 Windows 10
5 C:\Program Files\Common Files\Oracle\Java\javapath;C:\Python310\Scripts\;C:\P
Program took 3 milliseconds
```

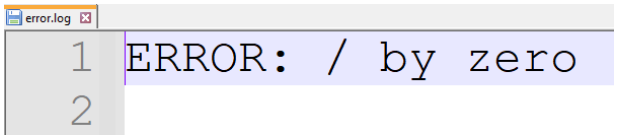
Εικόνα 27 - δείγμα της εξόδου εκτέλεσης κώδικα της κλάσης stream10

Τα ρεύματα `in`, `out`, `err` μπορούν να ανακατευθυνθούν με τη βοήθεια των μεθόδων `setIn(InputStream aStream)`, `setOut(PrintStream aStream)` και `setErr(PrintStream aStream)` αντίστοιχα.

Για παράδειγμα, μπορούμε να ανακατευθύνουμε το ρεύμα `System.err` ώστε τα γενικού χαρακτήρα μηνύματα να εξακολουθούν να στέλνονται στην οθόνη (το προκαθορισμένο για το `System.out`), ενώ τα μηνύματα σφαλμάτων να μην στέλνονται πλέον στην οθόνη, αλλά να γράφονται σε ένα log file. Αυτό κάνει ο κώδικας που ακολουθεί.

```
import java.io.*;

public class Stream11 {
      public static void main(String[] args) throws IOException {
            // δημιουργία ρεύματος για έξοδο σφαλμάτων
            PrintStream      fileErr      =      new      PrintStream(new
FileOutputStream("error.log"));
            // ρύθμιση της System.err να γράφει στο αρχείο
            System.setErr(fileErr);
            try {
                  int result = 10 / 0; // Exception: διαίρεση με μηδέν
            } catch (Exception e) {
                  System.err.println("ERROR:   "   +   e.getMessage());   //
γράφεται στο error.log
            }
            fileErr.close(); // απελευθέρωση πόρου
      }
}
```

Στον παραπάνω κώδικα, δημιουργούμε ένα ρεύμα εξόδου (`PrintStream`) που συνδέεται με αρχείο δίσκου με όνομα `error.log`. Στη συνέχεια με τη μέθοδο `setErr()` ανακατευθύνουμε στο αρχείο αυτό το ρεύμα `System.err`, αντί για οθόνη που είναι το προκαθορισμένο. Για δοκιμή, το επόμενο τμήμα του κώδικα προκαλεί μια εξαίρεση (διαίρεση με μηδέν) και με παγίδευση λαθών στέλνει το μήνυμα στο ρεύμα `System.err`, που τελικά ανακατευθύνεται στο αρχείο `error.log` όπου γράφεται το μήνυμα (Εικόνα 28).



Εικόνα 28 - ανακατεύθυνση του System.err στο αρχείο "error.log"

8.3.5.2 Η Κλάση Console

Η κλάση Console διευκολύνει κάποιες από τις λειτουργίες που ήδη έχουμε αν χρησιμοποιήσουμε τα ρεύματα `System.in` και `System.out`. Επιπλέον παρέχει και μερικά ακόμα χαρακτηριστικά που δεν παρέχονται άμεσα από αυτά τα ρεύματα. Ειδικότερα:

- Επιτρέπει την εισαγωγή κωδικών πρόσβασης που δεν εμφανίζει το κείμενο στην οθόνη με τη μέθοδο `readPassword()`.
- Διαθέτει καλύτερη και πιο εύκολη μορφοποίηση της εξόδου στην οθόνη με τις μεθόδους `printf()` και `String.format()`.
- Διαθέτει τη μέθοδο readLine() με την οποία μπορεί να διαβάσει μια γραμμή. Θυμηθείτε ότι για να έχετε είσοδο με την `System.in` χρειάζεστε να χρησιμοποιήσετε κλάσεις όπως η `Scanner`.
- Είναι καλύτερα οργανωμένη, καθώς τόσο η είσοδος όσο και η έξοδος (από και προς) κονσόλα γίνονται με ένα αντικείμενο `Console` (που περιέχει ένα αντικείμενο `PrintWriter` και ένα `Reader`).

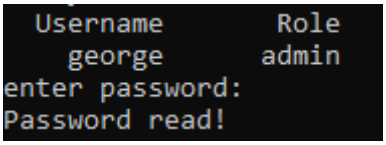
Σημειώστε, ωστόσο, ότι η `Console` είναι διαθέσιμη μόνο όταν η εφαρμογή εκτελείται σε πραγματική κονσόλα, και δεν λειτουργεί μέσα σε περιβάλλοντα όπως είναι τα περισσότερα IDEs. Η `Console` δεν έχει κατασκευαστή και χρησιμοποιούμε ένα αντικείμενο αυτού του τύπου με τη δήλωση `Console con = System.console();`

Αν η `System.console()` επιστρέψει `null`, η εφαρμογή δεν τρέχει σε κονσόλα και πρέπει να γίνει εναλλακτική διαχείριση της εισόδου/εξόδου. Ο κώδικας που ακολουθεί δεν θα εκτελεστεί σε περιβάλλον IDE, για να τον εκτελέσετε θα πρέπει να μεταγλωττιστεί και να τρέξει σε Command Prompt.

```
import java.io.Console;
public class Stream12 {
	public static void main(String [] args) {
		Console con = System.console();
		if (con == null) System.out.println("no console");
		else {
			String frmt = "%1$10s %2$10s\n";
			con.printf(frmt, "Username","Role");
			con.printf(frmt, "george", "admin");
			con.printf("enter password:");
			char[] pass = con.readPassword();
			con.printf("Password read!");
		}
	}
}
```

Χρησιμοποιούμε ένα αντικείμενο `Console`, το `con`, και ελέγχουμε αν έχει πάρει τιμή ή είναι `null`, οπότε στη δεύτερη περίπτωση ο κώδικας τερματίζει. Διαφορετικά, χρησιμοποιεί την

`printf(format, args)` για να εμφανίσει σε στήλες τα δεδομένα και την `readPassword()` για να διαβάσει κωδικό (η μέθοδος επιστρέφει πίνακα χαρακτήρων, κατά την πληκτρολόγηση οι χαρακτήρες δεν εμφανίζονται στην οθόνη). Δείτε στην Εικόνα 29 την οθόνη μετά την εκτέλεση του κώδικα σε παράθυρο εντολών.



Εικόνα 29 - εκτέλεση προγράμματος με χρήση Console

Τελειώνοντας, δείτε πώς χρησιμοποιούμε το `String frmt` για να το περάσουμε ως παράμετρο μορφοποίησης (format) στην `printf`: τα `%1$` και `%2$` είναι placeholders (που έρχονται σε αντιστοιχία με τις παραμέτρους `args` που ακολουθούν). Μετά το `$` δηλώνουμε πόσες θέσεις θέλουμε να καταλάβει το αλφαριθμητικό όρισμα (που σηματοδοτείται ως τέτοιο από το `s` στο τέλος).

8.4. Διαχείριση Αρχείων (Files)

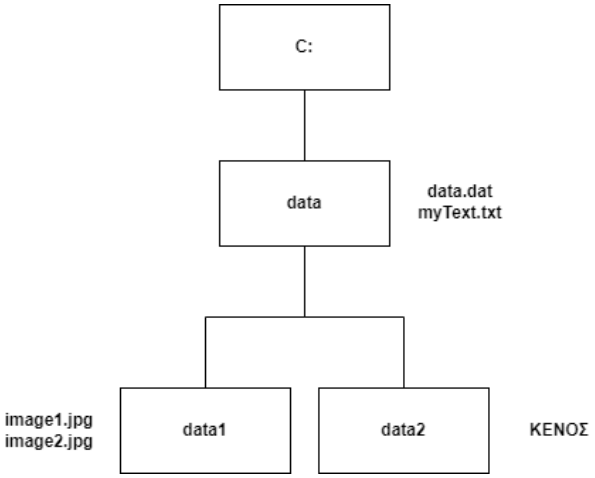
Στα προηγούμενα είδαμε πώς χρησιμοποιούμε τις παρεχόμενες κλάσεις του `java.io` για να δημιουργήσουμε ρεύματα από και προς αρχεία και να διαχειριστούμε το περιεχόμενό τους. Συχνά ωστόσο χρειαζόμαστε να ασχοληθούμε με αυτό καθαυτό το σύστημα των αρχείων, τη δομή των φακέλων (καταλόγων), τις ιδιότητες των ίδιων των αρχείων ή των καταλόγων. Για το σκοπό αυτό, μέσα στο πακέτο java.io υπάρχει η κλάση `File`. Στις επόμενες ενότητες θα δούμε τα κύρια χαρακτηριστικά της και δυνατότητες που μας δίνει.

8.4.1. Η Κλάση File

Η κλάση `File` του πακέτου `java.io` είναι μια από τις θεμελιώδεις κλάσεις για τη διαχείριση αρχείων και καταλόγων (φακέλων) στην Java. Είναι ιδιαίτερα χρήσιμη για εφαρμογές που χρειάζονται διαχείριση αρχείων ή οργανωμένων δομών δεδομένων στο τοπικό σύστημα αρχείων. Αντιπροσωπεύει ένα αρχείο ή έναν κατάλογο στο σύστημα αρχείων του συστήματος και προσφέρει διάφορες μεθόδους που χρησιμοποιούνται, για παράδειγμα, για τη δημιουργία, διαγραφή, μετονομασία, και διαχείριση αρχείων και φακέλων.

Η `File` δεν χειρίζεται απευθείας το περιεχόμενο των αρχείων (όπως είδαμε, για ανάγνωση και εγγραφή δεδομένων χρησιμοποιούνται άλλες κλάσεις, όπως π.χ. `FileInputStream`, `FileOutputStream`, `BufferedWriter`), αλλά παρέχει λειτουργίες, όπως έλεγχο ύπαρξης αρχείων, έλεγχο αν είναι αρχείο ή κατάλογος, ανάκτηση του μεγέθους αρχείου, ώρα, ημερομηνία, τη διαδρομή του στο σύστημα φακέλων (path), καθώς και αναζήτηση περιεχομένου μέσα σε καταλόγους.

Στη Java, ένας φάκελος (κατάλογος) αντιμετωπίζεται και αυτός ως αρχείο, ουσιαστικά ένας φάκελος είναι για την Java αρχείο, με μια πρόσθετη ιδιότητα τη λίστα των ονομάτων-αρχείων που περιέχει. Η κλάση `File` ενθυλακώνει ονόματα αρχείων, οπότε δημιουργώντας ένα αντικείμενό της πρακτικά διαθέτουμε αναφορά στο αρχείο του οποίου το όνομα έχουμε ενθυλακώσει. Παρέχει διάφορους κατασκευαστές με τους οποίους δημιουργούμε αντικείμενα.



Εικόνα 30 - ενδεικτική δομή συστήματος αρχείων

Στην Εικόνα 30 βλέπουμε τη δομή των φακέλων στην οποία αναφέρονται τα παραδείγματα που ακολουθούν (σε ορθογώνια οι φάκελοι, δίπλα τα περιεχόμενα αρχεία).

Κατασκευαστής	Παράδειγμα

`File(String pathname)`	`// Unix style file separator`
	`File f1 = new File("c:/data");`
	`//Windows style file separator, θέλει διπλό \`
	`File f2 = new File("c:\\data");`

`File(String parent, String child)`	`File f3 = new File("c:/data","data.dat");`
`File(File parent, String child)`	`File f4 = new File(f1, "data.dat");`

Μερικές από τις κυριότερες μεθόδους που περιέχει η κλάση `File` μπορούν να συνοψιστούν, ομαδοποιημένες ανά λειτουργικότητα, όπως παρακάτω:

Δημιουργία, Διαγραφή και Μετονομασία Αρχείων/Καταλόγων

- `boolean createNewFile()`: Δημιουργεί ένα νέο κενό αρχείο. Επιστρέφει true αν το αρχείο δεν υπάρχει και δημιουργήθηκε, διαφορετικά false
- `boolean mkdir()`: Δημιουργεί έναν νέο κατάλογο. Επιστρέφει true αν δημιουργήθηκε.
- `boolean delete()`: Διαγράφει το αρχείο ή τον κατάλογο. Επιστρέφει true αν διαγράφηκε.
- `boolean renameTo(File dest)`: Μετονομάζει το αρχείο ή τον κατάλογο. Επιστρέφει true αν η μετονομασία έγινε.

Ιδιότητες αρχείων και καταλόγων

- `boolean exists`(): Ελέγχει αν το αρχείο ή ο κατάλογος υπάρχει.
- `boolean isFile()`: Ελέγχει αν το αντικείμενο `File` είναι αρχείο.
- `boolean isDirectory()`: Ελέγχει αν το αντικείμενο `File` είναι κατάλογος.
- `long length()`: Επιστρέφει το μέγεθος του αρχείου σε bytes.

Πληροφορίες Διαδρομής και Ονόματος

- `String getName()`: Επιστρέφει το όνομα του αρχείου ή του καταλόγου.
- `String getAbsolutePath()`: Επιστρέφει την απόλυτη διαδρομή του αρχείου ή καταλόγου.

- `File getParentFile()`: Επιστρέφει τον γονικό κατάλογο ως αντικείμενο `File`.

Λίστα Περιεχομένων Καταλόγου

- `String[] list()`: Επιστρέφει τα ονόματα των αρχείων/καταλόγων που περιέχει ένας κατάλογος.
- `File[] listFiles()`: Επιστρέφει τα αρχεία/καταλόγους που περιέχει ένας κατάλογος ως αντικείμενα `File`.

8.4.2. Παράδειγμα Χρήσης της Κλάσης File

Στην παρούσα ενότητα θα δούμε ένα παράδειγμα χρήσης ορισμένων από τις παραπάνω μεθόδους. Ως υποκείμενη δομή του συστήματος αρχείου, θεωρήστε αυτήν που βλέπουμε στην Εικόνα 30.

```
import java.io.File;

public class Stream13 {
    public static void main(String[] args) {
        String path = "C:/data";  // φάκελος εκκίνησης
        File rootDir = new File(path);

        if (rootDir.exists() && rootDir.isDirectory()) {
            System.out.println("Ξεκινάμε από "+path +"\n"+"Η δομή φακέλων
και αρχείων είναι η εξής:");
            parseDirectory(rootDir, ""); //ξεκινάμε με καθόλου εσοχή
        } else {
            System.out.println("Δεν υπάρχει ο φάκελος που δώσατε: " +
path);
        }
    }

    private static void parseDirectory(File dir, String indent) {
        File[] files = dir.listFiles();

        if (files != null) {
            for (File file : files) {
                if (file.isDirectory()) {
                  // εμφάνιση ονόματος και καλούμε αναδρομικά για τους
φακέλους
                    System.out.println(indent    +    "[Φάκελος]    "    +
file.getName());
                    parseDirectory(file,  indent  +  "    "); //κάθε  φορά
αυξάνεται η εσοχή
                } else { //για αρχείο, εμφάνιση ονόματος
                    System.out.println(indent    +    "[Αρχείο]    "    +
file.getName());
                }
            }
        }
    }
}
```

Το αποτέλεσμα της εκτέλεσης του προγράμματος είναι αυτό που βλέπουμε στην Εικόνα 31. Αμέσως παρακάτω θα εξηγήσουμε τα κυριότερα σημεία του κώδικα.

```
Ξεκινάμε από C:/data
Η δομή φακέλων και αρχείων είναι η εξής:
[Αρχείο] data.dat
[Φάκελος] data1
  [Αρχείο] image1.jpg
  [Αρχείο] image2.JPG
[Φάκελος] data2
[Αρχείο] myText.txt
```

Εικόνα 31 - εμφάνιση δομής φακέλων και αρχείων με την κλάση Files

Στη μέθοδο `main()` δημιουργούμε ένα αντικείμενο `File` που είναι η ρίζα από όπου ξεκινάει η διάσχιση του συστήματος αρχείων. Ακολουθεί έλεγχος αν το όνομα που δώσαμε υπάρχει και επιπλέον αν είναι φάκελος. Σε αρνητική περίπτωση ο κώδικας τερματίζει, ενώ στη θετική καλείται η μέθοδος `parseDirectory()` που διασχίζει αναδρομικά τη δομή των φακέλων και εμφανίζει τα ονόματα φακέλων και αρχείων.

Η μέθοδος `parseDirectory()` δέχεται ως ορίσματα τον τρέχοντα φάκελο και ένα αλφαριθμητικό `indent` που περιέχει κενά και που σκοπό έχει να παρουσιάσει καλύτερα τη δομή φακέλων με τις κατάλληλες εσοχές (αυξανόμενες σε κάθε υποφάκελο). Στην αρχική κλήση της μεθόδου από την `main()` το `indent` είναι το κενό αλφαριθμητικό, αφού βρισκόμαστε στη ρίζα της δομής που θα διασχίσουμε.

Η μέθοδος `parseDirectory()`, σε κάθε κλήση της δημιουργεί έναν πίνακα από `Files` με τα περιεχόμενα του τρέχοντος φακέλου (αρχεία και υποφακέλους). Αν ο πίνακας δεν περιέχει στοιχεία, έχουμε τελειώσει. Διαφορετικά, κάθε ένα από τα περιεχόμενα του (τρέχοντος) πίνακα ελέγχεται αν είναι φάκελος ή αρχείο. Αν είναι αρχείο εμφανίζεται το λεκτικό [Αρχείο] και δίπλα το όνομά του. Αν είναι φάκελος, εμφανίζεται το λεκτικό [Φάκελος] και δίπλα το όνομά του και στη συνέχεια (αναδρομή) καλείται η `parseDirectory()` με όρισμα τον φάκελο αυτό και το `indent` αυξημένο (όσο ήταν συν άλλο ένα μήκος `indent`).

8.5. Διαχείριση FileSystem και Paths (java.nio & java.nio2 packages)

Ένα δομημένο και καλά οργανωμένο σύστημα όπως είναι αυτό της γλώσσας Java, βρίσκεται διαρκώς υπό αξιολόγηση και όποτε κρίνεται αναγκαίο γίνονται οι απαραίτητες προσθήκες και βελτιώσεις στις βιβλιοθήκες του. Έτσι, στις αρχικές εκδόσεις της, για την διαχείριση συστήματος αρχείων η γλώσσα προσέφερε την κλάση File του πακέτου `java.io`, την οποία είδαμε προηγουμένως. Στην πορεία εντοπίστηκαν προβλήματα και παραλείψεις στην υλοποίηση της κλάσης: για παράδειγμα, όπως είδαμε παραπάνω, οι μέθοδοι δημιουργίας, διαγραφής, μετονομασίας αντί να εγείρουν εξαίρεση επιστρέφουν τιμή `false` αν κάτι δεν πάει καλά, με αποτέλεσμα ο προγραμματιστής να μην μπορεί να ξέρει τι ακριβώς δεν πήγε καλά.

Έτσι, δημιουργήθηκε ένα νέο πακέτο το `java.nio` με πολλές βελτιώσεις και κύριο ζητούμενο την ταχύτητα εκτέλεσης. Εισήγαγε έννοιες όπως τα channels, τα buffers και την non-blocking I/O που χρησιμοποιούνται για την ανάγνωση και εγγραφή δεδομένων με τρόπο πιο αποδοτικό από ότι στην διαχείριση με τα byte streams που γνωρίσαμε παραπάνω. Ειδικότερα, τα δεδομένα διαβάζονται από ένα channel μέσα σε έναν buffer ή γράφονται από έναν buffer σε ένα channel. Επιπλέον, παρέχει είσοδο/έξοδο χωρίς μπλοκάρισμα του κώδικα. Για παράδειγμα, ο κώδικας μπορεί να

ζητήσει από ένα channel να διαβάσει δεδομένα μέσα σε έναν buffer και να κάνει κάτι άλλο μέχρι να ολοκληρωθεί το I/O, οπότε επιστρέφει για να συνεχίσει με την επεξεργασία τους.

Το java.nio2 είναι μεταγενέστερο, στηρίζεται και επεκτείνει το `java.nio` και συμπεριλαμβάνει μεθόδους που επικεντρώνονται στη διαχείριση αρχείων και φακέλων, φέρνοντας και εδώ σημαντικές βελτιώσεις σε σχέση με την κλάση `File`. Για να χρησιμοποιήσει κάποιος αυτές τις μεθόδους του java.nio2 θα πρέπει να κάνει την εισαγωγή τις κλάσεις του πακέτου `java.nio.file`.

Στα παραδείγματα που περιλαμβάνονται σε αυτή την ενότητα, αναφερόμαστε στη δομή αρχείων που χρησιμοποιήσαμε νωρίτερα και μπορείτε να δείτε στην Εικόνα 30.

8.5.1. Εισαγωγή στο package java.nio.file

Το `java.nio.file` προστέθηκε από την Java 7 στο πλαίσιο του java.nio2 και επικεντρώνεται στη διαχείριση του συστήματος αρχείων. Μεταξύ άλλων, προσφέρει δύο κύρια συστατικά για την επεξεργασία διαδρομών και αρχείων: τα `Path` και `Files`. Το `Path` αντικαθιστά την κλάση `File` στη διαχείριση διαδρομών και είναι πλήρως συμβατό με διαφορετικά συστήματα αρχείων, ενώ η `Files` περιλαμβάνει στατικές μεθόδους για λειτουργίες που αφορούν αρχεία και καταλόγους, όπως η ανάγνωση/εγγραφή δεδομένων και η διαχείριση μεταδεδομένων.

Τα κύρια χαρακτηριστικά του πακέτου `java.nio.file` σε σύγκριση με την κλάση `java.io.File` περιλαμβάνουν:

- Υποστήριξη πολλαπλών τύπων αρχείων και καταλόγων: Με το `Path` μπορούμε να αλληλοεπιδρούμε με διαφορετικά συστήματα αρχείων (π.χ. Windows, Linux, macOS) και εντός της ίδιας εφαρμογής.
- Διαχείριση συμβολικών συνδέσμων (symlinks): Οι `Path` και `Files` μπορούν να αναγνωρίσουν και να επεξεργαστούν συμβολικούς συνδέσμους, δυνατότητα που η `File` δεν υποστηρίζει.
- Βελτιωμένη Διαχείριση Εξαιρέσεων: Περιλαμβάνει περισσότερους τύπους εξαιρέσεων που βοηθούν στην αναγνώριση συγκεκριμένων σφαλμάτων του συστήματος αρχείων.

Το πακέτο `java.nio.file`, μπορεί να αντικαταστήσει την κλάση `java.io.File`, η οποία ωστόσο δεν καταργείται κυρίως λόγω συμβατότητας, και προσφέρει:

- Δημιουργία και διαγραφή αρχείων και καταλόγων.
- Πρόσβαση σε μεταδεδομένα, όπως ημερομηνίες τροποποίησης και δικαιώματα αρχείων.
- Χρήση streams για ανάγνωση και εγγραφή δεδομένων, κάτι που επιτρέπει την αποδοτικότερη διαχείριση μεγάλων αρχείων.
- Υποστήριξη αναδρομικών λειτουργιών για διαπέραση μεγάλων καταλόγων.

Στον μπορείτε να δείτε μια σύγκριση ανάμεσα στα πακέτα `java.io` και `java.nio`.

Πίνακας 3 - σύγκριση java.io με java.nio

Χαρακτηριστικό	*java.io*	*java.nio*
Σχεδίαση	Βασίζεται σε ροές (streams)	Βασίζεται σε buffers και κανάλια (channels)
Μοντέλο Επεξεργασίας	Blocking I/O (μπλοκαρισμένη είσοδος/έξοδος)	Non-blocking I/O (μη μπλοκαρισμένη είσοδος/έξοδος)
Αλληλεπίδραση	Κάθε λειτουργία εισόδου/εξόδου περιμένει να ολοκληρωθεί	Μπορεί να εκτελεί I/O χωρίς να περιμένει να ολοκληρωθεί
Απόδοση	Χαμηλότερη απόδοση για μεγάλους όγκους δεδομένων	Υψηλότερη απόδοση λόγω buffer-based λειτουργίας
Αντικείμενα Επεξεργασίας	InputStream, OutputStream, Reader, Writer	Buffer, Channel, Path, Files
Υποστήριξη Δικτύου	Υποστήριξη με sockets	Υποστήριξη non-blocking δικτύου μέσω SocketChannel
Διαχείριση Συστήματος Αρχείων	Λιγότερη ευελιξία και δυνατότητες	Πλούσιες δυνατότητες διαχείρισης με Path, Files
Κωδικοποιήσεις	Απλές επιλογές κωδικοποίησης	Υποστήριξη διαφορετικών κωδικοποιήσεων και charset
API Σύνταξη	Πιο απλή και εύκολη στη χρήση	Πιο περίπλοκη αλλά και πιο ευέλικτη
Πλεονεκτήματα	Καταλληλότερο για μικρά έργα και απλή χρήση	Κατάλληλο για μεγάλα έργα που απαιτούν απόδοση και ευελιξία

Τα κυριότερα στοιχεία του πακέτου, τα οποία θα αναπτύξουμε αμέσως, είναι το ζεύγος (`interface Path`, `class Paths`), το ζεύγος (`abstract class FileSystem`, `class FileSystems`) και η `class Files`.

8.5.2. Το interface java.nio.file.Path και η abstract class java.nio.file.FileSystem

Στο κέντρο της λειτουργικότητάς του java.nio.file βρίσκεται το `Interface Path` που είναι η συγκεκριμένη θέση μιας οντότητας (αρχείου ή φακέλου) σε ένα σύστημα αρχείων. Πρακτικά, είναι το γνωστό (απόλυτο ή σχετικό) μονοπάτι ή το (πλήρες ή σχετικό) όνομα αρχείου σε έναν δίσκο.

Προσέξτε ότι, μιλώντας για μονοπάτι/θέση οντότητας, έχουμε δύο είδη αναφοράς, το απόλυτο μονοπάτι (πλήρης διαδρομή από τη ρίζα του συστήματος αρχείων) και το σχετικό μονοπάτι που είναι η διαδρομή σχετικά με ένα άλλο μονοπάτι.

Το `interface Path` (ως Interface που είναι) δεν περιέχει κατασκευαστή, αλλά συμπληρώνεται από μια κλάση την `Paths` η οποία περιέχει την στατική μέθοδο `get()` που δέχεται όρισμα ένα `String` (ή περισσότερα που τα συνενώνει) και επιστρέφει ένα αντικείμενο τύπου `Path`.

Το `interface Path` περιλαμβάνει μια σειρά από μεθόδους με τις οποίες μπορεί κάποιος να διαχειριστεί το ίδιο το αντικείμενο `Path`. `Ενδεικτικά, να` διαβάσει το όνομα του αρχείου ή φακέλου, να δημιουργήσει ένα νέο αντικείμενο Path από ένα υπάρχον (ένας υποφάκελος, ένα

αρχείο μέσα σε φάκελο), να ελέγξει αν ένα μονοπάτι ξεκινά με κάποιο άλλο (π.χ. το folder1/folder2/folder3 ξεκινά με folder1/folder2), κ.λπ.

```
import java.nio.file.Path;
import java.nio.file.Paths;

public class Stream14 {
	public static void main(String[] args) {
		// 1. Αναπαράσταση του ριζικού καταλόγου C:\
		Path rootPath = Paths.get("C:/");
		System.out.println("1. Ρίζα: " + rootPath);
		// 2. Αναπαράσταση του υποφακέλου data στον C:\
		Path path1 = rootPath.resolve("data");
		System.out.println("2. Μονοπάτι του υποφακέλου: " + path1);
		// 3. Αναπαράσταση του αρχείου myText.txt μέσα στον φάκελο data
		Path path2 = path1.resolve("myText.txt");
		System.out.println("3. Αρχείο στον φάκελο data: " + path2);
		// 4. Ανάκτηση του γονικού φακέλου
		System.out.println("4. Γονικός φάκελος του αρχείου: " +
path2.getParent());
		// 5. Ανάκτηση του ονόματος αρχείου
		System.out.println("5. Όνομα του αρχείου: " +
path2.getFileName());
		// 6. Συνδυασμός διαδρομών για δημιουργία νέου μονοπατιού
		Path pathCombined =
path1.resolve("data1").resolve("image1.jpg");
		System.out.println("6. Συνδυασμένο μονοπάτι του image1.jpg: " +
pathCombined);
	}
}
```

Με τη μέθοδο `Paths.get()` δημιουργούμε ένα αντικείμενο `Path` που αναπαριστά μια διαδρομή, στην περίπτωσή μας μια απόλυτη διαδρομή στη ρίζα του δίσκου C:. Στη συνέχεια, με τη μέθοδο `resolve` της κλάσης `Path` κατασκευάζουμε νέα αντικείμενα `Path` που αναπαριστούν τον υποφάκελο `data` (#2), το αρχείο mytext.txt μέσα στον φάκελο `data` (#3). Μια άλλη μέθοδος της `Path` είναι η `getParent()` που μας επιστρέφει το γονέα του αντικειμένου που την καλεί (#4). Για οποιοδήποτε αντικείμενο `Path` η μέθοδος `getFileName()` επιστρέφει το όνομα του αρχείου που το αντικείμενο αυτό αναπαριστά (#5). Τέλος δείτε στην περίπτωση #6 όπου ξεκινάμε από το αντικείμενο `path1` (C:\data) και με διαδοχικά `resolve` κατασκευάζουμε το αντικείμενο `Path` που αναπαριστά τη διαδρομή C:\data\data1\image1.jpg.

Η εκτέλεση του παραπάνω κώδικα δίνει το αποτέλεσμα που βλέπουμε στην Εικόνα 32. Παρατηρήστε, ότι εμείς στον κώδικα χρησιμοποιούμε το Linux-style σύμβολο για διαχωρισμό των συστατικών του μονοπατιού (/) ενώ στην έξοδο βλέπουμε τον Windows-style διαχωριστή (\), καθώς ο κώδικας εκτελείται σε περιβάλλον Windows.

```
1. Ρίζα: C:\
2. Μονοπάτι του υποφακέλου: C:\data
3. Αρχείο στον φάκελο data: C:\data\myText.txt
4. Γονικός φάκελος του αρχείου: C:\data
5. Όνομα του αρχείου: myText.txt
6. Συνδυασμένο μονοπάτι του image1.jpg: C:\data\data1\image1.jpg
```

Εικόνα 32 - αποτέλεσμα χρήσης κλάσεων Path, Paths του java.nio.file

Ένα άλλο ζευγάρι κλάσεων που επίσης σχετίζεται με τη διαχείριση του συστήματος αρχείων είναι οι `FileSystem` και `FileSystems`. Η πρώτη είναι μια αφηρημένη (abstract) κλάση που χρησιμοποιείται για να αντιπροσωπεύει ένα συγκεκριμένο σύστημα αρχείων, η δεύτερη είναι μια βοηθητική κλάση (factory class) η οποία περιέχει στατικές μεθόδους δημιουργίας αντικειμένων `FileSystem`. Για παράδειγμα, η μέθοδος `FileSystems.getDefault()` επιστρέφει ένα αντικείμενο `FileSystem` με το τρέχον, εξ ορισμού σύστημα αρχείων της πλατφόρμας.

Η κλάση `FileSystem` περιέχει μεθόδους με τις οποίες μπορούμε να έχουμε πρόσβαση στα αρχεία και άλλα αντικείμενα ενός συστήματος αρχείων.

```
import java.nio.file.FileSystem;
import java.nio.file.FileSystems;
import java.nio.file.Path;

public class Stream15 {
	public static void main(String[] args) {
		// Απόκτηση του προεπιλεγμένου FileSystem μέσω της FileSystems
		FileSystem fileSystem = FileSystems.getDefault();
		System.out.println("Separator: " + fileSystem.getSeparator());
		// Δημιουργία ενός Path χρησιμοποιώντας το FileSystem
		Path path = fileSystem.getPath("C:/", "data", "myText.txt");
		System.out.println("Path: " + path);
	}
}
```

Με την μέθοδο `FileSystems.getDefault()` δημιουργούμε ένα αντικείμενο `FileSystem` που είναι το προεπιλεγμένο σύστημα αρχείων που χρησιμοποιεί η Java για να αλληλοεπιδρά με το τοπικό σύστημα αρχείων του υπολογιστή. Η μέθοδος `getSeparator()` της κλάσης `FileSystem` επιστρέφει τον διαχωριστή των ονομάτων σε ένα μονοπάτι που χρησιμοποιεί το σύστημά μας. Στα Windows είναι το backslash (\). Η μέθοδος `getPath()` της κλάσης `FileSystem` συνενώνει τα ορίσματα που της παρέχονται και δημιουργεί ένα αντικείμενο `Path`.

Η έξοδος του κώδικα φαίνεται στην Εικόνα 33.

```
Separator: \
Path: C:\data\myText.txt
```

Εικόνα 33 - αποτέλεσμα χρήσης μεθόδων κλάσεων FileSystem και FileSystems

Η κλάση `FileSystem` περιλαμβάνει και μια μέθοδο `close()` με την οποία κλείνουμε το σύστημα αρχείων που έχουμε δημιουργήσει με μια μέθοδο `newFileSystem()`. Δεν επιτρέπεται να κλείσουμε το εξ ορισμού σύστημα αρχείων και γι' αυτό δεν το κάναμε στον παραπάνω κώδικα. Αν δοκιμάσουμε να το κλείσουμε θα προκληθεί Εξαίρεση.

8.5.3. Η Κλάση Files

Η κλάση `Files` περιέχει στατικές μεθόδους και συνεργάζεται στενά με την `Path`, καθώς οι μέθοδοί της λειτουργούν πάνω σε αντικείμενα `Path`. Οι μέθοδοι αυτές επιτελούν λειτουργίες των οντοτήτων στις οποίες δείχνει το αντικείμενο. Ενδεικτικά, αντιγράφουν αρχεία, μετακινούν αρχεία, δημιουργούν φάκελο ή αρχείο, να ανακτούν το μέγεθος αρχείου, κ.λπ. Ειδικότερα:

- **Διαχείριση Αρχείων και Φακέλων**: Μέθοδοι όπως `createFile()`, `createDirectory()`, `delete()`, και `move()` παρέχουν δημιουργία, διαγραφή και μετακίνηση αρχείων και καταλόγων.
- **Ανάγνωση και Εγγραφή Δεδομένων**: Μέθοδοι όπως `readAllBytes()`, `readAllLines()`, `write()`, `newBufferedReader()`, `newBufferedWriter()` εξυπηρετούν για τη διαχείριση δεδομένων αρχείων σε bytes ή σε χαρακτήρες.
- **Διαχείριση Μεταδεδομένων**: Μέθοδοι όπως `getAttribute()`, `setAttribute()`, `isReadable()`, `isWritable()`, `size()` για ανάγνωση και τροποποίηση ιδιοτήτων αρχείων, όπως ημερομηνίες δημιουργίας, άδειες πρόσβασης κ.λπ.

```
import java.nio.file.*;

public class Stream16 {
	public static void main(String[] args) {
		Path root = Paths.get("c:/");
		Path target = root.resolve("data/data1/newFile.dat");

		// Δημιουργία αρχείου
		try {
			// Εγγραφή δεδομένων στο αρχείο
			Files.write(target, "Δοκιμάζω το πακέτο
java.nio.file!".getBytes());
			// Ανάγνωση δεδομένων από το αρχείο με readString (Java
11+)
			String content = Files.readString(target);
			System.out.println("Περιεχόμενα αρχείου: " + content);
			// Διαγραφή αρχείου
			Files.delete(target);
			System.out.println("Το αρχείο διαγράφηκε!");
		} catch (Exception e) {
			e.printStackTrace();
		}
	}
}
```

Αρχικά δημιουργούμε ένα αντικείμενο `Path` για τη ρίζα του συστήματος αρχείων και στη συνέχεια με τη μέθοδο `resolve()` ένα δεύτερο `Path` που αναπαριστά ένα νέο αρχείο με όνομα `newFile.dat` και το οποίο δεν υπάρχει στο φάκελο (Εικόνα 30). Θα μπορούσαμε απευθείας να δημιουργήσουμε μόνο το δεύτερο `Path`, με μια εντολή `Paths.get()`.

Χρησιμοποιούμε τη μέθοδο `Files.write()` για να γράψουμε το κείμενο στο αρχείο, αφού πρώτα με τη μέθοδο `String.getBytes()` το μετατρέψουμε σε ακολουθία από bytes. Η `write()` μπορεί να δεχτεί παραμέτρους που καθορίζουν τη συμπεριφορά της, στο πώς το αρχείο δημιουργείται ή ανοίγεται ανάλογα με το αν υπάρχει ή όχι. Εφόσον δεν υπάρχουν παράμετροι η `write()` δημιουργεί αν δεν υπάρχει ή καθαρίζει αν υπάρχει το αρχείο, στη συνέχεια γράφει τα bytes και όταν γραφτούν όλα, κλείνει το αρχείο.

Στη συνέχεια με τη μέθοδο `Files.readString()` διαβάζουμε το περιεχόμενο του αρχείου σε ένα `String` και το εμφανίζουμε στην οθόνη. Τέλος με τη μέθοδο `File.delete()` το αρχείο διαγράφεται.

Η μέθοδος `Files.readString()` προστέθηκε στην Java 11. Αν χρησιμοποιούμε κάποια παλαιότερη έκδοση, μπορούμε να χρησιμοποιήσουμε τη μέθοδο `Files.readAllLines()` η οποία επιστρέφει μια λίστα αλφαριθμητικών, όπου κάθε στοιχείο της λίστας είναι μια γραμμή του αρχείου. Στην περίπτωσή μας η λίστα θα έχει ένα μόνο στοιχείο.

Δύο ακόμα (από τις πολλές) μεθόδους της κλάσης `Files` είναι η boolean `Files.exists()` που ελέγχει αν ένα αρχείο υπάρχει, και η `Files.createFile()` που δημιουργεί ένα νέο, άδειο αρχείο, εφόσον δεν υπάρχει.

8.5.4. Παράδειγμα Χρήσης FileSystem, Path, και Files

Αφού είδαμε τα κυριότερα σημεία των σημαντικότερων κλάσεων του πακέτου `java.nio.file`, θα παρουσιάσουμε ένα συγκεντρωτικό παράδειγμα όπου χρησιμοποιούμε συνδυαστικά τις κλάσεις αυτές.

Ως περίπτωση χρήσης υποθέτουμε ότι ένα πρόγραμμα (π.χ. κατά την εγκατάστασή του) δημιουργεί έναν προσωρινό φάκελο στο δίσκο και μέσα εκεί ένα προσωρινό αρχείο. Αφού ολοκληρώσει τη διαδικασία, θα διαγράψει αρχείο και φάκελο, προτού τερματίσει.

Σημειώστε ότι ο κώδικας σαφώς και δεν είναι πλήρης ούτε καλύπτει όλες τις δυνατές περιπτώσεις, αλλά μείναμε σε σχετικά απλή διάταξη προκειμένου να επικεντρώσουμε στις λειτουργίες του `java.nio.file` και όχι στην συγγραφή ενός πλήρως λειτουργικού κώδικα. Για παράδειγμα, οι μέθοδοι κάνουν throw τις εξαιρέσεις και όλες αντιμετωπίζονται μαζικά από τη `main()` με μια γενική try-catch. Ο κώδικας θεωρεί δεδομένο ότι υπάρχει στο σύστημα δίσκος C:/ και σε κάθε περίπτωση προβλήματος με τις εντολές διαχείρισης των αρχείων, το πρόγραμμα απλά καλεί μια μέθοδο `error()` και τερματίζει.

```
import java.nio.file.*;
import java.nio.file.attribute.BasicFileAttributes;
import java.util.List;

public class Stream17 {
	/*
	 * 1. Δημιουργία ενός FileSystem και ενός Path για τον προσωρινό
κατάλογο
	 * εργασίας
	 */
	public static Path createTempFolder(String root) throws Exception {
		FileSystem fileSystem = FileSystems.getDefault();
		Path tempDir = fileSystem.getPath(root, "tmp00x");
		if (!Files.exists(tempDir)) {
			Files.createDirectory(tempDir);
			return tempDir;
		}
		return null;
	}

	/*
	 * 2. Δημιουργία ενός αρχείου μέσα στον κατάλογο εργασίας
	 */
	public static Path createTempFile(Path tempDir) throws Exception {
		Path filePath = tempDir.resolve("tempData.txt");
		if (!Files.exists(filePath)) {
```

```
                Files.createFile(filePath);
                return filePath;
            }
            return null;
      }

      /*
       * Βοηθητική μέθοδος, καλείται στη main σε περίπτωση επιστροφής null
από τις προηγούμενες μεθόδους
       */
      public static void error() {
            System.out.println("Error! Exiting...");
            System.exit(1);
      }

      /*
       * 3. Εγγραφή κειμένου στο αρχείο με κωδικοποίηση UTF-8
       */
      public static void writeTempFile(Path filePath, String content)
throws Exception {
            Files.write(filePath, content.getBytes("UTF-8"));
      }

      /*
       * 4. Ανάγνωση κειμένου από το αρχείο, εξ ορισμού UTF-8
       */
      public static List<String> retrieveText(Path filePath) throws
Exception {
            List<String> lines = Files.readAllLines(filePath); // default
UTF-8
            return lines;
      }

      /*
       * 5. Έλεγχος ύπαρξης αρχείου, ανάκτηση και εμφάνιση μεταδεδομένων
του
       */
      public static void displayMetadata(Path filePath) throws Exception {
            if (Files.exists(filePath)) {
                System.out.printf("File                                    %s
exists.\n".formatted(filePath));
                BasicFileAttributes                attrs                =
Files.readAttributes(filePath, BasicFileAttributes.class);
                System.out.println("Creation       time:       "       +
attrs.creationTime());
                System.out.println("Size: " + attrs.size() + " bytes");
                System.out.println("Is       directory:       "       +
attrs.isDirectory());
            } else
                System.out.println("File not found");
      }

      /*
       * 6. Διαγραφή του αρχείου και του γονικού καταλόγου
       */
      public static void deleteTemporary(Path filePath, Path tempDir)
throws Exception {
            Files.delete(filePath);
            Files.delete(tempDir);
```

```
            System.out.println("Διαγράφτηκε το αρχείο και ο γονικός
φάκελος.");
      }
/*
       * 7. Η μέθοδος main()
       */

      public static void main(String[] args) {
            Path tempDir, filePath;
            try {
                  // 1
                  tempDir = createTempFolder("C:/");
                  if (tempDir == null)
                        error();
                  // 2
                  filePath = createTempFile(tempDir);
                  if (filePath == null)
                        error();
                  // 3
                  writeTempFile(filePath,    "Δοκιμαστικό    κείμενο    στο
προσωρινό αρχείο " + filePath);
                  // 4
                  System.out.println("Περιεχόμενα του αρχείου:");
                  for (String line : retrieveText(filePath))
                        System.out.println(line);
                  // 5
                  displayMetadata(filePath);
                  // 6
                  deleteTemporary(filePath, tempDir);
            } catch (Exception e) {
                  e.printStackTrace();
            }
      }
}
```

Όπως είπαμε, ο κώδικας δημιουργεί και διαχειρίζεται προσωρινά αρχεία και φακέλους στο σύστημα αρχείων, διατρέχοντας όλα τα βήματα από τη δημιουργία έως τη διαγραφή. Αναλυτικά τα βήματα και οι λειτουργίες:

1. **Δημιουργία Προσωρινού Καταλόγου**
    - Η μέθοδος `createTempFolder()` δημιουργεί ένα `FileSystem` και ένα `Path` για έναν προσωρινό φάκελο με το όνομα "tmp00x" στον κατάλογο `root` (παραδοχή, ο C:/).
    - Ελέγχει αν υπάρχει ο φάκελος και εάν δεν υπάρχει ήδη, τον δημιουργεί.
2. **Δημιουργία Προσωρινού Αρχείου**
    - Η `createTempFile()` δημιουργεί ένα αρχείο "tempData.txt" μέσα στον προσωρινό φάκελο.
    - Χρησιμοποιεί τη μέθοδο `resolve()` του `Path` για να καθορίσει τη διαδρομή του αρχείου μέσα στον κατάλογο εργασίας.
    - Αν το αρχείο υπάρχει ήδη, η μέθοδος επιστρέφει `null`.
3. **Εγγραφή σε Προσωρινό Αρχείο**

- Η `writeTempFile()` γράφει μια συμβολοσειρά κειμένου (`content`) στο αρχείο, χρησιμοποιώντας την κωδικοποίηση UTF-8.

4. **Ανάγνωση Κειμένου από Αρχείο**
   - Η `retrieveText()` διαβάζει το περιεχόμενο του αρχείου ως λίστα από γραμμές, υποθέτοντας κωδικοποίηση UTF-8 εξ ορισμού.
5. **Εμφάνιση Μεταδεδομένων του Αρχείου**
   - Η `displayMetadata()` ελέγχει αν το αρχείο υπάρχει, και αν ναι, εκτυπώνει πληροφορίες μεταδεδομένων όπως χρόνος δημιουργίας, μέγεθος αρχείου, και αν είναι φάκελος.
   - Χρησιμοποιεί τις `Files.readAttributes()` και `BasicFileAttributes` για την ανάκτηση των μεταδεδομένων (δείτε σχετικά και στο java API documentation).
6. **Διαγραφή Προσωρινού Αρχείου και Καταλόγου**
   - Η `deleteTemporary()` διαγράφει το αρχείο και τον φάκελο που το περιέχει, καθαρίζοντας τα προσωρινά δεδομένα.
7. **Εκτέλεση στη Μέθοδο main**
   - Οι παραπάνω μέθοδοι καλούνται διαδοχικά μέσα στη `main`.
   - Αν οποιαδήποτε μέθοδος επιστρέψει `null`, καλείται η `error()`, η οποία τερματίζει την εκτέλεση με μήνυμα λάθους.
   - Τέλος, εμφανίζονται το περιεχόμενο και τα μεταδεδομένα του αρχείου πριν από τη διαγραφή του.

8.6. Serialization στην Java (java.io.Serializable Interface)

Στο παράδειγμα που είδαμε στην υποενότητα 8.3.3.1, αποθηκεύσαμε σε ένα αρχείο κειμένου τα δεδομένα των πεδίων ενός στιγμιότυπου της κλάσης `Student`, και στη συνέχεια ανακτήσαμε. Από τη διαδικασία που περιγράψαμε, έγινε φανερό ότι σε πολλές περιπτώσεις έπρεπε να γράψουμε ειδικό κώδικα για να διαχειριστούμε διαφορετικά κάθε πεδίο ανάλογα με τον τύπο του, όπως π.χ. μετατροπή σε ακέραιο, διαχείριση του πίνακα αλφαριθμητικών, κ.λπ. Η σειριοποίηση (Serialization) έρχεται να μας βοηθήσει και να απλουστεύει τη διαδικασία, προσφέροντας και επιπλέον πλεονεκτήματα.

Η σειριοποίηση είναι ένας μηχανισμός της Java με τον οποίο τα αντικείμενα μπορούν να μετατραπούν σε ρεύμα από bytes που μπορεί να μεταδοθεί μέσα από δίκτυο ή να αποθηκευτεί σε αρχείο. Το ρεύμα μπορεί να ανακατασκευαστεί (αποσειριοποίηση) και έτσι να δημιουργηθεί ξανά το αντίγραφο του αντικειμένου.

8.6.1. Η Έννοια του Serialization

Μένοντας στο παραπάνω παράδειγμα (στην υποενότητα 8.3.3.1), ας δούμε πώς θα μπορούσαμε να χρησιμοποιήσουμε το Serialization για να πετύχουμε παρόμοιο αποτέλεσμα. Θα χρησιμοποιήσουμε τις μεθόδους `java.io.ObjectOutputStream` και `java.io.ObjectInputStream` που απλά αναφέραμε στην ενότητα 8.3.2. Δείτε εκεί και την Εικόνα 17.

```
import java.io.*;
import java.util.Arrays;

@SuppressWarnings("serial")
class Student implements Serializable {
	private String name;
	private int age;
	private String[] subjects;

	public Student(String name, int age, String[] subjects) {
		this.name = name;
		this.age = age;
		this.subjects = subjects;
	}

	// Αποθήκευση αντικειμένου με σειριοποίηση
	public void saveToBinaryFile(String fileName) throws Exception {
		try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream(fileName))) {
			oos.writeObject(this);
		}
	}

	// Ανάκτηση αντικειμένου με απο-σειριοποίηση
	public static Student readFromBinaryFile(String fileName) throws
Exception {
		try (ObjectInputStream ois = new ObjectInputStream(new
FileInputStream(fileName))) {
			return (Student) ois.readObject();
		}
	}

	@Override
	public String toString() {
		return "Student{name='" + name + "', age=" + age + ",
subjects=" + Arrays.toString(subjects) + "}";
	}
}

public class MainStudentSer {
	public static void main(String[] args) throws Exception {
		String fileName = "student.ser";
		Student student = new Student("John Smith", 20, new String[] {
"Math", "Science", "History" });
		// Αποθήκευση με σειριοποίηση
		student.saveToBinaryFile(fileName);
		// Ανάκτηση με απο-σειριοποίηση
		Student loadedStudent = Student.readFromBinaryFile(fileName);
		System.out.println("Loaded Student: " + loadedStudent);
	}
}
```

Και εδώ έχουμε συμπεριλάβει τόσο την κλάση `Student` όσο και την κλάση-οδηγό (με την `main`) στο ίδιο αρχείο. Όλες οι μέθοδοι κάνουν throw κάθε `Exception` απλά για να μείνουμε στα σημεία της έννοιας της σειριοποίησης και από-σειριοποίησης.

Επίσης, για λίγο, αγνοήστε την οδηγία προς τον μεταγλωττιστή `@SuppressWarnings("serial")` που του ζητά να αγνοεί προειδοποιήσεις σχετικά με το σειριακό αριθμό που πρέπει κανονικά να έχει μια κλάση που σειριοποιείται. Θα δούμε παρακάτω, σχετικά.

Ας επικεντρώσουμε στην κλάση `Student`. Για να μπορέσουμε να χρησιμοποιήσουμε σειριοποίηση η κλάση αυτή θα πρέπει να υλοποιεί το `interface Serializable`. Η δήλωση σηματοδοτεί ότι τα αντικείμενα της κλάσης μπορούν να σειριοποιηθούν με ασφάλεια σε ένα ρεύμα από bytes, επομένως δεν χρειάζεται να κάνουμε κάτι παραπάνω, πέραν της δήλωσης `implements Serializable`. Η κλάση δηλώνει τα πεδία, τον κατασκευαστή της και την `toString()` όπως και προηγουμένως. Όμως τώρα έχουμε αντικαταστήσει τις μεθόδους αποθήκευσης και ανάκτησης με τις μεθόδους `saveToBinaryFile()` και `readFromBinaryFile()`. Έχουμε ήδη εξηγήσει παραπάνω το λόγο που η δεύτερη δηλώνεται ως `static`. Αυτές οι μέθοδοι, αντίστοιχα δημιουργούν:

- ένα ρεύμα εξόδου `ObjectOutputStream` και με τη μέθοδο `writeObject()` γράφουν το αντικείμενο στο αρχείο, και
- ένα ρεύμα εισόδου `ObjectInpuputStream` και με τη μέθοδο `readObject()` διαβάζουν το αντικείμενο από το αρχείο.

Η μέθοδος `main()` δημιουργεί και αρχικοποιεί ένα αντικείμενο `Student` και χρησιμοποιεί τις δύο αυτές μεθόδους για αποθήκευση και στη συνέχεια ανάκτηση.

Όπως μπορούμε εύκολα να διαπιστώσουμε, ο κώδικας είναι πολύ απλούστερος, καθώς η Java χειρίζεται αυτόματα όλα τα πεδία, χωρίς να χρειάζεται να ασχοληθούμε με λεπτομέρειες όπως π.χ. ο τύπος του καθενός και η διαχείρισή του.

Κατά τη σειριοποίηση γίνεται διάσχιση των αντικειμένων μέσω των πεδίων τους, μια αναδρομική διαδικασία που ακολουθεί τον Γράφο των αντικειμένων ώστε να συμπεριληφθούν στη διαδικασία σειριοποίησης όλα τα προσβάσιμα αντικείμενα από την αρχική πηγή. Η διαδικασία αφορά τα πεδία που δεν είναι `static` καθώς και δεν έχουν δηλωθεί με τη λέξη-κλειδί `transient`. Στην περίπτωση που κάποιο πεδίο είναι αντικείμενο (π.χ. μια κλάση περιέχει ως πεδίο αντικείμενο μιας άλλης κλάσης), η διαδικασία εφαρμόζεται αναδρομικά σε αυτό το αντικείμενο. Για να γράψουμε το αποτέλεσμα στο ρεύμα χρησιμοποιούμε τη μέθοδο `writeObject()` και για να διαβάσουμε από το ρεύμα χρησιμοποιούμε τη μέθοδο `readObject()`.

Παρόλο που οι πλέον σύγχρονες τάσεις μετακίνησης δεδομένων πηγαίνουν προς τη χρήση άλλων τρόπων όπως, για παράδειγμα, αρχεία json και xml, η σειριοποίηση εξακολουθεί να βρίσκει σημαντική εφαρμογή σε περιπτώσεις όπως:

- Αποθήκευση κατάστασης σε αρχείο: μια εφαρμογή αποθηκεύει τις προτιμήσεις του χρήστη για να τις ανακτήσει αργότερα.
- Δημιουργία αντιγράφων ασφαλείας: αποθηκεύεται η κατάσταση των αντικειμένων μιας εφαρμογής ώστε σε περίπτωση σφάλματος να μπορεί να ανακτηθεί.
- Αποθήκευση σε βάση δεδομένων: ένα αντικείμενο σειριοποιείται και αποθηκεύεται σε μια ΒΔ σε πεδίο τύπου BLOB (Binary Large Oject).
- Μεταφορά δεδομένων μέσω δικτύου: σε ένα κατανεμημένο σύστημα (π.χ. Apache Spark) συμμετέχουν πολλοί workers που επιλύουν μέρος του προβλήματος στη λογική του «διαίρει και βασίλευε». Επίσης κατανεμημένα βρίσκονται και τα δεδομένα (φανταστείτε πολλούς κόμβους, π.χ. πολλούς υπολογιστές). Είναι συχνή η ανάγκη μεταφοράς δεδομένων

ανάμεσα στους κόμβους αλλά και μεταφοράς του κώδικα στους κόμβους προς εκτέλεση. Η σειριοποίηση γίνεται στον αποστολέα και η αποσειριοποίηση στον παραλήπτη.

8.6.2. Κύριες Έννοιες του Serialization

Έχοντας ήδη δει ένα παράδειγμα serialization/deserialization θα κάνουμε μια σύντομη παρουσίαση των κυριότερων εννοιών που συνδέονται με αυτήν την τεχνική που θα μας βοηθήσει να συνοψίσουμε όσα είπαμε παραπάνω αλλά και να επεκτείνουμε την σχετική με τη διαδικασία αυτή:

Serializable interface: Η βασική προϋπόθεση για τη σειριοποίηση μιας κλάσης είναι να υλοποιεί το Serializable interface. Αυτό δηλώνει στον μεταγλωττιστή ότι η κλάση και τα αντικείμενά της μπορούν να μετατραπούν με ασφάλεια σε byte stream. Το interface δεν έχει μεθόδους ή άλλα πεδία με τιμές οπότε το μόνο που πρέπει να κάνουμε είναι η δήλωση `implements Serializable` κατά τον ορισμό της κλάσης (ένα interface αυτού του τύπου ονομάζεται marker interface και η Java έχει και άλλα, π.χ. `Cloneable`).

Δήλωση πεδίων transient: Ένα πεδίο που στη δήλωσή του χρησιμοποιούμε τη λέξη-κλειδί `transient` δεν περιλαμβάνεται στη σειριοποίηση.

serialVersionUID: Κάθε σειριοποιήσιμη κλάση πρέπει να έχει ένα `serialVersionUID`, το οποίο είναι ένας μοναδικός αριθμός που χρησιμοποιείται για τον έλεγχο εκδόσεων. Αν δεν οριστεί, η JVM το δημιουργεί αυτόματα, αλλά είναι προτιμότερο να το ορίζουμε και να το ελέγχουμε εμείς για λόγους που θα συζητήσουμε παρακάτω.

ObjectOutputStream και ObjectInputStream: Αυτές οι κλάσεις χρησιμοποιούνται για τη σειριοποίηση και αποσειριοποίηση αντικειμένων, αντίστοιχα.

- Η `ObjectOutputStream` γράφει το αντικείμενο σε byte stream, και μπορεί να το αποθηκεύσει σε ένα αρχείο ή να το στείλει μέσω δικτύου.
- Η `ObjectInputStream` διαβάζει τον byte stream και ανακατασκευάζει το αντικείμενο.

Deserialization (Αποσειριοποίηση): Η διαδικασία κατά την οποία ο byte stream διαβάζεται και ανασυγκροτείται το αρχικό αντικείμενο. Για μπορεί να γίνει η αποσειριοποίηση, πρέπει η κλάση να είναι διαθέσιμη στο περιβάλλον όπου γίνεται η ανάγνωση, και το `serialVersionUID` κλάσης και σειριοποιημένου αντικειμένου να είναι συμβατά.

Προσαρμοσμένη σειριοποίηση (Custom Serialization): Μπορούμε να ελέγχουμε τη διαδικασία σειριοποίησης αν υλοποιήσουμε τις μεθόδους `writeObject` και `readObject`. Για παράδειγμα, μπορεί κάποια πεδία να χρειάζονται επεξεργασία πριν την σειριοποίηση. Θα την δούμε παρακάτω.

Εξαίρεση NotSerializableException: Αυτή η εξαίρεση εμφανίζεται όταν ένα αντικείμενο περιέχει πεδία τύπων που δεν υλοποιούν το `Serializable interface` και δεν έχουν δηλωθεί `transient`.

8.6.3. Transient Πεδία και Serial Version UID - Δημιουργία Serializable Κλάσης

Κάθε κλάση που υλοποιεί το `interface Serializable` πρέπει να περιέχει ένα πεδίο με όνομα **`serialVersionUID`**. Αν δεν το δηλώσουμε εμείς, τότε δημιουργεί αυτόματα ένα η JVM κατά την

λειτουργία της σειριοποίησης λαμβάνοντας υπόψη στοιχεία της κλάσης, όπως τα ονόματα και οι τύποι των πεδίων, οι υπογραφές των μεθόδων και κατασκευαστών, κ.λπ. Συνιστάται ιδιαίτερα το πεδίο να δηλώνεται ρητά από τον προγραμματιστή προκειμένου να ελέγχει τις εκδόσεις. Τυπικά είναι δηλωμένο ως `private static final long`, με το `private` να μην είναι υποχρεωτικό αλλά καλή πρακτική.

Το `serialVersionUID` που ορίζουμε στην κλάση ενσωματώνεται στον byte code κατά την σειριοποίηση. Στην αποσειριοποίηση ελέγχεται αν το `serialVersionUID` της κλάσης είναι ίδιο με το `serialVersionUID` που είναι αποθηκευμένο στον byte code. Αν ταιριάζουν, τότε η JVM γνωρίζει την κλάση από την οποία προέκυψε το byte stream και προχωρά στην ανασυγκρότηση του αντικειμένου. Διαφορετικά εγείρεται η εξαίρεση `InvalidClassException`.

Από το παραπάνω γίνεται αντιληπτό πως αν στηριχθούμε στην αυτόματη δημιουργία `serialVersionUID`, τυχόν μικρές αλλαγές στην κλάση (π.χ., αλλαγή ονόματος ή τύπου σε ένα πεδίο) μπορεί να προκαλέσουν αλλαγή στο παραγόμενο `serialVersionUID`, με αποτέλεσμα τη μη συμβατότητα.

Τα `transient` πεδία στη Java είναι εκείνα που εξαιρούνται από τη διαδικασία της σειριοποίησης. Όταν δηλώσουμε ένα πεδίο ως `transient` μέσα σε μια κλάση που υλοποιεί το Serializable interface, αυτό το πεδίο δεν θα αποθηκευτεί στο byte stream κατά τη σειριοποίηση. Έτσι, κατά την αποσειριοποίηση (deserialization), το `transient` πεδίο δεν θα διατηρήσει την αρχική του τιμή, αλλά θα επαναφερθεί στην προεπιλεγμένη τιμή του τύπου του (π.χ., `null` για αντικείμενα, 0 για ακέραιους, `false` για `boolean`).

Αυτό είναι χρήσιμο για πεδία που δεν θέλουμε να αποθηκευτούν όπως ένας κωδικός σύνδεσης, ή που δεν έχει νόημα να αποθηκευτούν όπως ένας σύνδεσμος σε αρχείο ή βάση δεδομένων (που θα αλλάξει). Ακόμα, δεν μπορούν να σειριοποιηθούν αντικείμενα που η κλάση τους δεν υλοποιεί το `Serializable`.

Επίσης, εξ ορισμού, τα `static` πεδία της κλάσης εξαιρούνται από τη σειριοποίηση καθώς ανήκουν στην κλάση και όχι στο αντικείμενο. Κατά την αποσειριοποίηση οι `static` μεταβλητές παίρνουν τιμή κατά τη δημιουργία της κλάσης.

Για να δούμε ένα παράδειγμα, θεωρήστε μια κλάση `User` που παριστάνει τους χρήστες ενός πληροφοριακού συστήματος. Περιέχει μια στατική μεταβλητή που αποθηκεύει το όνομα της εταιρείας που διαθέτει το πληροφοριακό σύστημα, το όνομα χρήστη και τον κωδικό σύνδεσης. Το πεδίο `password` (κωδικός σύνδεσης) είναι δηλωμένο ως `transient`. Επίσης, η κλάση περιέχει μεθόδους getters για όλα τα πεδία και δύο μεθόδους `serialize()` και `deserialize()` που την σειριοποιούν και την αποσειριοποιούν.

```
public class User implements Serializable {
	private static final long serialVersionUID = 1L;
	public static String systemName = "MyCompany System";
	private String userName;
	private transient String password; // δεν θα αποθηκευτεί κατά τη
σειριοποίηση

	public User(String userName, String password) {
		this.userName = userName;
		this.password = password;
```

```
	}

	public String getPassword() {
		return password;
	}

	public String getUserName() {
		return userName;
	}

	public String getSystemName() {
		return systemName;
	}

	public void serialize(String fileName) throws Exception {
		try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream(fileName))) {
			oos.writeObject(this);
		}
	}

	public static User deserialize(String fileName) throws Exception {
		try (ObjectInputStream ois = new ObjectInputStream(new
FileInputStream(fileName))) {
			return (User) ois.readObject();
		}
	}
}
```

Οι δύο κλάσεις που ακολουθούν αφορούν σε δύο προγράμματα.

Το πρώτο αρχικοποιεί ένα στιγμιότυπο της κλάσης, το σειριοποιεί και το αποθηκεύει στο δίσκο. Παρατηρήστε ότι πριν τη δημιουργία του αντικειμένου της `User` αλλάζουμε την τιμή της `static` μεταβλητής της κλάσης.

```
public class Stream20a {
	public static void main(String[] args) throws Exception {
		String fileName = "user.dat";
		User.systemName = "New Company";
		User u = new User("John Doe", "password");
		u.serialize(fileName);
	}
}
```

Το δεύτερο διαβάζει το αρχείο και ανακατασκευάζει το αντικείμενο.

```
public class Stream20b {
	public static void main(String[] args) throws Exception {
		String fileName = "user.dat";
		User inu = User.deserialize(fileName);
		System.out.println("username = " + inu.getUserName());
		System.out.println("static systemName = " +
inu.getSystemName());
		System.out.println("password = " + inu.getPassword());
	}
}
```

Το αποτέλεσμα της εκτέλεσης το βλέπουμε στην Εικόνα 34 .

```
username = John Doe
static systemName = MyCompany System
password = null
```

Εικόνα 34 - αποτέλεσμα αποσειριοποίησης

Παρατηρήστε ότι το όνομα χρήστη αποσειριοποιήθηκε κατά τα αναμενόμενα. Το `static` πεδίο δεν έχει την αλλαγμένη τιμή που είχαμε δώσει αλλά την αρχική της κλάσης (δεν σειριοποιήθηκε, πήρε την αρχική τιμή). Τέλος, το `password` που είναι δηλωμένο ως `transient`, επίσης δεν σειριοποιήθηκε και στην αποσειριοποίηση πήρε τιμή `null` (ως `String` που είναι).

8.6.4. Προσαρμοσμένη Serialization/Deserialization

Με την υλοποίηση custom serialization μπορούμε να επηρεάσουμε **ποια** δεδομένα αποθηκεύονται και **πώς** αυτά αποθηκεύονται. Για να το πετύχουμε αυτό, ορίζουμε/εξειδικεύουμε στον κώδικά μας τις μεθόδους `writeObject()` και `readObject()`. Αν δεν τις ορίσουμε και τις χρησιμοποιήσουμε απλά, αυτές εκτελούν την προεπιλεγμένη (εξ ορισμού) διαδικασία (όλα τα μη-transient και μη-στατικά πεδία σειριοποιούνται αυτόματα), όπως είδαμε στα προηγούμενα παραδείγματα. Αν τις ορίσουμε μπορούμε να τροποποιήσουμε τη λειτουργία τους.

Σκεφτείτε την εξής περίπτωση: έστω ότι στην κλάση `Student` υπάρχει ένα επιπλέον πεδίο, για παράδειγμα το ΑΦΜ του μαθητή. Για λόγους προστασίας προσωπικών δεδομένων που μπορεί να υποκλαπούν, θέλουμε κατά τη σειριοποίηση να αποθηκεύουμε το ΑΦΜ αφού πρώτα το κρυπτογραφήσουμε, ενώ τα υπόλοιπα πεδία θα υποστούν την συνηθισμένη, εξ ορισμού σειριοποίηση. Αυτό γίνεται ως εξής:

Θα επανα-ορίσουμε τις μεθόδους `writeObject()` και `readObject()` ώστε η πρώτη να σειριοποιεί «κανονικά» τα υπόλοιπα δεδομένα ενώ θα σειριοποιεί το ΑΦΜ αφού πρώτα το κρυπτογραφήσει.

- Για το σκοπό αυτό, στην τροποποιημένη `writeObject()` χρησιμοποιούμε αρχικά τη μέθοδο `defaultWriteObject()` για τα υπόλοιπα πεδία και στη συνέχεια τη `writeObject()` για το ΑΦΜ μετά από την κρυπτογράφηση.
- Ανάλογα πράττουμε και στην ανάγνωση, δηλαδή τροποποιούμε την `readObject()` που διαβάζει «κανονικά» τα υπόλοιπα δεδομένα με τη μέθοδο `defaultReadObject()` και ξεχωριστά το ΑΦΜ που το αποκρυπτογραφεί.

Για λόγους απλότητας αφενός έχουμε κάνει throw όλες τις Εξαιρέσεις, ενώ ως «κρυπτογράφηση» απλά αντιστρέφουμε τη σειρά των χαρακτήρων του ΑΦΜ.

```
import java.io.*;
import java.util.Arrays;

class Student implements Serializable {
    private static final long serialVersionUID = 1L;
    private String name;
    private int age;
    private transient String AFM; // Δεν θέλουμε να αποθηκευτεί ως έχει
    private String[] subjects;
```

```
      public Student(String name, int age, String AFM, String[] subjects) {
            this.name = name;
            this.age = age;
            this.AFM = AFM;
            this.subjects = subjects;
      }

    // Custom Serialization μέθοδος
    private void writeObject(ObjectOutputStream out) throws IOException {
        out.defaultWriteObject(); // Σειριοποιεί τα τυπικά πεδία
        out.writeObject(encrypt(AFM)); // Κρυπτογραφούμε το AFM πριν το
γράψουμε
    }

    // Custom Deserialization μέθοδος
    private void readObject(ObjectInputStream in) throws IOException,
ClassNotFoundException {
        in.defaultReadObject(); // Αποσειριοποιεί τα τυπικά πεδία
        AFM = decrypt((String) in.readObject()); // Αποκρυπτογραφούμε το
SSN
    }

 // Αποθήκευση αντικειμένου με σειριοποίηση
      public void saveToBinaryFile(String fileName) throws Exception {
            try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream(fileName))) {
                  oos.writeObject(this);
            }
      }

      // Ανάκτηση αντικειμένου με απο-σειριοποίηση
      public static Student readFromBinaryFile(String fileName) throws
Exception {
            try (ObjectInputStream ois = new ObjectInputStream(new
FileInputStream(fileName))) {
                  return (Student) ois.readObject();
            }
      }
 // "Κρυπτογράφηση" για παράδειγμα (αντιστρέφει το αλφαριθμητικό)
    private String encrypt(String data) {
        return new StringBuilder(data).reverse().toString();
    }
 // "Αποκρυπτογράφηση" για το παράδειγμα
    private String decrypt(String data) {
        return new StringBuilder(data).reverse().toString();
    }

    @Override
    public String toString() {
      return "Student{name=" + name + ", age=" + age + ", AFM=" + AFM + ",
subjects=" + Arrays.toString(subjects) + "}";
    }
}
```

Η μέθοδος `main()` χρησιμοποιεί τις μεθόδους `saveToBinaryFile()` και `readFromBinaryFile()` για να σειριοποιήσει και αποσειριοποιήσει το αντικείμενο. Αυτές κάνουν χρήση των τροποποιημένων `writeObject()` και `readObject()` που επίσης περιλαμβάνονται στην κλάση. Μέσα σε αυτές παρατηρήστε τη χρήση των μεθόδων `defaultWriteObject()` και `defaultReadObject()`. Μόνον έτσι μπορούν να χρησιμοποιηθούν, καθώς είναι διαθέσιμες μόνο

όταν καλούνται μέσα σε `writeObject/readObject` και δεν μπορούν να χρησιμοποιηθούν οπουδήποτε αλλού στην κλάση.

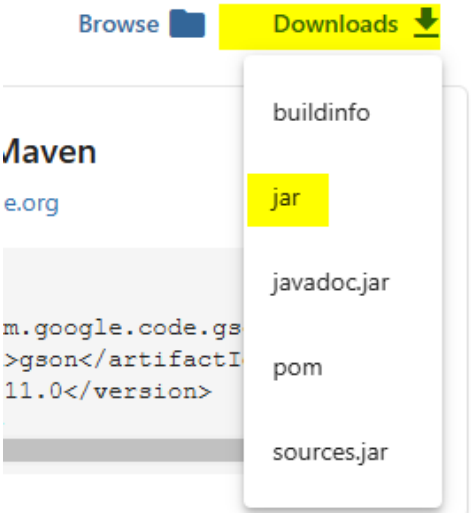
```
public class Main {
	public static void main(String[] args) throws Exception{
		String fileName = "student.ser";
		Student student = new Student("John Smith", 20, "123456789",new
String[] { "Math", "Science", "History" });
		// Αποθήκευση σε αρχείο κειμένου
		student.saveToBinaryFile(fileName);
		// Ανάκτηση από αρχείο κειμένου
		Student loadedStudent = Student.readFromBinaryFile(fileName);
		System.out.println("Loaded Student: " + loadedStudent);
	}
}
```

8.6.5. Εξαγωγή δεδομένων σε διαφορετικές μορφές και σειριοποίηση

Οι σύγχρονες εφαρμογές συνήθως χρειάζονται αποθήκευση ή ανταλλαγή δεδομένων σε διαφορετικές μορφές (όπως XML ή JSON). Η Java παρέχει σχετικές βιβλιοθήκες μετατροπής αντικειμένων σε αυτές τις μορφές, για παράδειγμα για τη μορφή JSON υπάρχουν οι βιβλιοθήκες Gson και Jackson οι οποίες διατηρούν τις πληροφορίες των αντικειμένων σε μορφή που είναι πιο φιλική προς τον χρήστη (μορφή αναγνώσιμη από τον άνθρωπο) και ανεξάρτητη από την πλατφόρμα.

Στο παράδειγμα που ακολουθεί θα μετατρέψουμε ένα αντικείμενο μιας κλάσης Student (που ορίζεται παρακάτω) σε αντικείμενο JSON χρησιμοποιώντας τη βιβλιοθήκη Gson. Αρχικά θα πρέπει να την προσθέσουμε στο πρότζεκτ μας στο eclipse και ένας τρόπος είναι ο εξής:

- Στη διεύθυνση https://search.maven.org/artifact/com.google.code.gson/gson
- Θα βρούμε λίστα με τις εκδόσεις. Επιλέγουμε την επιθυμητή έκδοση, περνάμε στη σελίδα της όπου υπάρχει πλήκτρο Downloads και κατεβάζουμε το jar (Εικόνα 35).



Εικόνα 35 - κατέβασμα jar του Gson

- Στο eclipse κάνουμε δεξί κλικ στο project και Properties.

- Καρτέλα Libraries, κλικ στο Classpath και Add External JARs από όπου επιλέγουμε το αρχείο που κατεβάσαμε (δεν θα μπούμε σε λεπτομέρειες για το Classpath, αποθηκεύστε το jar σε μέρος που να μην κινδυνεύει να διαγραφεί).
- Apply and Close

Η κλάση Student (ενδεικτική, για το παράδειγμα) είναι η ακόλουθη:

```
import java.util.Arrays;

class Student {
	private String name;
	private int age;
	private String[] subjects;

	public Student(String name, int age, String[] subjects) {
		this.name = name;
		this.age = age;
		this.subjects = subjects;
	}

	@Override
	public String toString() {
		return "Student{name='" + name + "', age=" + age + ", 
subjects=" + Arrays.toString(subjects) + "}";
	}
}
```

Στην μέθοδο `main()` δημιουργούμε ένα αντικείμενο `Student`, ένα αντικείμενο JSON και μετατρέπουμε την κλάση σε JSON. Εμφανίζουμε το JSON (είναι κείμενο) και στη συνέχεια μετατρέπουμε το JSON σε αντικείμενο κλάσης `Student` και με την μέθοδο `toString()` εμφανίζουμε τα πεδία. Αν προσέξετε, ουσιαστικά κάνουμε σειριοποίηση του `Student` σε JSON και αποσειριοποίηση του JSON σε Student. Αυτό δεν είναι σειριοποίηση με την παραδοσιακή έννοια του **Java Serialization**, αλλά μια εναλλακτική μορφή που επιτρέπει την αποθήκευση και ανάκτηση δεδομένων σε διαφορετικές μορφές. Το αποτέλεσμα της εκτέλεσης το βλέπουμε στην Εικόνα 36.

```
import com.google.gson.*;

public class Main {
	public static void main(String[] args) {
		// Δημιουργία αντικειμένου Student
		Student student = new Student("John Doe", 20, new String[] {
"Math", "Science" });

		// Μετατροπή του αντικειμένου σε JSON
		GsonBuilder builder = new GsonBuilder();
		builder.setPrettyPrinting(); // Για μορφοποιημένη εμφάνιση του
JSON
		Gson gson = builder.create(); // Αντικείμενο JSON
		String json = gson.toJson(student); // Μετατροπή κλάσης σε JSON

		// Εμφάνιση του JSON
		System.out.println("Serialized JSON:\n " + json);

		// Μετατροπή από JSON σε αντικείμενο της κλάσης
		Student dStudent = gson.fromJson(json, Student.class);
		System.out.println("Deserialized Student: " +
dStudent.toString());
```

```
        }
}
```

Το αποτέλεσμα της εκτέλεσης:

```
Serialized JSON:
 {
  "name": "John Doe",
  "age": 20,
  "subjects": [
    "Math",
    "Science"
  ]
}
Deserialized Student: Student{name='John Doe', age=20, subjects=[Math, Science]}
```

Εικόνα 36 - από κλάση σε JSON και αντίστροφα

Η βιβλιοθήκη Gson λειτουργεί σε επίπεδο χαρακτήρων, καθώς παράγει και διαχειρίζεται δεδομένα κειμένου. Δημιουργεί αναγνώσιμα αλφαριθμητικά τα οποία, στη συνέχεια, μπορούμε να διαχειριστούμε όπως χρειαστεί.

Για να αποθηκεύσουμε το JSON σε αρχείο, μπορούμε να το μετατρέψουμε σε character stream και να το γράψουμε σε αρχείο κειμένου.

```
// Γράψιμο JSON σε αρχείο με character stream
try (BufferedWriter writer = new BufferedWriter(new
FileWriter("student.json"))) {
writer.write(json);
      } catch (IOException e) {
      e.printStackTrace();
}
```

Αν θέλουμε να το αποθηκεύσουμε (ή να το μεταδώσουμε) ως byte stream, μπορούμε να το μετατρέψουμε σε bytes.

```
// Μετατροπή JSON σε byte stream και αποθήκευση
try (FileOutputStream fos = new FileOutputStream("student.dat")) {
fos.write(json.getBytes(StandardCharsets.UTF_8));
      } catch (IOException e) {
       e.printStackTrace();
}
```

Έτσι, χρησιμοποιούμε τις εξωτερικές βιβλιοθήκες για να δημιουργήσουμε δομημένα δεδομένα (π.χ., JSON) και, αν θέλουμε, τα διαχειριζόμαστε ως byte stream για πιο παραδοσιακή σειριοποίηση.

Σημείωση: αν θέλετε να προσθέσετε στην προηγούμενη `main()` τους τελευταίους κώδικες που παρουσιάσαμε, θα πρέπει να κάνετε τα κατάλληλα imports, π.χ. τα `java.io.*;` και `java.nio.charset.StandardCharsets;`

8.7. Πρακτική Εξάσκηση

Δημιουργήστε μια εφαρμογή διαχείρισης αιτήσεων που κατατίθενται από πολίτες. Η εφαρμογή θα χρησιμοποιεί την παραδοσιακή σειριοποίηση της Java για να αποθηκεύσει τις αιτήσεις σε αρχείο σε bytes και την παραδοσιακή αποσειριοποίηση για να ανακτά τα δεδομένα. Παρακάτω θα βρείτε περιγραφή της εφαρμογής και ενδεικτικές οδηγίες για την υλοποίησή της.

Η κύρια λογική της εφαρμογής θα είναι:

- Ξεκινά με έλεγχο ύπαρξης του προκαθορισμένου (σε μια μεταβλητή στη `main`) μονοπατιού προς το αρχείο καθώς και του αρχείου. Αν περιέχει δεδομένα, αυτά φορτώνονται σε έναν πίνακα `requests`.
- Στον χρήστη εμφανίζεται ένα μενού επιλογών μέσα από το οποίο μπορεί να α) προσθέσει νέο αίτημα, β) προβάλει όλα τα αιτήματα, γ) τερματίσει την εφαρμογή. Η διαδικασία αυτή εκτελείται επαναληπτικά, μέχρι να επιλέξει τερματισμό ο χρήστης.
- Τα δεδομένα του πίνακα `requests`, πριν από την έξοδο, αποθηκεύονται στο αρχείο.

Ειδικότερα, μπορεί να περιλαμβάνει τρεις κλάσεις:

Α. `Request` που αντιπροσωπεύει μια αίτηση. Πεδία: ονοματεπώνυμο πολίτη (`String`), αριθμός πρωτοκόλλου (`int`), περιεχόμενο αίτησης (`String`). Προσθέστε στην κλάση τη δυνατότητα να σειριοποιηθεί, κατασκευαστή που θα αναθέτει τιμές στα πεδία, μια μέθοδο `toString()` που θα επιστρέφει τα δεδομένα της αίτησης (π.χ. όνομα και τιμή του κάθε πεδίου).

Β. `Utility` που θα περιέχει στατικές μεθόδους για την σειριοποίηση/αποθήκευση σε αρχείο, την ανάκτηση/αποσειριοποίηση από το αρχείο και τον έλεγχο ύπαρξης του φακέλου αποθήκευσης και του αρχείου δεδομένων. Ειδικότερα:

- Η `saveToFile()` δέχεται έναν πίνακα από αντικείμενα `Request` και το πλήρες όνομα του αρχείου (μονοπάτι από τη ρίζα του δίσκου μαζί με όνομα αρχείου). Διατρέχει τον πίνακα, σειριοποιεί και γράφει διαδοχικά τα αντικείμενα `Request` στο αρχείο.
- Η `loadFromFile()` δέχεται ως όρισμα το πλήρες όνομα του αρχείου, φορτώνει και αποσειριοποιεί τα δεδομένα και τα επιστρέφει σε μορφή πίνακα από `Requests`.
- Η `checkFileName()` δέχεται ως όρισμα το πλήρες όνομα αρχείου και ελέγχει αν ο φάκελος ή το αρχείο υπάρχουν, επιστρέφοντας αντίστοιχη τιμή (π.χ., 3 αν ο φάκελος δεν υπάρχει, 2 αν ο φάκελος υπάρχει αλλά όχι το αρχείο, 1 αν υπάρχουν φάκελος και αρχείο).

Γ. `ManageRequests` που θα περιέχει τη `main()`, η οποία:

- Δημιουργεί ένα όνομα αρχείου `(fileName)` για την αποθήκευση σειριοποιημένων δεδομένων.
- Διατηρεί έναν πίνακα `requests` από αντικείμενα `Requests` που αρχικά δείχνει `null`.
- Ελέγχει αν ο φάκελος και το αρχείο υπάρχουν με τη βοήθεια της μεθόδου `checkFileName()` από την κλάση `Utility`.
- Ανάλογα με το αποτέλεσμα του ελέγχου, εκτελεί διαφορετικές ενέργειες:
- αν ο φάκελος δεν υπάρχει, το πρόγραμμα τερματίζει,
- αν υπάρχει φάκελος και αρχείο , γίνεται ανάκτηση δεδομένων στον πίνακα requests
- αν υπάρχει ο φάκελος και όχι το αρχείο, διατήρηση της τιμής `requests` = `null`.
- Εμφανίζει επαναληπτικά μενού επιλογών με τις οποίες ο χρήστης μπορεί να κάνει τις λειτουργίες που περιεγράφηκαν παραπάνω, δηλαδή:

- Εισαγωγή αιτήματος: Δημιουργεί ένα νέο αντικείμενο `Request` και το προσθέτει στον πίνακα `requests`. Αν είναι το πρώτο αίτημα, δημιουργεί τον πίνακα, αλλιώς επεκτείνει τον πίνακα με νέο στοιχείο. Για να επεκτείνει τον πίνακα, δημιουργεί ένα νέο, κατά μια θέση μεγαλύτερο, αντιγράφει τις υπάρχουσες αιτήσεις, προσθέτει στο τέλος τη νέα και βάζει το `requests` να δείξει σε αυτόν.
- Εμφάνιση αιτημάτων: Διατρέχει τον πίνακα `requests` και εμφανίζει τα περιεχόμενα του.
- Έξοδος: Αποθηκεύει τα αιτήματα σε αρχείο, κλείνει `το Scanner` που έχει ανοίξει για εισαγωγή δεδομένων από το χρήστη.

Απάντηση. Παραθέτουμε παρακάτω μια ενδεικτική επίλυση της άσκησης, με την παρατήρηση ότι ο κώδικας μπορεί να μην ελέγχει όλες τις περιπτώσεις, αλλά δίνεται ως δείγμα για εκπαιδευτικούς λόγους.

```
import java.io.Serializable;

public class Request implements Serializable {
	private static final long serialVersionUID = 1324L;
	private String fullName;
	private int protocol;
	private String content;

	Request(String fullName, int protocol, String content) {
		this.fullName = fullName;
		this.protocol = protocol;
		this.content = content;
	}

	@Override
	public String toString() {
		return "Ονοματεπώνυμο: [" + fullName + "] Πρωτόκολλο:[ " +
protocol + "] Περιεχόμενο: [" + content + "]";
	}
}


import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.EOFException;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.ArrayList;
import java.util.List;

public class Utility {
	public static void saveToFile(Request[] requests, String fileName) {
		try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream(fileName))) {
			for (Request req : requests)
				oos.writeObject(req);
		} catch (IOException e) {
			e.printStackTrace();
```

```
                System.exit(4);
            }
        }

        public static Request[] loadFromFile(String fileName) {
            List<Request> reqList = new ArrayList<>();
            try (ObjectInputStream ois = new ObjectInputStream(new
FileInputStream(fileName))) {
                while (true) {
                    try {
                        Request req = (Request) ois.readObject();
                        reqList.add(req);
                    } catch (EOFException e) { // Τέλος αρχείου
                        break; // of while
                    }
                }
            } catch (IOException | ClassNotFoundException e) {
                e.printStackTrace();
                System.exit(5);
            }
            return reqList.toArray(new Request[0]);
        }

        public static int checkFileName(String fullPath) {
            Path path = Paths.get(fullPath);
            Path folderPath = path.getParent();
            // Έλεγχος αν ο φάκελος υπάρχει
            if (Files.exists(folderPath) && Files.isDirectory(folderPath))
{
                // Έλεγχος αν το αρχείο υπάρχει στον φάκελο
                if (Files.exists(path) && Files.isRegularFile(path)) {
                    return 1; // Υπάρχει το αρχείο μέσα στο φάκελο
                } else {
                    return 2; // Υπάρχει ο φάκελος αλλά όχι το αρχείο
μέσα του
                }
            } else {
                return 3; // Δεν υπάρχει ο φάκελος
            }
        }
}


import java.util.Scanner;

public class ManageRequests {

        public static void main(String[] args) {
            String fileName = "request.ser"; // αρχείο αποθήκευσης
            Request[] requests = null; // πίνακας αποθήκευσης αιτήσεων
            int check = Utility.checkFileName(fileName); // έλεγχος ύπαρξης
φακέλου, αρχείου
            switch (check) {
            case 3:
                System.out.println("Ο φάκελος δεν υπάρχει!");
                System.exit(3);
                break;
            case 2: // Δεν υπάρχουν αιτήματα στο αρχείο
                break;
            case 1:
```

```
                requests = Utility.loadFromFile(fileName);
                break;
            }
            int selection = 5;
            Scanner sc = new Scanner(System.in);
            while (selection != 0) {
                System.out.println("1. Εισαγωγή αιτήματος");
                System.out.println("2. Εμφάνιση αιτημάτων");
                System.out.println("0. Έξοδος");
                System.out.print("Επιλέξτε:");
                selection = sc.nextInt();
                sc.nextLine(); // καθαρισμός υπόλοιπης γραμμής
                switch (selection) {
                case 1:
                    System.out.print("Ονοματεπώνυμο:");
                    String x = sc.nextLine();
                    System.out.print("Πρωτόκολλο:");
                    int y = sc.nextInt();
                    sc.nextLine();
                    System.out.print("Περιεχόμενο:");
                    String z = sc.nextLine();
                    Request rc = new Request(x, y, z);
                    if (requests == null) { // πρώτο αίτημα
                        requests = new Request[1];
                        requests[0] = rc;
                    } else { // επόμενα αιτήματα εκτός του 1ου
                        Request tmp[] = new Request[requests.length +
1];
                        System.arraycopy(requests,    0,    tmp,    0,
requests.length);
                        tmp[tmp.length - 1] = rc;
                        requests = tmp;
                    }
                    break;
                case 2:
                    if (requests != null) // αν περιέχονται αιτήματα
στον πίνακα
                        for (Request item : requests)
                            System.out.println(item.toString());
                    else
                        System.out.println("Δεν υπάρχουν αιτήματα!");
                }
            }
            if (requests != null)
                Utility.saveToFile(requests, fileName);
            sc.close();
        }
}
```

8.8. Ερωτήσεις Αυτο-αξιολόγησης

1. Ποιο από τα παρακάτω πακέτα χρησιμοποιείται για λειτουργίες εισόδου/εξόδου στην Java;

α) java.util.io

β) java.io

γ) java.nio

δ) java.streams

2. **Ποια μέθοδος της κλάσης File ελέγχει αν ένα αρχείο υπάρχει;**

   α) isFile()

   β) exists()

   γ) isDirectory()

   δ) canRead()

3. **Ποια είναι η κύρια διαφορά μεταξύ δυαδικών ρευμάτων (byte streams) και ρευμάτων χαρακτήρων (character streams);**

   α) Τα byte streams διαχειρίζονται χαρακτήρες, ενώ τα character streams διαχειρίζονται bytes

   β) Τα byte streams διαχειρίζονται bytes, ενώ τα character streams διαχειρίζονται χαρακτήρες

   γ) Δεν υπάρχει διαφορά

   δ) Τα character streams είναι ταχύτερα από τα byte streams

4. **Ποιο από τα παρακάτω δεν είναι κλάση δυαδικού ρεύματος (byte stream);**

   α) FileInputStream

   β) BufferedInputStream

   γ) FileWriter

   δ) DataOutputStream

5. **Ποια μέθοδος της κλάσης Console χρησιμοποιείται για την εισαγωγή δεδομένων από τον χρήστη;**

   α) readLine()

   β) inputLine()

   γ) getLine()

   δ) read()

6. **Ποιο πακέτο εισήγαγε βελτιωμένες λειτουργίες διαχείρισης αρχείων και καταλόγων;**

   α) java.nio

   β) java.io

   γ) java.util

   δ) java.fs

7. **Ποια μέθοδος της κλάσης Files ελέγχει αν ένα αρχείο είναι κενό;**

   α) isEmpty()

   β) isBlank()

   γ) size()

   δ) readAllBytes()

8. **Ποιο interface πρέπει να υλοποιήσει μια κλάση για να γίνει serializable;**

   α) Serializable

   β) Serialization

   γ) Externalizable

   δ) Streamable

9. **Ποια μεταβλητή δεν θα σειριοποιηθεί σε μια κλάση που υλοποιεί το Serializable;**

   α) Μεταβλητές με την ένδειξη private

   β) Μεταβλητές με την ένδειξη transient

   γ) Μεταβλητές με την ένδειξη protected

   δ) Μεταβλητές με την ένδειξη final

10. Ποια είναι η σημασία της μεταβλητής serialVersionUID;

α) Χρησιμοποιείται για να αυξήσει την ταχύτητα σειριοποίησης

β) Ελέγχει τη συμβατότητα της κλάσης κατά την αποσειριοποίηση

γ) Αποθηκεύει το μέγεθος του αντικειμένου

δ) Ορίζει το όνομα του stream

11. Ποια κλάση χρησιμοποιείται για τη χρήση buffered streams για είσοδο δεδομένων;

α) BufferedReader

β) BufferedInputStream

γ) DataInputStream

δ) InputStreamReader

12. Ποια από τις παρακάτω κλάσεις μπορεί να χρησιμοποιηθεί για να αποθηκεύσει αντικείμενα στη μνήμη αντί σε αρχείο;

α) ByteArrayOutputStream

β) FileOutputStream

γ) BufferedOutputStream

δ) DataOutputStream

13. Ποια μέθοδος της κλάσης File επιστρέφει το απόλυτο μονοπάτι ενός αρχείου;

α) getPath()

β) getAbsolutePath()

γ) getCanonicalPath()

δ) getFullPath()

14. Ποια μέθοδος του Files χρησιμοποιείται για να διαβάσει όλα τα bytes ενός αρχείου;

α) readAllBytes()

β) readAllLines()

γ) read()

δ) getBytes()

15. Ποια από τις παρακάτω μεθόδους δεν απαιτείται να υλοποιηθεί κατά την υλοποίηση του Serializable;

α) writeExternal()

β) readObject()

γ) writeObject()

δ) Καμία από τις παραπάνω

16. Ποιο από τα παρακάτω είναι σωστό για την κλάση Externalizable;

α) Είναι υποπακέτο του Serializable

β) Απαιτεί υλοποίηση των μεθόδων writeExternal() και readExternal()

γ) Δεν μπορεί να χρησιμοποιηθεί για σειριοποίηση

δ) Χρησιμοποιείται μόνο για δυαδικά δεδομένα

17. Ποια μέθοδος του System.out μπορεί να χρησιμοποιηθεί για εκτύπωση στη standard έξοδο;

α) println()

β) write()

γ) printObject()

δ) outPrint()

18. Ποιο είναι το αποτέλεσμα της χρήσης System.err σε μια Java εφαρμογή;

α) Εμφάνιση λάθους στην οθόνη

β) Επιστροφή λάθους στον compiler

γ) Έξοδος δεδομένων σε ξεχωριστό log file

δ) Τερματισμός της εφαρμογής

19. Ποια κλάση στο java.nio package χρησιμοποιείται για να διαβάσει ένα αρχείο σε μορφή stream;

α) Files

β) Path

γ) DirectoryStream

δ) StreamReader

20. Τι συμβαίνει κατά την αποσειριοποίηση αν η κλάση έχει αλλάξει μετά τη σειριοποίηση;

α) Η διαδικασία αποτυγχάνει

β) Δημιουργείται νέα έκδοση του αντικειμένου

γ) Χρησιμοποιείται το serialVersionUID για έλεγχο συμβατότητας

δ) Τα δεδομένα χάνονται

8.9. Απαντήσεις στις ερωτήσεις Αυτο-αξιολόγησης

1. β	2. β	3. β	4. γ	5. α
6. α	7. γ	8. α	9. β	10. β
11. β	12. α	13. β	14. α	15. δ
16. β	17. α	18. α	19. α	20. γ

ΚΕΦΑΛΑΙΟ 9: Σχεδιασμός γραφικού περιβάλλοντος (GUI)

9.1. Εκπαιδευτικοί στόχοι του παρόντος κεφαλαίου

Οι εκπαιδευτικοί στόχοι του παρόντος κεφαλαίου, συνοψίζονται στα κάτωθι σημεία. Οι εκπαιδευόμενοι:

- θα μάθουν να αναγνωρίζουν τα Γραφικά Περιβάλλοντα - Java Foundation Classes (JFC),
- θα κατανοήσουν τα εισαγωγικά στοιχεία για τα συστατικά AWT και Swing,
- θα μάθουν να χρησιμοποιούν κουμπιά (Buttons), πεδία Κειμένου (TextFields) και άλλα Components,
- θα μάθουν να εξετάζουν τους διαχειριστές διάταξης (Layout Managers),
- θα κατανοήσουν τον Event-Driven προγραμματισμό,
- θα μάθουν να εφαρμόζουν τον χειρισμό των συμβάντων (Event-Handling),
- θα μάθουν να δημιουργούν Μενού (Menu),
- θα μάθουν να δημιουργούν Διάλογους (Dialogs),
- θα μάθουν να σχεδιάζουν User Interface με τη χρήση Eclipse WindowBuilder Pro,
- θα μάθουν να αξιολογούν και υλοποιούν την αρχιτεκτονική Model-View-Controller (MVC) στην Java.

9.2. Εισαγωγή στη Java GUI (Java Foundation Classes, AWT, Swing)

Το Java Foundation Classes (JFC) είναι μια συλλογή από βιβλιοθήκες με τις οποίες δημιουργούμε τις γραφικές διεπαφές (GUI) με τον χρήστη. Τα κυριότερα μέλη που περιλαμβάνει είναι τα:

- Abstract Window Toolkit (AWT). Περιέχει βασικά στοιχεία GUI (κουμπιά, λίστες, μενού) που εξαρτώνται από το λειτουργικό σύστημα (heavyweight), επομένως από την πλατφόρμα εκτέλεσης. Αυτό έχει ως αποτέλεσμα θέματα ομοιομορφίας στην εμφάνιση αλλά και πιθανές ασυμβατότητες σε διαφορετικά περιβάλλοντα (π.χ. Windows και MacOS).
- Swing. Είναι επέκταση του AWT με πλουσιότερα στοιχεία, γραμμένο πλήρως σε Java και ως εκ τούτου ανεξάρτητο της πλατφόρμας εκτέλεσης (lightweight συστατικά). Υποστηρίζει τα λεγόμενα Java look and feels, δηλαδή ομοιόμορφες εμφανίσεις και συμπεριφορές στη χρήση των συστατικών του GUI (Java Nimbus look and feel, Java Metal look and feel, κ.λπ.). Ο σχεδιασμός του Swing είναι στηριγμένος στην αρχιτεκτονική ανάπτυξης εφαρμογών MVC (Model-View-Controller), θα δούμε περισσότερες πληροφορίες παρακάτω.
- Java 2D. Για σχεδίαση γραφικών δύο διαστάσεων.
- Accessibility API. Για τη δημιουργία λογισμικού που μπορεί να χρησιμοποιηθεί από χρήστες με περιορισμούς, όπως προβλήματα όρασης ή κίνησης.

Τα πιο συχνά χρησιμοποιούμενα πακέτα του Swing είναι τα `javax.swing` και `javax.swing.event`. Για μια γρήγορη εισαγωγή, θα δούμε δύο μικρά παραδείγματα κώδικα. Θα εξηγήσουμε κάποια σημεία, και παρακάτω θα τα δούμε και πιο αναλυτικά.

Στο πρώτο παράδειγμα[1], το πρόγραμμα δημιουργεί ένα παράθυρο και εμφανίζει ένα κουμπί. Αυτό το επιτυγχάνουμε χρησιμοποιώντας τις κλάσεις `JFrame` (για το παράθυρο) και `JButton` (για το κουμπί).

```
import javax.swing.JButton;
import javax.swing.JFrame;
public class Swing1 {
	public static void main(String[] args) {
		// Δημιουργία του κύριου παραθύρου (JFrame)
		JFrame frame = new JFrame("Παράδειγμα Swing 1");
		frame.setSize(350, 200);
		// Όταν κλείσουμε το παράθυρο, να τερματίζει η εφαρμογή
		frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		frame.setLayout(null); // Χρήση απόλυτης διάταξης
		// Δημιουργία κουμπιού (JButton)
		JButton button = new JButton("Πατήστε εδώ!");
		// Θέση και μέγεθος του κουμπιού
		button.setBounds(100, 80, 150, 30);
		// Προσθήκη του κουμπιού στο πλαίσιο
		frame.add(button);
		// Εμφάνιση του παραθύρου
		frame.setVisible(true);
	}
}
```

Αρχικά δημιουργούμε ένα αντικείμενο `JFrame` (το κύριο παράθυρο της εφαρμογής) περνώντας στον κατασκευαστή ένα `String` που θα είναι ο τίτλος του παράθυρου. Η μέθοδος `setSize()` ορίζει τις απόλυτες διαστάσεις του παράθυρου σε pixels, που στον κώδικα είναι 350 px πλάτος και 200 px ύψος. Η μέθοδος `setDefaultCloseOperation()` καθορίζει πως όταν κλείσουμε το παράθυρο θα τερματίσει η εφαρμογή. Η μέθοδος `setLayout()` με όρισμα `null` καθορίζει ότι ο προγραμματιστής θα τοποθετεί τα αντικείμενα (π.χ. το κουμπί) στη θέση που επιθυμεί.

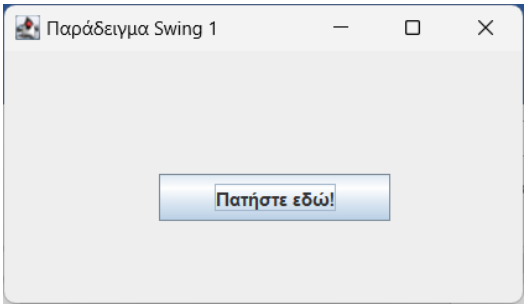
Στη συνέχεια δημιουργούμε ένα αντικείμενο `JButton`, δηλαδή ένα κουμπί, περνώντας στον κατασκευαστή το μήνυμα που θέλουμε αυτό να εμφανίζει πάνω του. Η μέθοδος `setBounds()` καθορίζει το μέγεθος και την απόλυτη θέση του κουμπιού μέσα στο παράθυρο. Στο παράδειγμα, το κουμπί τοποθετείται στις συντεταγμένες (100, 80) και έχει πλάτος 150 px και ύψος 30 px. Εδώ κάναμε λίγο αριθμητική: το πλάτος του παράθυρου είναι 350 px και το πλάτος του κουμπιού 150 px. Τοποθετούμε το κουμπί από τη θέση 100, καταλαμβάνει πλάτος 150, άρα απομένουν άλλα 100 μετά το κουμπί. Αυτό τοποθετεί το κουμπί στο κέντρο της φόρμας (οριζόντια στοίχιση). Ωστόσο, αν κάποιος κάνει μεγιστοποίηση στη φόρμα ή της αλλάξει το μέγεθος καθώς ο κώδικας εκτελείται, η στοίχιση θα χαθεί.

[1] Αν τα εκτελέσετε στο eclipse μπορεί να χρειαστεί να προσθέσετε **requires java.desktop;** στο module-info.java

Παρατηρήστε ότι μέχρι τώρα δεν έχουμε συσχετίσει το κουμπί με το παράθυρο. Αυτό γίνεται με τη μέθοδο `add()` της κλάσης `JFrame`, που προσθέτει το κουμπί στο παράθυρο που δημιουργήσαμε.

Η μέθοδος `setVisible()` κάνει το παράθυρο ορατό. Αν εκτελούσαμε τον κώδικα χωρίς να την συμπεριλάβουμε, το παράθυρο θα είχε μεν δημιουργηθεί, αλλά δεν θα εμφανιζόταν στην οθόνη. Η λογική αυτή δίνει τη δυνατότητα στον προγραμματιστή να δημιουργεί παράθυρα της εφαρμογής μια φορά και να τα εμφανίζει όποτε χρειάζεται, χωρίς να χρειάζεται η (χρονοβόρα) επαναδημιουργία των αντικειμένων.

Αν εκτελέσετε τον κώδικα, θα δείτε την Εικόνα 37.



Εικόνα 37 - το πρώτο μας παράθυρο με το Java Swing

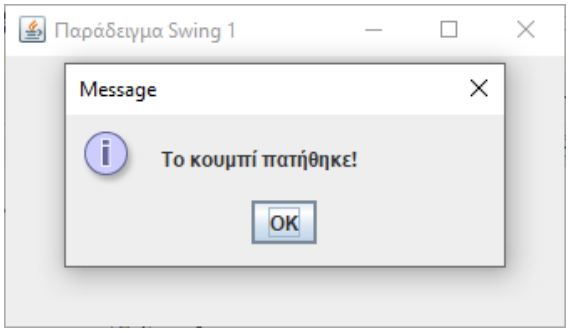
Δοκιμάστε να πατήσετε το κουμπί και θα δείτε ότι δεν κάνει τίποτε. Αυτό οφείλεται στο ότι δεν έχουμε περιλάβει κώδικα που να παρακολουθεί τις ενέργειες του χρήστη και να αποκρίνεται κατάλληλα σε αυτές. Για να το κάνουμε αυτό, θα χρειαστούμε τις μεθόδους του πακέτου `javax.swing.event`. Δείτε ένα δείγμα στον κώδικα που ακολουθεί, όπου έχουμε συμπληρώσει μερικά imports και λίγο κώδικα προσθήκης μιας ενδεικτικής λειτουργικότητας στο κουμπί.

```
import javax.swing.JButton;
import javax.swing.JFrame;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JOptionPane;
public class Swing2 {
	public static void main(String[] args) {
		// Δημιουργία του κύριου παραθύρου (JFrame)
		JFrame frame = new JFrame("Παράδειγμα Swing 1");
		frame.setSize(350, 200);
		// Όταν κλείσουμε το παράθυρο, να τερματίζει η εφαρμογή
		frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		frame.setLayout(null); // Χρήση απόλυτης διάταξης
		// Δημιουργία κουμπιού (JButton)
		JButton button = new JButton("Πατήστε εδώ!");
		// Θέση και μέγεθος του κουμπιού
		button.setBounds(100, 80, 150, 30);
		// *ΕΔΩ* Προσθήκη λειτουργικότητας στο κουμπί
		button.addActionListener(new ActionListener() {
			@Override
			public void actionPerformed(ActionEvent e) {
				JOptionPane.showMessageDialog(frame,   "Το   κουμπί
πατήθηκε!");
			}
		});

		// Προσθήκη του κουμπιού στο πλαίσιο
```

```
            frame.add(button);
            // Εμφάνιση του παραθύρου
            frame.setVisible(true);
        }
}
```

Η μέθοδος `addActionListener(new ActionListener() { ... })` της κλάσης `JButton` συνδέει ένα αντικείμενο listener (ακροατής) με το κουμπί. Στο παραπάνω παράδειγμα, το αντικείμενο είναι ανώνυμο (`new ActionListener`) καθώς η Java μας δίνει αυτή τη δυνατότητα, αν δεν χρειαζόμαστε να του δώσουμε όνομα, π.χ. για να το χρησιμοποιήσουμε αλλού. Μέσα έχουμε συμπεριλάβει τον κώδικα που θα εκτελεστεί όταν το κουμπί πατηθεί.

Το `ActionListener` είναι interface και χρειάζεται να υλοποιήσουμε τη μέθοδο `actionPerformed()`, δείτε το `@Override` που προηγείται. Ο κώδικας αποκρίνεται σε γεγονός (event), που εδώ είναι το πάτημα του κουμπιού (Εικόνα 38). Το αναδυόμενο παράθυρο με το μήνυμα εμφανίζεται με χρήση της μεθόδου `JOptionPane.showMessageDialog()`. Η κλάση `JoptionPane` μας παρέχει ευκολίες για να εμφανίζουμε μηνύματα προς τον χρήστη καθώς και πιο σύνθετες μορφές διαλόγου και θα την δούμε παρακάτω.



Εικόνα 38 – προσθέτοντας λειτουργικότητα στο JButton

Προτού προχωρήσουμε, λάβετε υπόψη ότι πέραν των παραπάνω, το JavaFX είναι μια πιο σύγχρονη βιβλιοθήκη για την ανάπτυξη GUI εφαρμογών και δημιουργήθηκε με στόχο να αποτελέσει το διάδοχο του Swing. Παρόλο που το JavaFX θεωρείται πιο ευέλικτο και κατάλληλο για σύνθετες και όμορφης εμφάνισης εφαρμογές, το Swing εξακολουθεί και σήμερα να χρησιμοποιείται λόγω της μακροχρόνιας παρουσίας του και της ευρείας υποστήριξης που έχει στην κοινότητα ανάπτυξης λογισμικού με Java.

9.3. Βασικά Συστατικά GUI (Components) και Διαχειριστές Διάταξης (Layout Managers)

Στην ενότητα αυτή θα συνεχίσουμε με αναλυτικότερες παρουσιάσεις των στοιχείων που χρειάζονται για τη δημιουργία GUI στη Java. Θα δούμε τα κυριότερα Συστατικά (Components) GUI στο Swing, τους Διαχειριστές Διάταξης και τη συνδυαστική χρήση Συστατικών με Διαχειριστές Διάταξης.

Τα Συστατικά GUI είναι αντικείμενα που αναπαριστούν στοιχεία διεπαφής με τον χρήστη, για παράδειγμα κουμπιά, πλαίσια κειμένου, πτυσσόμενες λίστες, κ.λπ. Τέτοια περιλαμβάνουν τόσο το AWT, όσο και το Swing (αλλά και το Java FX).

Στην περίπτωση του Swing, τα κυριότερα συστατικά και η λειτουργικότητά τους είναι η ακόλουθη:

- `JFrame`. Δημιουργεί ένα top-level παράθυρο που τυπικά χρησιμοποιείται ως το κύριο παράθυρο της εφαρμογής. Μπορούμε, αν θέλουμε, να δημιουργήσουμε και άλλα, ωστόσο αυτό είναι γενικά μια κακή πρακτική. Στη συντριπτική πλειοψηφία των περιπτώσεων, μια εφαρμογή ξεκινά με ένα ακριβώς αντικείμενο `JFrame`.
- `JPanel`. Δημιουργεί ένα Δοχείο (κοντέινερ) για ομαδοποίηση συστατικών.
- `JLabel`. Αντικείμενο Ετικέτας για κείμενο ή εικόνες.
- `JButton`. Κουμπί για αλληλεπίδραση με τον χρήστη.
- `JTextField`. Περιοχή εισαγωγής κειμένου (μία γραμμή).
- `JTextArea`. Περιοχή εισαγωγής κειμένου (πολλές γραμμές).
- `JCheckBox`. Πλαίσιο ελέγχου (επιλογή ναι/όχι).
- `JRadioButton`. Ένα από μια ομάδα κουμπιών επιλογής. Όταν είναι ομαδοποιημένα, επιτρέπεται η επιλογή μόνον ενός από την ομάδα. Για να λειτουργούν σωστά ως ομάδα, τα `JRadioButton` πρέπει να ομαδοποιούνται με τη χρήση της κλάσης `ButtonGroup`.
- `JComboBox`. Πτυσσόμενη λίστα επιλογών (drop-down list).
- `JList`. Αναπτυγμένη λίστα επιλογών (δυνατότητα multi-select).

Τα παραπάνω συστατικά συνήθως τοποθετούνται πάνω σε έναν κοντέινερ (container). Αυτά είναι επίσης συστατικά που όμως μπορούν να περιέχουν άλλα συστατικά. Τα κυριότερα κοντέινερς είναι το `JFrame` και το `JPanel` (αλλά και τα `JDialog`, `JScrollPane`).

Ο προγραμματιστής μπορεί να επιλέξει να τοποθετήσει τα συστατικά σε κοντέινερ (π.χ. πάνω στο παράθυρο της εφαρμογής), με απόλυτο τρόπο, αλλά μπορεί και να χρησιμοποιήσει έναν Διαχειριστή Διάταξης (Layout Manager), που αναλαμβάνει αυτός την τοποθέτηση, με κάποιον προδιαγεγραμμένο τρόπο.

Η βασική ροή εργασίας στην υλοποίηση μιας εφαρομογής GUI, είναι, ενδεικτικά, η εξής:

- Δημιουργία παράθυρου εφαρμογής (`new JFrame`).
- Καθορισμός διαχειριστή διάταξης για το παράθυρο (`frame.setLAyout()`).
- Δημιουργία συστατικών που θα περιέχονται στο παράθυρο (π.χ. `new JButton`, `new JList`, κ.λπ.)
- Προσθήκη των συστατικών στο παράθυρο (`frame.add()`).
- Καθορισμός μεγέθους παράθυρου κατά την εκκίνηση (`frame.setSize()`).
- Άλλες ρυθμίσεις όπως π.χ. καθορισμός ενεργειών όταν ο χρήστης κλείσει το παράθυρο (`frame.setDefaultCloseOperation()`).
- Εμφάνιση του παράθυρου (`frame.setVisible()`).

Ορισμένοι διαχειριστές διάταξης περιλαμβάνονται στο πακέτο `java.swing` και άλλοι στο `java.awt`. Θα δούμε αμέσως μερικούς από τους κυριότερους διαχειριστές διάταξης, με ανάλογα, μικρά παραδείγματα. Αν δεν καταλαβαίνετε κάτι, μην ανησυχείτε, θα δούμε λεπτομέρειες

αργότερα. Για την ώρα επικεντρωθείτε στη λειτουργικότητα εμφάνισης που παρέχουν οι διαχειριστές διάταξης.

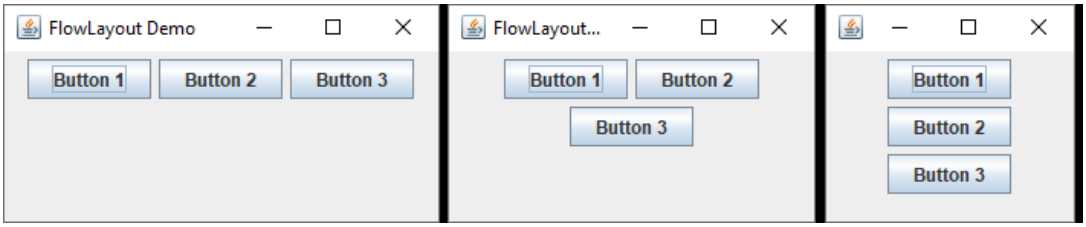
9.3.1. Διαχειριστές Διάταξης (Layout Managers)

Οι Διαχειριστές Διάταξης είναι επιφορτισμένοι με τη διάταξη («το στήσιμο») των συστατικών στο παράθυρο ή γενικά σε έναν κοντέινερ. Η Java παρέχει μια σειρά από σχετικές κλάσεις, ενδεικτικά:

- Ο `FlowLayout` είναι ίσως ο απλούστερος διαχειριστής διάταξης. Οργανώνει τα components σε σειρές, από αριστερά προς τα δεξιά. Όταν δεν χωράνε σε μία σειρά, μετακινούνται στην επόμενη γραμμή.

Στον κώδικα που ακολουθεί, έχουμε προσθέσει τρία κουμπιά σε ένα παράθυρο. Δείτε στην Εικόνα 39 την αρχική εμφάνιση του παράθυρου, αλλά και πώς τα κουμπιά αναδιατάσονται όταν αλλάξουμε τις διαστάσεις του (από αριστερά προς τα δεξιά).

```
import javax.swing.*;
import java.awt.*;

public class FlowLayoutDemo {
	public static void main(String[] args) {
		JFrame frame = new JFrame("FlowLayout Demo");
		// Διαχειριστής Διάταξης ο FlowLayout
		frame.setLayout(new FlowLayout());
		// Δημιουργία τριών κουμπιών
		JButton button1 = new JButton("Button 1");
		JButton button2 = new JButton("Button 2");
		JButton button3 = new JButton("Button 3");

		// Προσθήκη κουμπιών στο Frame
		frame.add(button1);
		frame.add(button2);
		frame.add(button3);

		// Καθορισμός μεγέθους παράθυρου, κ.λπ.
		frame.setSize(300, 150);
		frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		frame.setVisible(true);
	}
}
```

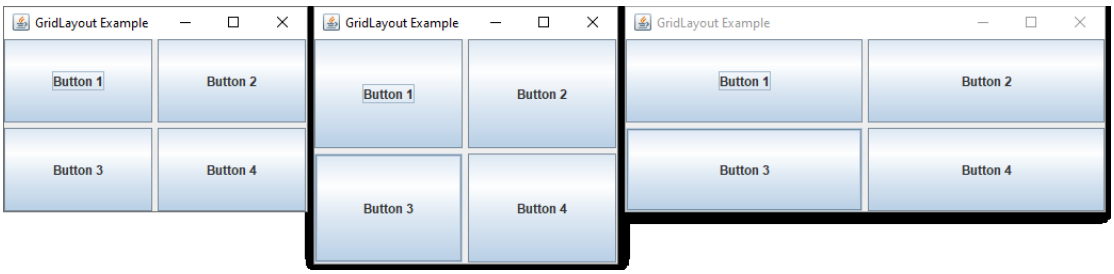


Εικόνα 39 – η συμπεριφορά του FlowLayout manager

- Ο `GridLayout` τακτοποιεί τα συστατικά σε έναν πίνακα (πλέγμα) με γραμμές και στήλες. Τα κελιά έχουν ίδιο μέγεθος και τα συστατικά τοποθετούνται σειριακά από αριστερά προς τα δεξιά και από πάνω προς τα κάτω και προσαρμόζονται για να γεμίσουν το κελί τους. Μπορούμε και να ορίσουμε κενό χώρο ανάμεσα σε γραμμές και στήλες.

Στον κώδικα που ακολουθεί προσθέτουμε έναν GridLayout στο παράθυρο και τέσσερα κουμπιά. Δείτε στην Εικόνα 40 τη συμπεριφορά του, στις αλλαγές μεγέθους του παράθυρου.

```
import javax.swing.*;
import java.awt.*;

public class GridLayoutDemo {
	public static void main(String[] args) {
		// Δημιουργία του frame
		JFrame frame = new JFrame("GridLayout Example");
		frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		frame.setSize(300, 200);

		// GridLayout με 2 γραμμές, 2 στήλες και 5 pixels απόσταση
ανάμεσα
		frame.setLayout(new GridLayout(2, 2, 5, 5));

		// Προσθήκη συστατικών στο frame
		frame.add(new JButton("Button 1"));
		frame.add(new JButton("Button 2"));
		frame.add(new JButton("Button 3"));
		frame.add(new JButton("Button 4"));

		// Εμφάνιση του frame
		frame.setVisible(true);
	}
}
```



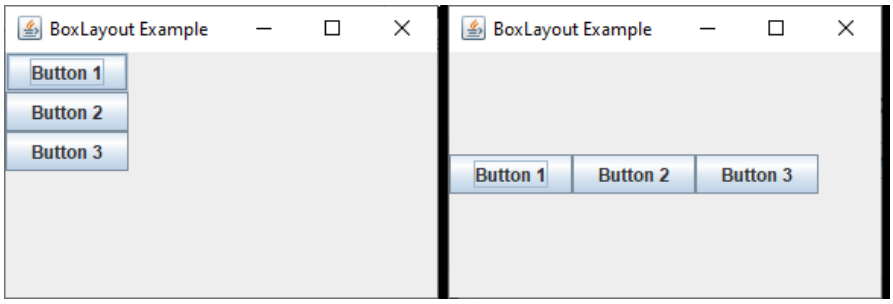
Εικόνα 40 - η συμπεριφορά του GridLayout manager

- Ο `BoxLayout` οργανώνει τα συστατικά σε μια κάθετη ή οριζόντια γραμμή, ανάλογα με την παράμετρο `BoxLayout.Y_AXIS`, ή `BoxLayout.X_AXIS` που περνάμε στον κατασκευαστή του.

Στον κώδικα που ακολουθεί, προσθέτουμε ένα `JPanel` στο `JFrame` και του καθορίζουμε διαχειριστή διάταξης τον `BoxLayout`. Στην Εικόνα 41 βλέπουμε αριστερά το αποτέλεσμα αν καθορίσουμε κατακόρυφη στοίχιση (αριστερά) και οριζόντια στοίχιση (δεξιά).

```
import javax.swing.*;

public class BoxLayoutDemo {
	public static void main(String[] args) {
		// Δημιουργία του frame
		JFrame frame = new JFrame("BoxLayout Example");
		frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		frame.setSize(300, 200);

		// Δημιουργία panel με BoxLayout
		JPanel panel = new JPanel();
```

```
            // Κατακόρυφη στοίχιση
            panel.setLayout(new BoxLayout(panel, BoxLayout.Y_AXIS));
            // Για Οριζόντια στοίχιση
            // panel.setLayout(new BoxLayout(panel, BoxLayout.X_AXIS));

            // Προσθήκη συστατικών
            panel.add(new JButton("Button 1"));
            panel.add(new JButton("Button 2"));
            panel.add(new JButton("Button 3"));

            // Προσθήκη panel στο frame
            frame.add(panel);

            // Εμφάνιση του frame
            frame.setVisible(true);
      }
}
```



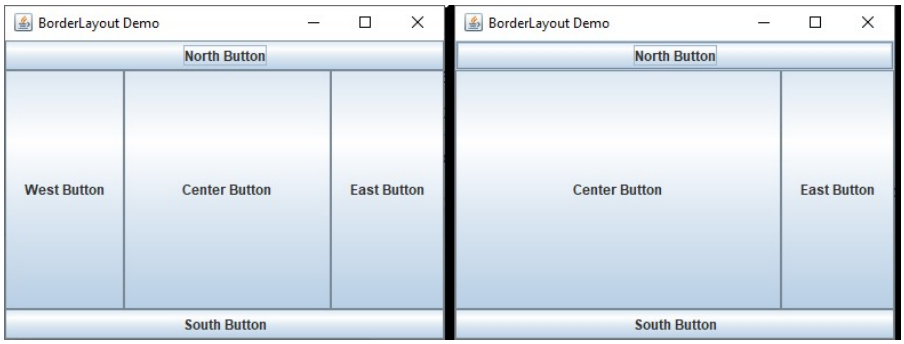
Εικόνα 41 - η συμπεριφορά του BoxLayout για κατακόρυφη και οριζόντια στοίχιση

- Ο `BorderLayout` είναι ένας διαχειριστής διάταξης από τους πιο συνηθισμένους. Διαχωρίζει το κοντέινερ σε πέντε περιοχές και τοποθετεί τα συστατικά στις πέντε θέσεις μέσα στον κοντέινερ (π.χ. στο παράθυρο). North & South για τοποθέτηση στο πάνω και κάτω μέρος αντίστοιχα. East & West για τοποθέτηση στο δεξί και αριστερό μέρος αντίστοιχα. Center για τοποθέτηση στο κέντρο, καταλαμβάνοντας όλο τον χώρο που απομένει.

Στον κώδικα που ακολουθεί, το `BorderLayout` οργανώνει τα συστατικά στο παράθυρο σε πέντε περιοχές και κάθε κουμπί τοποθετείται στην αντίστοιχη περιοχή του layout με τη χρήση της μεθόδου `add()` και της καθορισμένης περιοχής. Στην Εικόνα 42 βλέπουμε το αποτέλεσμα της εκτέλεσης του κώδικα (αριστερά) και δεξιά το αποτέλεσμα όπου στον κώδικα έχουμε παραλείψει το West Button (δείτε πώς το κέντρο καταλαμβάνει όλο τον απομένοντα χώρο).

```
import javax.swing.*;
import java.awt.*;

public class BorderLayoutDemo {
      public static void main(String[] args) {
            JFrame frame = new JFrame("BorderLayout Demo");
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            frame.setSize(400, 300);

            frame.setLayout(new BorderLayout());

            // Προσθήκη κουμπιών στις πέντε διαφορετικές περιοχές
            frame.add(new JButton("North Button"), BorderLayout.NORTH);
            frame.add(new JButton("South Button"), BorderLayout.SOUTH);
            frame.add(new JButton("East Button"), BorderLayout.EAST);
```

```
            frame.add(new JButton("West Button"), BorderLayout.WEST);
            frame.add(new JButton("Center Button"), BorderLayout.CENTER);

            frame.setVisible(true);
      }
}
```



Εικόνα 42 - η διάταξη συστατικών που παρέχει ο BorderLayout

- Ο `CardLayout` παρουσιάζει μόνο ένα συστατικό κάθε φορά. Σκεφτείτε, για παράδειγμα ένα παράθυρο με καρτέλες που κάθε φορά θα εμφανίζεται μια από αυτές.

Στον κώδικα που ακολουθεί, προσθέτουμε ένα `JPanel mainPanel` στο παράθυρο όπου χρησιμοποιούμε τον `CardLayout myCardLayout`, που έχουμε δημιουργήσει νωρίτερα. Στη συνέχεια δύο ακόμα `JPanel`, `card1` και `card2` με ένα κουμπί στο πρώτο και ένα `JTexArea` στο δεύτερο. Προσθέτουμε τα `card1` και `card2` στο `mainPanel` και αυτό στο παράθυρο. Με την εντολή `myCardLayout.show()` μπορούμε να εμφανίζουμε κάθε φορά την κάρτα που θέλουμε (συνήθως ανταποκρινόμενοι σε κάποιο συμβάν, π.χ. πάτημα πλήκτρου, ή ανάλογα με την επιλογή του χρήστη σε ένα πτυσσόμενο κουτί επιλογών, `JComboBox`).

```
import javax.swing.*;
import java.awt.*;

public class CardLayoutDemo {
      public static void main(String[] args) {
            JFrame frame = new JFrame("CardLayout Demo");
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

            CardLayout myCardLayout = new CardLayout();
            JPanel mainPanel = new JPanel(myCardLayout);

            // Κάρτα με κουμπί
            JPanel card1 = new JPanel();
            card1.add(new JButton("Button"));

            // Κάρτα με JTextArea
            JPanel card2 = new JPanel();
            card2.add(new JTextArea(5, 20));

            mainPanel.add(card1, "Card 1");
            mainPanel.add(card2, "Card 2");

            frame.add(mainPanel);

            // Εμφάνιση της πρώτης κάρτας
            myCardLayout.show(mainPanel, "Card 1");
```

```
            // Εμφάνιση της δεύτερης κάρτας
            // myCardLayout.show(mainPanel, "Card 2");

            frame.setSize(300, 200);
            frame.setVisible(true);
      }
}
```

Μια εναλλακτική δυνατότητα για να έχουμε καρτέλες σε ένα παράθυρο εφαρμογής, αντί για τον `CardLayout` manager, είναι να χρησιμοποιήσουμε το συστατικό `JTabbedPane` του Swing, τη χρήση του οποίου θα δούμε σε παράδειγμα, παρακάτω.

Θα δούμε τώρα μερικά παραδείγματα που χρησιμοποιούν τις σχετικές κλάσεις, κατασκευαστές και μεθόδους τους, πάντα για τη βιβλιοθήκη Swing.

9.3.2. JFrame, JPanel, JLabel

Ο κώδικας που ακολουθεί, δημιουργεί ένα παράθυρο και μέσα σε αυτό προσθέτει τέσσερα panel. Χρησιμοποιεί έναν `GridLayout` manager για να στοιχίσει σε δύο γραμμές και δύο στήλες τα panel μέσα στο παράθυρο:

```
frame.setLayout(new GridLayout(2, 2));
```

Μπορούμε να δημιουργήσουμε το κάθε panel χωριστά, π.χ. `panel1`, `panel2`, κ.λπ. Ωστόσο, είναι πολύ πιο εύχρηστη η ομαδική διαχείριση με τη βοήθεια π.χ. ενός πίνακα από συστατικά, στην περίπτωση εδώ, ενός πίνακα από panel:

```
JPanel[] panel = new JPanel[4];
```

Αφού δημιουργήσουμε τον πίνακα, επαναληπτικά σε κάθε θέση του, δημιουργούμε ένα αντικείμενο `JPanel`, του δίνουμε χρώμα υπόβαθρου (μόνο για επίδειξη στο παράδειγμα, γενικά δεν συνηθίζεται), και σε κάθε ένα προσθέτουμε ένα αντικείμενο `JLabel`, μια ετικέτα δηλαδή με κείμενο (δεν δίνουμε όνομα στην ετικέτα, εναλλακτικά, μπορούμε πρώτα να την δημιουργήσουμε και μετά να την προσθέσουμε):

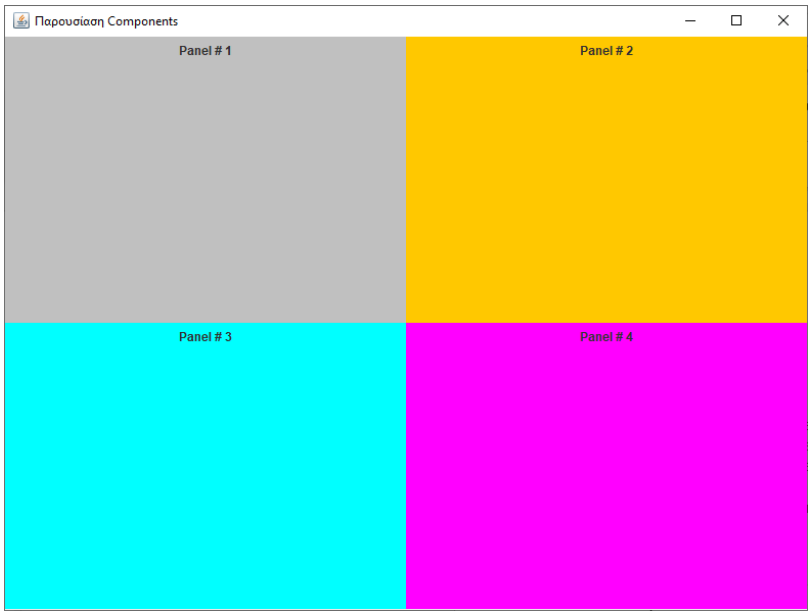
```
panel[i].add(new JLabel("Panel # " + (i + 1)));
```

Παρατηρήστε ότι για τα χρώματα δημιουργήσαμε έναν πίνακα αντικειμένων της κλάσης `Color` που περιέχεται στο πακέτο `java.awt`, δίνοντας τιμές από τα σχετικά (`static`) πεδία που επίσης περιέχει η κλάση.

```
import java.awt.Color;
import java.awt.GridLayout;

import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;

public class Swing4 {
      public static void main(String[] arguments) {
            JFrame frame = new JFrame("Παρουσίαση Components");
            // Οργάνωση σε δύο σειρές με δύο panels ανά σειρά
            frame.setLayout(new GridLayout(2, 2));
            // Δημιουργία ΠΙΝΑΚΑ που θα αποθηκεύσει τέσσερα JPanel
```

```
            JPanel[] panel = new JPanel[4];
            // Πίνακας Colors για καθορισμό χρώματος φόντου σε κάθε JPanel
(για επίδειξη!)
            Color[] colors = { Color.LIGHT_GRAY, Color.ORANGE, Color.CYAN,
Color.MAGENTA };

            for (int i = 0; i < panel.length; i++) {
                  panel[i] = new JPanel(); // Δημιουργία των panel
                  panel[i].setBackground(colors[i]);  //  Ορισμός  χρώματος
φόντου
                  panel[i].add(new  JLabel("Panel  #  "  +  (i  +  1)));  //
Προσθήκη ετικέτας στο panel
                  frame.add(panel[i]); // Προσθήκη panel στο πλαίσιο
            }
            frame.setSize(800, 600);
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            frame.setVisible(true);
      }
}
```

Η Εικόνα 43 παρουσιάζει το αποτέλεσμα της εκτέλεσης του κώδικα.



Εικόνα 43 - ένα frame με τέσσερα panels

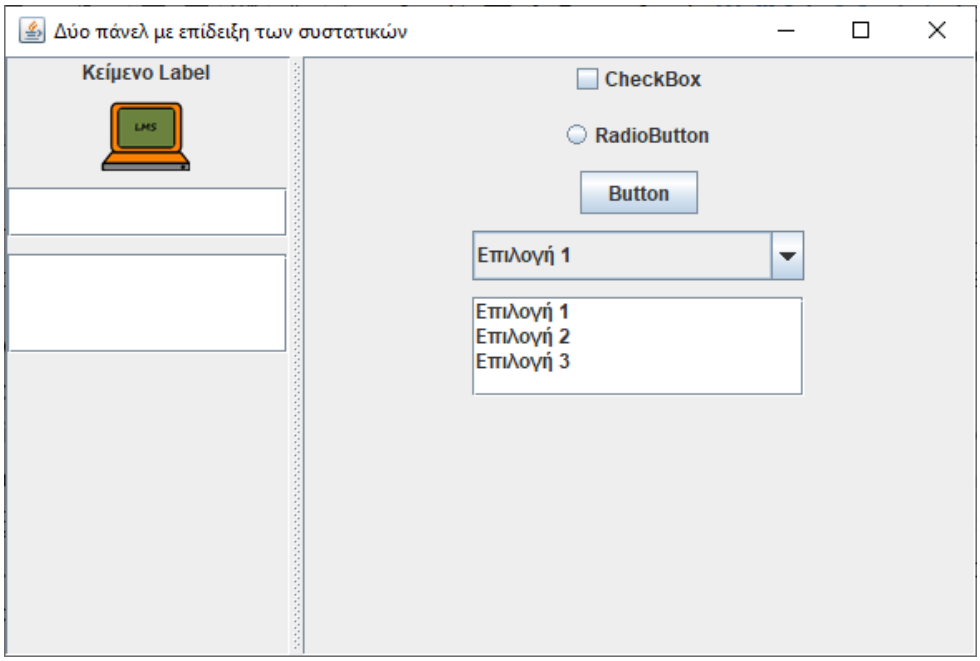
Είναι προφανές ότι ένα περιβάλλον GUI αποκτά αξία όταν μπορεί να ανταποκρίνεται σε γεγονότα, που κατά κύριο λόγο δημιουργεί ο χρήστης. Θα δούμε τη σχετική ύλη στην επόμενη ενότητα.

9.3.3. JLabel, JTextField, JTextArea, JChekBox, JRadioButton, JButton, JComboBox, JList

Στο παράδειγμα που ακολουθεί, χωρίζουμε το παράθυρο κατακόρυφα σε δύο panels σε αναλογία 40-60. Στο αριστερό πάνελ προσθέτουμε ένα `JLabel` με κείμενο και ένα με εικόνα, ένα `JTextField`, και ένα `JTextArea`. Στο δεξί πάνελ προσθέτουμε από ένα `JCheckBox`, `JRadioButton`, `JButton`, `JComboBox` και `JList`.

Ο διαχωρισμός σε αναλογία γίνεται με χρήση ενός `JSplitPane`, ως διαχειριστή διάταξης χρησιμοποιούμε τον `BoxLayout`, ενώ για να αφήσουμε κενό ανάμεσα στα συστατικά

χρησιμοποιούμε τη μέθοδο `Box.createVerticalStrut()`. Στην Εικόνα 44 μπορείτε να δείτε το αποτέλεσμα της εκτέλεσης του κώδικα που παρουσιάζει όλα τα βασικά συστατικά. Παρακάτω εξηγούμε τις λεπτομέρειες του κώδικα.



Εικόνα 44 - δύο πάνελ με αναλογία 40-60 και τα κυριότερα συστατικά

```
import javax.swing.*;
import java.awt.*;

public class Swing5 {
      public static void main(String[] args) {
            JFrame frame = new JFrame("Δύο πάνελ με επίδειξη των
συστατικών");
            frame.setSize(600, 400);
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

            // Δημιουργία του αριστερού panel
            JPanel leftPanel = new JPanel();
            leftPanel.setLayout(new                    BoxLayout(leftPanel,
BoxLayout.Y_AXIS));

            JLabel textLabel = new JLabel("Κείμενο Label");
            textLabel.setAlignmentX(Component.CENTER_ALIGNMENT);         //
Κεντράρισμα

            JLabel        imageLabel        =        new        JLabel(new
ImageIcon("data/image1.png")); // Εικόνα που (πρέπει να) υπάρχει στον
υποφάκελο data του τρέχοντος φακέλου
            imageLabel.setAlignmentX(Component.CENTER_ALIGNMENT);

            JTextField textField = new JTextField(15); // 15 στήλες
            textField.setMaximumSize(new   Dimension(200,   30));   //   max
μέγεθος
            textField.setAlignmentX(Component.CENTER_ALIGNMENT);

            JTextArea  textArea  =  new  JTextArea(3,  15);  //  Εμφάνιση  3
γραμμών
```

```
            textArea.setLineWrap(true);
            textArea.setWrapStyleWord(true);
            textArea.setMaximumSize(new Dimension(200, 60)); // Μέγεθος
            textArea.setAlignmentX(Component.CENTER_ALIGNMENT);

            JScrollPane textAreaScroll = new JScrollPane(textArea);
            textAreaScroll.setMaximumSize(new   Dimension(200,   60));   //
Μέγεθος

            // Προσθήκη των components στο αριστερό panel
            leftPanel.add(textLabel);
            // Κενό μεταξύ των components
            leftPanel.add(Box.createVerticalStrut(10));
            leftPanel.add(imageLabel);
            leftPanel.add(Box.createVerticalStrut(10));
            leftPanel.add(textField);
            leftPanel.add(Box.createVerticalStrut(10));
            leftPanel.add(textAreaScroll);

            // Δημιουργία του δεξιού panel
            JPanel rightPanel = new JPanel();
            rightPanel.setLayout(new                  BoxLayout(rightPanel,
BoxLayout.Y_AXIS));

            JCheckBox checkBox = new JCheckBox("CheckBox");
            checkBox.setAlignmentX(Component.CENTER_ALIGNMENT);

            JRadioButton radioButton = new JRadioButton("RadioButton");
            radioButton.setAlignmentX(Component.CENTER_ALIGNMENT);

            JButton button = new JButton("Button");
            button.setAlignmentX(Component.CENTER_ALIGNMENT);

            JComboBox<String> comboBox = new JComboBox<>(new String[] {
"Επιλογή 1", "Επιλογή 2", "Επιλογή 3" });
            comboBox.setMaximumSize(new Dimension(200, 30));
            comboBox.setAlignmentX(Component.CENTER_ALIGNMENT);
            JList<String> list = new JList<>(new String[] { "Επιλογή 1",
"Επιλογή 2", "Επιλογή 3" });
            list.setVisibleRowCount(3); // Εμφάνιση 3 γραμμών
            list.setFixedCellHeight(30);
            list.setFixedCellWidth(150);
            list.setAlignmentX(Component.CENTER_ALIGNMENT);

            JScrollPane listScroll = new JScrollPane(list);
            listScroll.setMaximumSize(new Dimension(200, 60));

            // Προσθήκη των components στο δεξιό panel
            rightPanel.add(checkBox);
            rightPanel.add(Box.createVerticalStrut(10));
            rightPanel.add(radioButton);
            rightPanel.add(Box.createVerticalStrut(10));
            rightPanel.add(button);
            rightPanel.add(Box.createVerticalStrut(10));
            rightPanel.add(comboBox);
            rightPanel.add(Box.createVerticalStrut(10));
            rightPanel.add(listScroll);

            // Δημιουργία διαχωριστικού JSplitPane με αναλογία 40-60
```

```
            JSplitPane splitPane = new
JSplitPane(JSplitPane.HORIZONTAL_SPLIT, leftPanel, rightPanel);
            splitPane.setDividerLocation(0.4); // Χωρίζει το frame σε 40-60
αναλογία

            // Προσθήκη του splitPane στο frame
            frame.add(splitPane);
            frame.setVisible(true);
      }
}
```

Ο παραπάνω κώδικας, αρχικά δημιουργεί το παράθυρο, καθορίζει το μέγεθός του, καθώς και τη συμπεριφορά του όταν το κλείσουμε (στο παράδειγμα γίνεται τερματισμός της εφαρμογής).

Στη συνέχεια, δημιουργούμε το αριστερό πάνελ και καθορίζουμε ως διαχειριστή διάταξης τον `BoxLayout`. Η παράμετρος `BoxLayout.Y_AXIS` σηματοδοτεί στοίχιση το ένα κάτω από το άλλο (άξονας Υ):

```
leftPanel.setLayout(new BoxLayout(leftPanel, BoxLayout.Y_AXIS));
```

Δημιουργούμε ένα `JLabel` με κείμενο `"Κείμενο Label"` και του καθορίζουμε οριζόντιο κεντράρισμα μέσα στο πάνελ. Παρόμοια, δημιουργούμε ένα δεύτερο `JLabel` που αυτό περιέχει μια εικόνα (δεν ελέγχουμε αν υπάρχει στον φάκελο που αναφέρεται, το θεωρούμε δεδομένο). Ως όρισμα στο `JLabel` μπαίνει ένα αντικείμενο `javax.swing.ImageIcon` όπου ένας κατασκευαστής της κλάσης αυτής δέχεται το πλήρες όνομα του αρχείου με την εικόνα:

```
JLabel imageLabel = new JLabel(new ImageIcon("data/image1.png"))
```

Δημιουργούμε ένα `JTextField` με 15 στήλες, το κεντράρουμε και του ορίζουμε μέγιστο μέγεθος.

Δημιουργούμε ένα `JTextArea` με 3 γραμμές και 15 στήλες, το κεντράρουμε, ενεργοποιούμε την αναδίπλωση κειμένου σε επίπεδο λέξης (και όχι χαρακτήρα) και καθορίζουμε μέγιστο μέγεθος.

Καθώς έχουμε ορίσει μέγιστο μέγεθος για το `JTextArea`, για την περίπτωση που χρειαστεί περισσότερο χώρο το ενθυλακώνουμε σε ένα αντικείμενο `JScrollPane` που του δίνει τη δυνατότητα εμφάνισης ράβδων κύλισης (scroll bars):

```
JScrollPane textAreaScroll = new JScrollPane(textArea);
```

Αφού δημιουργήσαμε όλα τα συστατικά, τα προσθέτουμε στο αριστερό πάνελ (κατακόρυφα, λόγω του διαχειριστή διάταξης `BoxLayout`), τοποθετώντας ανάμεσά τους κενό με χρήση της μεθόδου `javax.swing.Box.CreateVerticalStrut()` που προσθέτει ένα μη ορατό συστατικό με ύψος που καθορίζεται από το όρισμα.

Ανάλογα, δημιουργούμε το δεξί πάνελ και τα σχετικά συστατικά και τα προσθέτουμε σε αυτό.

Το `JComboBox` δημιουργείται ώστε να εμφανίζει τρεις επιλογές, ως εξής:

```
JComboBox<String> comboBox = new JComboBox<>(new String[] { "Επιλογή 1",
"Επιλογή 2", "Επιλογή 3" });
```

Παρόμοια το `Jlist`:

```
JList<String> list = new JList<>(new String[] { "Επιλογή 1", "Επιλογή 2",
"Επιλογή 3" });
```

Στην τελευταία περίπτωση, καθορίζουμε πόσες γραμμές θα είναι ορατές, καθώς και το μέγεθος των κελιών της λίστας (της κάθε γραμμής). Και εδώ της δίνουμε τη δυνατότητα προσθήκης ράβδων κύλισης, εφόσον χρειαστεί.

Με το `JSplitPane` χωρίζουμε το παράθυρο σε δύο πάνελ (αριστερό και δεξί) και καθορίζουμε τη θέση του διαχωριστή (divider) ώστε το αριστερό πάνελ να καταλαμβάνει το 40% του παραθύρου και το δεξί το 60%:

```
JSplitPane splitPane = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT,
leftPanel, rightPanel);

splitPane.setDividerLocation(0.4);
```

Τέλος, προσθέτουμε το `JSplitPane` στη φόρμα και την καθιστούμε ορατή.

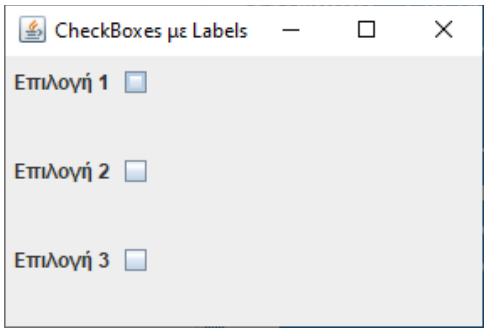
9.3.4. Παράδειγμα ομάδας από JCheckBox

Στο επόμενο παράδειγμα θα δούμε τη δημιουργία πολλαπλών (τριών) `JCheckBox` όπου ο χρήστης μπορεί να επιλέξει ανεξάρτητα ένα ή περισσότερα. Δημιουργούμε ένα `mainPanel` στο οποίο χρησιμοποιούμε ένα `BoxLayout` για κατακόρυφη στοίχιση των περιεχομένων του. Στη συνέχεια για κάθε ζεύγος `JLabel` και `JCheckBox` (κείμενο επεξήγησης και αντίστοιχο κουτί επιλογής) δημιουργούμε από ένα `JPanel` με διαχειριστή `FlowLayout`, προκειμένου να τοποθετηθούν δίπλα. Επιπρόσθετα καθορίζουμε αριστερή στοίχιση με `FlowLayout.LEFT`. Τα `JLabel` και `JCheckBox` τοποθετούνται σε πίνακα που, όπως έχουμε πει, μας δίνει πιο ξεκάθαρο κώδικα και μεγαλύτερη ευελιξία στη διαχείριση των συστατικών.

```
import javax.swing.*;
import java.awt.*;

public class Swing6 {
      public static void main(String[] args) {
            JFrame frame = new JFrame("CheckBoxes με Labels");
            frame.setSize(300, 200);
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
// Δημιουργία κύριου panel
            JPanel mainPanel = new JPanel();
            // Κάθετη στοίχιση για τα ζεύγη label-checkbox
            mainPanel.setLayout(new                    BoxLayout(mainPanel,
BoxLayout.Y_AXIS));
            // Πίνακας JCheckBox
            JCheckBox[] checkBox = new JCheckBox[3];
            // Πίνακας JLabel
            JLabel[] label = new JLabel[3];

            // Δημιουργία ζευγών με JLabel και JCheckBox
            for (int i = 0; i <= 2; i++) {
                  // Στοίχιση προς τα αριστερά
                  JPanel       groupPanel       =       new       JPanel(new
FlowLayout(FlowLayout.LEFT));
                  label[i] = new JLabel("Επιλογή " + (i + 1));
                  checkBox[i] = new JCheckBox();
                  // Προσθήκη των components στο panel
                  groupPanel.add(label[i]);
                  groupPanel.add(checkBox[i]);
                  // Προσθήκη στο κύριο panel
```

```
                mainPanel.add(groupPanel);
            }
            frame.add(mainPanel);
            frame.setVisible(true);
      }
}
```

Στην Εικόνα 45 βλέπουμε το αποτέλεσμα εκτέλεσης του κώδικα.



Εικόνα 45 - checkboxes και αντίστοιχα labels

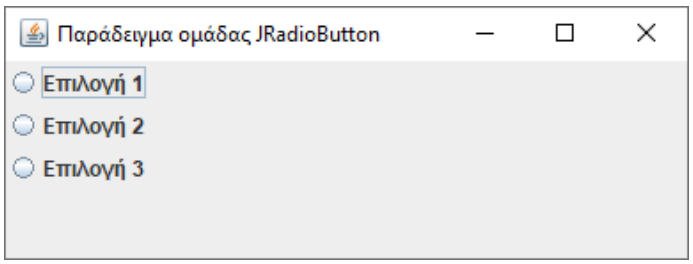
9.3.5. Διαχείριση ομάδας από JRadioButton

Θα δούμε τώρα ένα παράδειγμα διαχείρισης των `JRadioButton` που συνήθως χρησιμοποιούνται για την επιλογή μόνο ενός ανάμεσα σε μια ομάδα από αυτά. Η ομαδοποίηση γίνεται με τη χρήση της κλάσης `ButtonGroup`, η οποία μας εξασφαλίζει ότι μόνο ένα `JRadioButton` μπορεί να επιλεχθεί κάθε φορά από τον χρήστη.

```
import javax.swing.*;

public class Swing7 {
      public static void main(String[] args) {
            JFrame frame = new JFrame("Παράδειγμα ομάδας JRadioButton");
            frame.setSize(300, 150);
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            // Ένα panel
            JPanel panel = new JPanel();
            // Κάθετη διάταξη
            panel.setLayout(new BoxLayout(panel, BoxLayout.Y_AXIS));
            // Τρία radio buttons
            JRadioButton radioButton1 = new JRadioButton("Επιλογή 1");
            JRadioButton radioButton2 = new JRadioButton("Επιλογή 2");
            JRadioButton radioButton3 = new JRadioButton("Επιλογή 3");

            // Ομαδοποίηση των RadioButtons
            ButtonGroup group = new ButtonGroup();
            group.add(radioButton1);
            group.add(radioButton2);
            group.add(radioButton3);

            // Προσθήκη στο panel
            panel.add(radioButton1);
            panel.add(radioButton2);
            panel.add(radioButton3);
            // Προσθήκη του panel στο παράθυρο
            frame.add(panel);
            frame.setVisible(true);
```

```
	}
}
```

Το αποτέλεσμα εκτέλεσης του κώδικα το βλέπουμε στην Εικόνα 46.



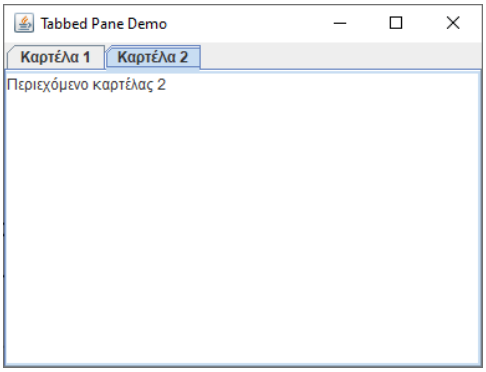
Εικόνα 46 - μια ομάδα από αλληλοαποκλειόμενα JRadioButton

9.3.6. Εφαρμογή πολλών καρτελών με JTabbedPane

Για να φτιάξετε εφαρμογή με καρτέλες που εναλλάσσονται με κλικ στο πάνω μέρος τους, μπορούμε να χρησιμοποιήσουμε το `JTabbedPane` της Java (αντί για τον `CardLayout` Manager που έχουμε ήδη περιγράψει παραπάνω). Το `JTabbedPane` επιτρέπει την προσθήκη καρτελών, και κάθε φορά που ο χρήστης κλικάρει σε μία καρτέλα, εμφανίζεται το αντίστοιχο περιεχόμενο.

Στον παρακάτω κώδικα, δημιουργούμε ένα `JTabbedPane` και προσθέτουμε δύο καρτέλες, με ενδεικτικό περιεχόμενο από ένα `JtextArea` στην κάθε μια. Αν εκτελέσουμε τον κώδικα (Εικόνα 47) θα δούμε ότι ο χρήστης μπορεί να περνά από τη μία καρτέλα στην άλλη χωρίς να χρειαστεί να έχουμε γράψει κώδικα χειρισμού κάποιου συμβάντος, όπως στην περίπτωση του διαχειριστή `CardLayout`.

```
import javax.swing.*;

public class TabbedPaneDemo {
	public static void main(String[] args) {
		JFrame frame = new JFrame("Tabbed Pane Demo");
		frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		frame.setSize(400, 300);

		// Δημιουργία του TabbedPane
		JTabbedPane tabbedPane = new JTabbedPane();

		// Δημιουργία καρτελών
		tabbedPane.addTab("Καρτέλα 1", new JTextArea("Περιεχόμενο
καρτέλας 1"));
		tabbedPane.addTab("Καρτέλα 2", new JTextArea("Περιεχόμενο
καρτέλας 2"));

		// Προσθήκη JTabberPane στο JFrame
		frame.add(tabbedPane);

		frame.setVisible(true);
	}
}
```



Εικόνα 47 - παράδειγμα καρτελών με την κλάση JTabbedPane

Με όσα είδαμε ως εδώ, πρέπει να έχετε πάρει μια καλή ιδέα για το πώς χειριζόμαστε τα συνηθισμένα συστατικά στοιχεία του Swing, πώς καθορίζουμε το μέγεθος και χρησιμοποιούμε ράβδο κύλισης αν δεν επαρκέσει, πώς τα διατάσσουμε με χρήση (απλών) διαχειριστών διάταξης, πώς δημιουργούμε διαχωρισμένα πάνελ, πώς ομαδοποιούμε τα radio buttons, πώς δημιουργούμε καρτέλες με διαφορετικά συστατικά στην καθεμιά. Στην επόμενη ενότητα θα δούμε τον χειρισμό συμβάντων που συσχετίζει τις ενέργειες του χρήστη με τα συστατικά, αφού όμως πρώτα παρουσιάσουμε μια σύνοψη των κλάσεων των Components.

Παρακάτω παρουσιάζονται μερικά από τα κυριότερα χαρακτηριστικά των πιο συνηθισμένων κλάσεων Components, που είδαμε νωρίτερα. Αν θέλουμε τη λεπτομερή και αναλυτική περιγραφή των συστατικών, όλους τους κατασκευαστές και τις μεθόδους, και γενικά τις πλήρεις δυνατότητες, θα πρέπει να επισκεφθούμε την τεκμηρίωση του Java API.

9.3.7. Συνοπτικά χαρακτηριστικά των κυριότερων συστατικών στοιχείων

JFrame

Κατασκευαστές

`JFrame(String title)` Δημιουργεί ένα παράθυρο με τον τίτλο του ορίσματος. Αν το όρισμα είναι κενό, το παράθυρο δημιουργείται χωρίς τίτλο.

Μέθοδοι

`setSize(int width, int height)` Ορίζει το μέγεθος του παραθύρου.

`setDefaultCloseOperation(int operation)` Καθορίζει τη συμπεριφορά κατά το κλείσιμο (π.χ. `JFrame.EXIT_ON_CLOSE`).

`setVisible(boolean flag)` Εμφανίζει ή κρύβει το παράθυρο, ανάλογα με την τιμή του ορίσματος.

`setLayout(LayoutManager manager)` Καθορίζει τον Διαχειριστή Διάταξης με βάση τον οποίο θα τοποθετηθούν τα συστατικά. Με όρισμα `null` δεν χρησιμοποιείται κανένας και είναι ευθύνη του προγραμματιστή να τοποθετήσει τα συστατικά.

JPanel

Κατασκευαστές

`JPanel()` Δημιουργεί ένα αντικείμενο `JPanel` με την εξ ορισμού διάταξη που σημαίνει ότι τοποθετεί τα συστατικά από αριστερά προς τα δεξιά και από πάνω προς τα κάτω.

`JPanel(LayoutManager l)` Δημιουργεί ένα αντικείμενο `JPanel` με τον Διαχειριστή Διάταξης που δίνεται ως όρισμα.

Μέθοδοι

`add(Component c)` προσθέτει το συστατικό στον κοντέινερ.

`setLayout(LayoutManager l)` Καθορίζει τη διάταξη του κοντέινερ σύμφωνα με το όρισμα που δίνεται.

JLabel

Κατασκευαστές

`JLabel()` Δημιουργεί αντικείμενο χωρίς εικόνα και με κενό string για κείμενο.

`JLabel(Icon image)` Δημιουργεί αντικείμενο με την εικόνα που δίνεται ως όρισμα, με οριζόντια και κατακόρυφη στοίχιση στο κέντρο.

`JLabel(String text)` Δημιουργεί αντικείμενο με το κείμενο που δίνεται ως όρισμα με οριζόντια στοίχιση αριστερά και κατακόρυφη στο κέντρο.

Μέθοδοι

`setIcon(Icon icon)` Καθορίζει την εικόνα που θα εμφανίζει το συστατικό αυτό.

`setText(String text)` Καθορίζει το κείμενο που θα εμφανίζει αυτό το συστατικό.

JButton

Κατασκευαστές

`JButton()` Δημιουργεί ένα αντικείμενο `JButton` (ένα κουμπί) χωρίς να εμφανίζει πάνω του κείμενο ή εικόνα.

`JButton(Icon icon)` Δημιουργεί ένα κουμπί με εικόνα.

`JButton(String text)` Δημιουργεί ένα κουμπί με κείμενο.

`JButton(String text, Icon icon)` Δημιουργεί ένα κουμπί με εικόνα και κείμενο.

Μέθοδοι

`setText(String text)` Ορίζει το κείμενο που εμφανίζεται πάνω στο κουμπί.

`getText()` Επιστρέφει το κείμενο που εμφανίζεται πάνω στο κουμπί.

`addActionListener(ActionListener listener)` Προσθέτει έναν listener που παρακολουθεί τα γεγονότα πατήματος του κουμπιού.

`setEnabled(boolean enabled)` Ενεργοποιεί ή απενεργοποιεί το κουμπί ανάλογα με την τιμή του ορίσματος, καθορίζοντας αν μπορεί να πατηθεί.

`doClick()` Προκαλεί προγραμματιστικά ένα κλικ στο κουμπί, σαν να το έχει πατήσει ο χρήστης.

`setIcon(Icon icon)` Ορίζει ένα εικονίδιο που εμφανίζεται στο κουμπί.

JTextField

Κατασκευαστές

`JTextField()` Δημιουργεί ένα πεδίου κειμένου, κενό.

`JTextField(int columns)` Δημιουργεί ένα κενό πεδίο κειμένου με επιθυμητό μήκος columns.

`JTextField(String text)` Δημιουργεί ένα πεδίο κειμένου με κείμενο που δίνεται ως όρισμα.

`JTextField(String text, int columns)` Δημιουργεί πεδίο κειμένου με κείμενο και επιθυμητό μήκος που δίνονται σαν ορίσματα.

Μέθοδοι

`setText(String text)` Ορίζει το κείμενο που θα εμφανίζεται στο πεδίο κειμένου.

`getText()` Επιστρέφει το κείμενο που περιέχεται στο πεδίο.

`setEditable(boolean editable)` Καθορίζει αν το πεδίο κειμένου είναι επεξεργάσιμο από τον χρήστη ή μόνο για ανάγνωση.

`addActionListener(ActionListener listener)` Προσθέτει έναν listener για τα γεγονότα πληκτρολόγησης (π.χ., όταν ο χρήστης πατάει Enter).

`setColumns(int columns)` Ορίζει τον αριθμό των στηλών που επηρεάζουν το εμφανιζόμενο πλάτος του πεδίου κειμένου.

`selectAll()` Επιλέγει όλο το κείμενο στο πεδίο κειμένου.

`setEnabled(boolean enabled)` Ενεργοποιεί ή απενεργοποιεί το πεδίο κειμένου, καθορίζοντας αν μπορεί να το επεξεργαστεί ο χρήστης.

JTextArea

Κατασκευαστές

`JTextArea()` Δημιουργεί ένα αντικείμενο, κενό.

`JTextArea(int rows, int columns)` Δημιουργεί ένα κενό αντικείμενο, με αριθμό γραμμών και στηλών όπως καθορίζεται από τα ορίσματα.

`JTextArea(String text)` Δημιουργεί ένα αντικείμενο που περιέχει το κείμενο του ορίσματος.

`JTextArea(String text, int rows, int columns)` Δημιουργεί αντικείμενο με περιεχόμενο, γραμμές και στήλες σύμφωνα με τα ορίσματα.

Μέθοδοι

`setText()`, `getText()`, `setEditable()`, `selectAll()` με ανάλογη λειτουργία όπως στην κλάση `JTextField`.

`append(String str)` Προσθέτει κείμενο στο τέλος του ήδη υπάρχοντος κειμένου.

JCheckBox

Κατασκευαστές

`JCheckBox(String text)` Δημιουργεί ένα πλαίσιο ελέγχου με κείμενο και μη επιλεγμένο.

`JCheckBox(String text, boolean selected)` Δημιουργεί ένα πλαίσιο ελέγχου με κείμενο και επιλεγμένο ή όχι, ανάλογα με την τιμή του ορίσματος `selected`.

Μέθοδοι

`isSelected()` Επιστρέφει `true` αν `to JCheckBox` είναι επιλεγμένο, διαφορετικά επιστρέφει `false`.

`setSelected(boolean selected)` Ορίζει αν το `JCheckBox` θα είναι επιλεγμένο (`true`) ή αποεπιλεγμένο (`false`).

JRadioButton

Κατασκευαστές

`JRadioButton(String text)` Δημιουργεί ένα `JRadioButton` με το συγκεκριμένο κείμενο.

`JRadioButton(String text, boolean selected)` Δημιουργεί ένα `JRadioButton` με το συγκεκριμένο κείμενο και καθορίζει αν είναι επιλεγμένο ή όχι.

Μέθοδοι

`isSelected()` Επιστρέφει `true` αν το `JRadioButton` είναι επιλεγμένο, αλλιώς `false`.

`setSelected(boolean selected)` Ορίζει αν το `JRadioButton` θα είναι επιλεγμένο ή όχι.

JComboBox

Κατασκευαστές

`JComboBox(E[] items)` Δημιουργεί ένα `JComboBox` και το γεμίζει με τα στοιχεία από τον πίνακα items.

Μέθοδοι

`addItem(E item)` Προσθέτει ένα στοιχείο στο JComboBox.

`removeItem(Object item)` Αφαιρεί ένα στοιχείο από το JComboBox.

`removeAllItems()` Αφαιρεί όλα τα στοιχεία από το JComboBox.

`getSelectedItem()` Επιστρέφει το τρέχον επιλεγμένο στοιχείο.

`getItemAt(int index)` Επιστρέφει το στοιχείο στη συγκεκριμένη θέση.

`getItemCount()` Επιστρέφει τον αριθμό των στοιχείων στο `JComboBox`.

`setEditable(boolean editable)` Καθορίζει αν το `JComboBox` είναι επεξεργάσιμο από τον χρήστη (ο χρήστης μπορεί να πληκτρολογήσει μια τιμή).

`isEditable()` Επιστρέφει true αν το `JComboBox` είναι επεξεργάσιμο.

`setMaximumRowCount(int count)` Ορίζει το μέγιστο αριθμό των γραμμών που εμφανίζονται όταν το `JComboBox` ανοίγει με το κλικ του χρήστη.

JList

Κατασκευαστές

`JList(E[] listData)` Δημιουργεί ένα `JList` και το γεμίζει με τα στοιχεία από τον πίνακα `listData`.

Μέθοδοι

`setListData(E[] listData)`: Ορίζει τα στοιχεία της λίστας από έναν πίνακα.

`getSelectedValue()`: Επιστρέφει το πρώτο επιλεγμένο στοιχείο στη λίστα.

`getSelectedValuesList()`: Επιστρέφει μια λίστα με όλα τα επιλεγμένα στοιχεία.

`getSelectedIndex()`: Επιστρέφει το δείκτη του πρώτου επιλεγμένου στοιχείου.

`getSelectedIndices()`: Επιστρέφει έναν πίνακα με τους δείκτες όλων των επιλεγμένων στοιχείων.

`setSelectedIndex(int index)`: Ορίζει ένα συγκεκριμένο στοιχείο ως επιλεγμένο.

`setSelectedIndices(int[] indices)`: Ορίζει πολλαπλά στοιχεία ως επιλεγμένα.

`clearSelection()`: Καθαρίζει την επιλογή από τη λίστα.

`setVisibleRowCount(int count)`: Ορίζει πόσες γραμμές εμφανίζονται όταν η λίστα δεν είναι πλήρως εμφανής.

`setSelectionMode(int mode)`: Καθορίζει τον τρόπο επιλογής (π.χ. μονή επιλογή, πολλαπλή συνεχόμενη, πολλαπλή διάσπαρτη).

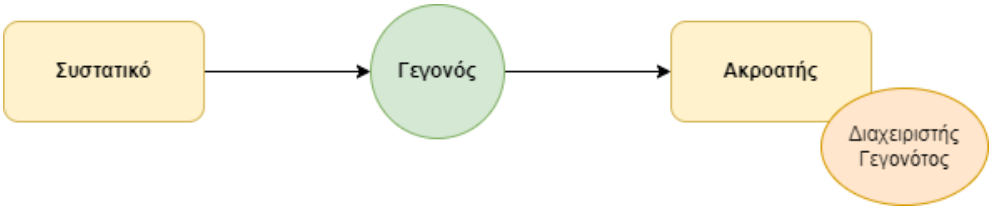
9.4. Event-Driven Programming και Χειρισμός Συμβάντων (Event-Handling)

Όπως έχουμε πει και παραπάνω, σε ένα πρόγραμμα που χρησιμοποιεί GUI, οι χρήστες αλληλεπιδρούν με την εφαρμογή μέσω των στοιχείων ελέγχου (components). Κάθε φορά που ο χρήστης αλληλεπιδρά με αυτά τα στοιχεία, δημιουργούνται συμβάντα (events), για παράδειγμα κάνει κλικ σε ένα κουμπί, επιλέγει από μια λίστα, αλλάζει την επιλογή σε ένα πτυσσόμενο μενού, κ.λπ. Τα συμβάντα αυτά πρέπει να ανιχνευθούν και να υποστούν επεξεργασία και διαχείριση από

την εφαρμογή προκειμένου να πυροδοτηθούν οι αντίστοιχες ενέργειες, για παράδειγμα το κλικ να αποθηκεύει κάποια δεδομένα.

Στη Java, ο χειρισμός αυτών των συμβάντων γίνεται μέσω του μηχανισμού Event-Handling, ο οποίος βασίζεται στην έννοια του Listener (Ακροατής). Πρόκειται για αντικείμενα που "ακούνε", εντοπίζουν συμβάντα/γεγονότα από πηγές (συστατικά) και ενεργοποιούν κατάλληλες μεθόδους διαχειριστές γεγονότος (Εικόνα 48). Για παράδειγμα, όταν ο χρήστης κάνει κλικ σε ένα κουμπί (=πηγή), το σύστημα ανιχνεύει το γεγονός και εκτελεί την αντίστοιχη ενέργεια/μέθοδο μέσω του κατάλληλου ακροατή.

Στην ενότητα αυτή θα δούμε τους κυριότερους τύπους συμβάντων και ακροατών, καθώς και τον τρόπο συγγραφής των μεθόδων χειρισμού συμβάντων.



Εικόνα 48 – η ροή χειρισμού γεγονότων

9.4.1. ActionEvent

Ας ξεκινήσουμε με έναν από τους πλέον συνηθισμένους τύπους συμβάντων, το `ActionEvent`. Ένα `ActionEvent` είναι μια κλάση που αντιπροσωπεύει ένα συμβάν το οποίο δημιουργείται όταν ο χρήστης κάνει μια ενέργεια, π.χ. το πάτημα ενός `JButton`, την επιλογή στοιχείου σε `JComboBox`, το πάτημα του enter σε ένα `JTextField`. Το αντικείμενο `ActionEvent` που δημιουργείται περιέχει πληροφορίες για το συμβάν, όπως την πηγή του (το συστατικό που το προκάλεσε), κ.ά.

Όταν ένα συστατικό (π.χ. ένα `JButton`) δημιουργήσει ένα αντικείμενο `ActionEvent` (διότι πατήθηκε), το `ActionListener` που έχει συσχετιστεί με το συστατικό λαμβάνει αυτό το συμβάν και εκτελεί τη μέθοδο διαχειριστή γεγονότος `actionPerformed(ActionEvent e)`. Η τελευταία υλοποιείται από τον προγραμματιστή για να καθορίσει τι θα κάνει το πρόγραμμα ως αντίδραση στο πάτημα του κουμπιού.

Οι μέθοδοι χειρισμού συμβάντων μπορούν να γραφτούν είτε ως ανώνυμες κλάσεις (που το είδαμε στο δεύτερο παράδειγμα του παρόντος κεφαλαίου) είτε ως ξεχωριστές κλάσεις για καλύτερη οργάνωση του κώδικα.

Θα ξεκινήσουμε με ένα παράδειγμα ενός πολύ συχνά εμφανιζόμενου `ActionEvent` που δημιουργείται με το πάτημα ενός πλήκτρου. Η διαδικασία εντοπισμού και χειρισμού του είναι ως εξής:

- Δημιουργούμε μια ξεχωριστή κλάση `ButtonActionHandler` που υλοποιεί το interface `ActionListener`. Εκεί περιλαμβάνουμε τη μέθοδο `actionPerformed()`, που ο κώδικάς της καθορίζει τι θέλουμε να συμβεί όταν πατιέται το κουμπί.

- Συνδέουμε αυτή την κλάση με το κουμπί. Αυτό γίνεται με την μέθοδο `addActionListener()` της κλάσης `JButton` που της περνάμε ένα αντικείμενο της κλάσης που φτιάξαμε.

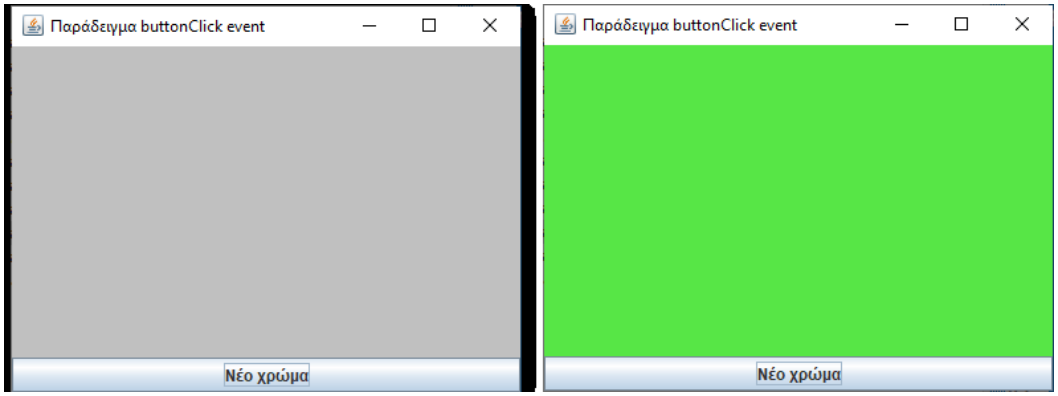
Στον κώδικα που ακολουθεί, κάθε φορά που ο χρήστης κάνει κλικ στο κουμπί που βρίσκεται στο κάτω μέρος του παράθυρου, το υπόλοιπο μέρος αλλάζει το χρώμα του φόντου του με κάποιο τυχαίο χρώμα.

Για το σκοπό αυτό δημιουργούμε δύο συστατικά, ένα panel και ένα κουμπί. Χρησιμοποιούμε τον διαχειριστή διάταξης `BorderLayout` για να βάλουμε το κουμπί στο κάτω μέρος (`SOUTH`) και το panel στο κέντρο (`CENTER`). Όπως έχουμε πει, το κεντρικό συστατικό θα πιάσει όλο τον χώρο που απομένει.

Η κλάση `ButtonActionHandler` έχει κατασκευαστή που δέχεται ένα panel και η μέθοδος actionPerformed υπολογίζει ένα τυχαίο νέο χρώμα RGB και το εφαρμόζει στο panel. Βλέπουμε δείγμα εκτέλεσης του κώδικα στην Εικόνα 49. Σε κάθε κλικ, το μέρος πάνω από το κουμπί αλλάζει χρώμα φόντου.

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

//Η ξεχωριστή κλάση για το χειρισμό του ActionEvent
class ButtonActionHandler implements ActionListener {
private JPanel panel;

 // Κατασκευαστής που παίρνει το panel για να του αλλάξει χρώμα
 public ButtonActionHandler(JPanel panel) {
     this.panel = panel;
 }

 @Override
 public void actionPerformed(ActionEvent e) {
     // Αλλαγή χρώματος του panel σε τυχαίο χρώμα
     panel.setBackground(new Color(
             (int)(Math.random() * 255),
             (int)(Math.random() * 255),
             (int)(Math.random() * 255)
     ));
 }
}

public class Swing8 {
    public static void main(String[] args) {
        // Δημιουργία του frame
        JFrame frame = new JFrame("Παράδειγμα buttonClick event");
        frame.setSize(400, 300);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setLayout(new BorderLayout());

        // Δημιουργία panel που θα αλλάζει χρώμα
        JPanel colorPanel = new JPanel();
        colorPanel.setBackground(Color.LIGHT_GRAY);

        // Δημιουργία κουμπιού
```

```
        JButton changeColorButton = new JButton("Νέο χρώμα");

        // Δημιουργία του ActionListener με ξεχωριστή κλάση, περνάμε το
colorPanel
        ButtonActionHandler btnListener = new
ButtonActionHandler(colorPanel);

        // Προσθήκη του ActionListener στο κουμπί
        changeColorButton.addActionListener(btnListener);

        // Προσθήκη των components στο frame
        frame.add(colorPanel, BorderLayout.CENTER);
        frame.add(changeColorButton, BorderLayout.SOUTH);

        // Εμφάνιση του frame
        frame.setVisible(true);
    }
}
```



Εικόνα 49 - αντιδρώντας στο ActionEvent «κλικ σε κουμπί»

9.4.2. MouseEvent

Ένας δεύτερος, συνηθισμένος τύπος συμβάντων, είναι το `MouseEvent`. Αυτό αφορά ενέργειες όπως το πάτημα ή η απελευθέρωση ενός πλήκτρου του ποντικιού, η μετακίνηση του ποντικιού πάνω από ένα συστατικό, κ.λπ. Ένα `MouseEvent` είναι μια κλάση που αντιπροσωπεύει ένα συμβάν που δημιουργείται όταν ο χρήστης εκτελεί μια ενέργεια με το ποντίκι, όπως το κλικ, η μετακίνηση, το πάτημα ή η απελευθέρωση ενός κουμπιού του ποντικιού. Το αντικείμενο `MouseEvent` που δημιουργείται περιέχει πληροφορίες για το συμβάν, όπως την πηγή του (το συστατικό στο οποίο συνέβη), τη θέση του ποντικιού, και το είδος της ενέργειας που έγινε (π.χ. πάτημα ή απελευθέρωση κουμπιού).

Όταν ένα συστατικό (π.χ. ένα `JPanel`) δημιουργήσει ένα `MouseEvent` (π.χ. επειδή το ποντίκι κινήθηκε πάνω του), ένας `MouseListener` που έχει συσχετιστεί με το συστατικό λαμβάνει αυτό το συμβάν και εκτελεί τη σχετική μέθοδο διαχείρισης γεγονότων, π.χ. `mouseClicked(MouseEvent e)`, κ.λπ. Αυτές οι μέθοδοι υλοποιούνται από τον προγραμματιστή για να καθορίσουν τι θα κάνει το πρόγραμμα σε αντιδράσεις, όπως το κλικ ή η μετακίνηση του ποντικιού.

```
import javax.swing.*;
import java.awt.event.*;

// Κλάση που υλοποιεί το MouseListener
```

```
class MouseEventHandler implements MouseListener {
// Για να μετράμε τα γεγονότα που πιάνει ο ακροατής
	static int x = 0;

	// Πρέπει υποχρεωτικά να υλοποιηθούν και οι πέντε παρακάτω μέθοδοι
	@Override
// Κλικ στο ποντίκι
	public void mouseClicked(MouseEvent e) {
		x++;
		System.out.println(x + " Mouse Click!");
	}

	@Override
// Πίεση πλήκτρου ποντικιού
	public void mousePressed(MouseEvent e) {
		x++;
		System.out.println(x + " Mouse Down!");
	}

	@Override
// Άφεση πλήκτρου ποντικιού
	public void mouseReleased(MouseEvent e) {
		x++;
		System.out.println(x + " Mouse Up!");
	}

	@Override
// Είσοδος σε περιοχή συστατικού
	public void mouseEntered(MouseEvent e) {
		x++;
		System.out.println(x + " Mouse entered component!");
	}

	@Override
// Έξοδος από περιοχή συστατικού
	public void mouseExited(MouseEvent e) {
		x++;
		System.out.println(x + " Mouse exited component!");
	}
}

public class Swing10 {
	public static void main(String[] args) {
		JFrame frame = new JFrame("Παράδειγμα με Mouse Event Handler");
		frame.setSize(400, 300);
		frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

		JPanel panel = new JPanel();
		// Αντιστοίχιση του MouseEventHandler
		panel.addMouseListener(new MouseEventHandler());

		frame.add(panel);
		frame.setVisible(true);
	}
}
```

Το παραπάνω παράδειγμα δεν υλοποιεί κάποια ουσιαστική λειτουργικότητα, οι πέντε μέθοδοι έχουν υλοποιηθεί ώστε απλά να εμφανίζουν σχετικό μήνυμα στην κονσόλα για το ποιο γεγονός συμβαίνει κάθε φορά. Ο μετρητής `x` απλά αυξάνει κατά ένα κάθε φορά μετρώντας τα συμβάντα.

Χρησιμοποιώντας την `MouseListener`, είμαστε υποχρεωμένοι να υλοποιήσουμε και τις πέντε μεθόδους που βλέπουμε στην κλάση `MouseEventHandler`.

Αν θέλουμε να υλοποιήσουμε μόνο μέρος αυτών, υπάρχει η βοηθητική κλάση `MouseAdapter`, που με παρόμοιο κώδικα όπως παραπάνω μας επιτρέπει να υλοποιήσουμε μόνον αυτές που θέλουμε. Επίσης, για συμβάντα που σχετίζονται με την κίνηση του ποντικιού (π.χ. σύρσιμο, drag) υπάρχει η κλάση `MouseMotionListener`.

9.4.3. KeyEvent

Το `KeyEvent` είναι μια κλάση που αντιπροσωπεύει ένα συμβάν που δημιουργείται όταν ο χρήστης αλληλεπιδρά με το πληκτρολόγιο (πατάει ή απελευθερώνει πλήκτρα του πληκτρολογίου).

Στο παρακάτω παράδειγμα, χρησιμοποιούμε έναν `KeyListener` για να ακούμε τα γεγονότα πληκτρολόγησης και να μετατρέπουμε σε κεφαλαία τα γράμματα που εισάγονται σε ένα `JTextField`.

```
import javax.swing.*;
import java.awt.event.*;

// Δημιουργία κλάσης χειρισμού για τα KeyEvents
class UpperCaseKeyHandler extends KeyAdapter {
	@Override
	public void keyTyped(KeyEvent e) {
		// Μετατροπή του χαρακτήρα που πληκτρολογήθηκε σε κεφαλαίο
		char keyChar = e.getKeyChar();
		if (Character.isLetter(keyChar)) {
			e.setKeyChar(Character.toUpperCase(keyChar));
		}
	}
}

public class Swing11 {
	public static void main(String[] args) {
		JFrame frame = new JFrame("Παράδειγμα Key Event");
		JTextField textField = new JTextField(20);

		// Προσθήκη του KeyListener χρησιμοποιώντας την κλάση χειρισμού
		textField.addKeyListener(new UpperCaseKeyHandler());

		frame.add(textField);
		frame.setSize(300, 100);
		frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		frame.setVisible(true);
	}
}
```

Η κλάση `UpperCaseKeyHandler` επεκτείνει τον `KeyAdapter`. Η `KeyAdapter` είναι βοηθητική κλάση, στην ίδια λογική με την `MouseAdapter`, που αναφέραμε παραπάνω, και μας επιτρέπει να υλοποιήσουμε όποια μέθοδο θέλουμε και μόνο (διαφορετικά, μπορούμε να χρησιμοποιήσουμε το `KeyListener`, αλλά τότε πρέπει να υλοποιήσουμε όλες τις μεθόδους που περιέχει). Στην περίπτωση του παραδείγματος υλοποιούμε τη μέθοδο `keyTyped()` για να μετατρέπει τους χαρακτήρες σε κεφαλαίους. Στην μέθοδο `main()` το `JTextField` χρησιμοποιεί την κλάση αυτή ως `KeyListener`.

Ας επικεντρώσουμε λίγο την προσοχή μας στη μέθοδο `keyTyped()` που υλοποιούμε. Δέχεται ως όρισμα ένα αντικείμενο `e` τύπου `KeyEvent`. Η κλάση `KeyEvent` διαθέτει τη μέθοδο `getKeyChar()` που επιστρέφει τον χαρακτήρα που πληκτρολογήθηκε και δημιούργησε το `KeyEvent`. Αφού τον ανακτήσουμε, ελέγχουμε αν είναι γράμμα και, αν είναι, χρησιμοποιούμε την μέθοδο `setKeyChar()` της κλάσης `KeyEvent` για να θέσουμε στο `e` την ίδια τιμή, αλλά αφού την μετατρέψουμε σε κεφαλαίο. Με τον τρόπο αυτό, ο χαρακτήρας θα είναι σίγουρα κεφαλαίος προτού εισαχθεί στο `JtextField`.

9.4.4. WindowEvent

Το `WindowEvent` είναι η κλάση που αντιπροσωπεύει τα γεγονότα τα σχετιζόμενα με παράθυρα (Window). Τέτοια γεγονότα παράγονται από αντικείμενα όπως το γνωστό μας `JFrame` (αλλά και από άλλες κλάσεις παραθύρων όπως το `JDialog`) όταν ο χρήστης αλληλεπιδρά με το παράθυρο, για παράδειγμα το άνοιγμα, το κλείσιμο, η ελαχιστοποίηση του παράθυρου.

Για να διαχειριστούμε τέτοιου τύπου γεγονότα χρησιμοποιούμε είτε το interface `WindowListener` οπότε πρέπει να υλοποιήσουμε όλες τις μεθόδους του, είτε την αφηρημένη (abstract) κλάση `WindowAdapter` που μας επιτρέπει να υλοποιήσουμε ακριβώς τις μεθόδους που θέλουμε και μόνον αυτές.

Στο επόμενο παράδειγμα, η κλάση `WindowHandler`, που δημιουργούμε, επεκτείνει το `WindowAdapter` για να υλοποιήσουμε (`@Override`) μόνο την απαραίτητη μέθοδο που χρειαζόμαστε (`windowClosing`). Όταν το παράθυρο πρόκειται να κλείσει, εμφανίζεται ένα μενού διαλόγου `JoptionPane` (θα τα παρουσιάσουμε παρακάτω), που ζητά από τον χρήστη επιβεβαίωση. Εάν ο χρήστης επιλέξει "Yes", καλείται η μέθοδος `dispose()` για να κλείσει το παράθυρο, ακολουθούμενη από `System.exit(0)` για να τερματιστεί η εφαρμογή. Εάν ο χρήστης επιλέξει "No", το παράθυρο παραμένει ανοιχτό.

Παρατηρήστε στο κύριο πρόγραμμα ότι έχουμε ορίσει την εξ ορισμού ενέργεια που θα γίνεται όταν ο χρήστης προσπαθήσει να κλείσει το παράθυρο σε `JFrame.DO_NOTHING_ON_CLOSE`. Αυτό σημαίνει ότι πρέπει να γράψουμε εμείς τον κώδικα κλεισίματος και τερματισμού εφαρμογής, πράγμα που κάνουμε στο χειριστή συμβάντος, αφού πάρουμε επιβεβαίωση από τον χρήστη.

```
import java.awt.event.*;
import javax.swing.*;

class WindowHandler extends WindowAdapter {
    @Override
    public void windowClosing(WindowEvent e) {
        // Παίρνουμε την πηγή του συμβάντος (το JFrame)
        JFrame frame = (JFrame) e.getSource();

        // Εμφάνιση διαλόγου επιβεβαίωσης
        int result = JOptionPane.showConfirmDialog(frame,
                "Θέλετε σίγουρα να εξέλθετε από την εφαρμογή;",
"Επιβεβαίωση",
                JOptionPane.YES_NO_OPTION);

        // Έλεγχος της απάντησης
        if (result == JOptionPane.YES_OPTION) {
            frame.dispose(); // Κλείνει το παράθυρο
```

```
            System.exit(0); // Κλείνει την εφαρμογή
        }
    }
}
public class Swing12 {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Κλείσιμο εφαρμογής με επιβεβαίωση");

        // Προσθήκη του WindowHandler ως χειριστή συμβάντων παραθύρου
        frame.addWindowListener(new WindowHandler());

        frame.setSize(300, 200);
        frame.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);  // Να
μην κλείνει αυτόματα
        frame.setVisible(true);
    }
}
```

Κλείνουμε την παρούσα ενότητα λέγοντας ότι κάποιοι λίγοι Listener υποστηρίζονται από όλα τα συστατικά του Swing, όπως για παράδειγμα ο key listener ή ο mouse listener. Μπορούμε να καταλάβουμε ποια είδη συμβάντων μπορεί να ενεργοποιήσει ένα συστατικό, κοιτάζοντας στην τεκμηρίωση του API τα είδη των ακροατών συμβάντων που μπορούμε να συσχετίσουμε με αυτό. Για παράδειγμα, η κλάση `JComboBox` ορίζει τις μεθόδους `addActionListener`, `addItemListener` και `addPopupMenuListener` (συν βέβαια εκείνες που κληρονομεί από την γονική του κλάση).

Σχετικές πληροφορίες καθώς και μια **συνοπτική λίστα με τα διάφορα συστατικά του Java Swing και τους ακροατές που ο καθένας υποστηρίζει** μπορούμε να βρούμε στη διεύθυνση https://docs.oracle.com/javase/tutorial/uiswing/events/eventsandcomponents.html

9.5. Σύνθετα Συστατικά GUI: Διάλογοι και Μενού

Ήδη σε προηγούμενα παραδείγματα έχουμε χρησιμοποιήσει παράθυρα διαλόγου, για παράδειγμα στον προηγούμενο κώδικα όταν χρειαστήκαμε την επιβεβαίωση του χρήστη για έξοδο από την εφαρμογή. Στην ενότητα αυτή, θα εξετάσουμε τις κύριες κατηγορίες μενού, όπως τα `JMenuBar`, `JMenu`, και `JMenuItem`, καθώς και διάφορους τύπους παραθύρων διαλόγου, όπως τα `JOptionPane` και προσαρμοσμένους διαλόγους. Τα μενού, ως γνωστόν, παρέχουν δυνατότητα ομαδοποίησης επιλογών και ενεργειών σε μια δομή που βελτιώνει την πλοήγηση στην εφαρμογή, ενώ οι διάλογοι χρησιμοποιούνται για την εμφάνιση πληροφοριών, την είσοδο δεδομένων και την παροχή επιβεβαίωσης από τον χρήστη ή ειδοποιήσεων στους χρήστες.

9.5.1. Διάλογοι

Καθώς έχουμε ήδη μια μικρή εμπειρία με την κλάση `JoptionPane` από τις προηγούμενες ενότητες, θα ξεκινήσουμε από αυτήν. Η κλάση μας προσφέρει μεθόδους που δημιουργούν παράθυρα διαλόγου για συχνά εμφανιζόμενες περιπτώσεις, όπως:

1. Διάλογοι μηνύματος (Message Dialogs). Εμφανίζουν στον χρήστη ένα μήνυμα, π.χ. ένα παράθυρο που ενημερώνει για την επιτυχή ολοκλήρωση μιας λειτουργίας (μέθοδος `showMessageDialog()`).

2. Διάλογοι επιβεβαίωσης (Confirm Dialogs). Ζητούν από τον χρήστη να επιβεβαιώσει μια ενέργεια με επιλογές όπως "Yes", "No", ή "Cancel" (μέθοδος `showConfirmDialog()`).
3. Διάλογοι εισαγωγής (Input Dialogs). Επιτρέπουν στον χρήστη να εισάγει κείμενο (μέθοδος `showInputDialog()`).
4. Διάλογοι επιλογών (Option Dialogs). Προσφέρουν στον χρήστη επιλογές που μπορούν να προσαρμοστούν (μέθοδος `showOptionDialog()`).

Στον κώδικα που ακολουθεί κάνουμε μια επίδειξη χρήσης και των τεσσάρων τύπων διαλόγου, χρησιμοποιώντας τις σχετικές μεθόδους της κλάσης `JOptionPane`. Παρατηρήστε ότι δεν χρειάζεται να δημιουργήσουμε κάποιο παράθυρο εφαρμογής (`JFrame`), αλλά μπορούμε να χρησιμοποιήσουμε απευθείας τα σχετικά παράθυρα διαλόγου, οπότε τα μηνύματα εμφανίζονται στο κέντρο της οθόνης. Το γεγονός αυτό το καθορίζουμε δίνοντας την τιμή `null` στο πρώτο όρισμα κάθε μεθόδου. Αν κάποιο παράθυρο διαλόγου ανοίγει στο πλαίσιο κάποιου άλλου συστατικού, π.χ. παράθυρου, τότε στο πρώτο όρισμα βάζουμε αυτό το συστατικό, οπότε το παράθυρο θα ανοίξει μέσα στο γονικό συστατικό.

```
import javax.swing.JOptionPane;

public class Swing14 {
	public static void main(String[] args) {
		// Διάλογος μηνύματος
		JOptionPane.showMessageDialog(null, "Αυτό είναι ένα μήνυμα!",
"Message Dialog", JOptionPane.INFORMATION_MESSAGE);

		// Διάλογος επιβεβαίωσης και έλεγχος απάντησης
		int confirmResult = JOptionPane.showConfirmDialog(null,
"Συμφωνείτε;", "Confirm Dialog", JOptionPane.YES_NO_CANCEL_OPTION);
		if (confirmResult == JOptionPane.YES_OPTION) {
			System.out.println("Ο χρήστης επέλεξε: Yes");
		} else if (confirmResult == JOptionPane.NO_OPTION) {
			System.out.println("Ο χρήστης επέλεξε: No");
		} else if (confirmResult == JOptionPane.CANCEL_OPTION) {
			System.out.println("Ο χρήστης επέλεξε: Cancel");
		}

		// Διάλογος επιβεβαίωσης και διαχείριση απάντησης
		String inputResult = JOptionPane.showInputDialog(null, "Γράψτε
το όνομά σας:", "Input Dialog", JOptionPane.QUESTION_MESSAGE);
		if (inputResult != null) {
			System.out.println("Ο χρήστης έγραψε: " + inputResult);
		} else {
			System.out.println("Ο χρήστης πάτησε Cancel");
		}

		// Διάλογος επιλογών με προσαρμογή
		String[] options = { "Περίπτωση 1", "Περίπτωση 2", "Περίπτωση
3" };
		int optionResult = JOptionPane.showOptionDialog(null,
"Επιλέξτε:", "Option Dialog", JOptionPane.DEFAULT_OPTION,
JOptionPane.QUESTION_MESSAGE, null, options, options[0]);
		if (optionResult != -1) {
			System.out.println("Ο χρήστης επέλεξε: " +
options[optionResult]);
		} else {
```

```
                System.out.println("Ο χρήστης πάτησε Cancel");
            }
        }
}
```

Ο κώδικας εμφανίζει διαδοχικά παράθυρα των τεσσάρων τύπων διαλόγου που προαναφέραμε και ειδικότερα:

1. Διάλογοι μηνύματος

```
JOptionPane.showMessageDialog(null, "Αυτό είναι ένα μήνυμα!", "Message Dialog", JOptionPane.INFORMATION_MESSAGE);
```

Ένα απλό μήνυμα πληροφόρησης. Η μέθοδος `showMessageDialog` έχει ως παραμέτρους:

- `null`: Καθορίζει ότι το διάλογο δεν τον φιλοξενεί κάποιο συγκεκριμένο παράθυρο (άρα κεντρική θέση στην οθόνη).
- Το μήνυμα "Αυτό είναι ένα μήνυμα!" που θα εμφανιστεί.
- Τον τίτλο του παραθύρου "`Message Dialog`".
- Τον τύπο μηνύματος `JOptionPane.INFORMATION_MESSAGE` που εμφανίζει ένα εικονίδιο πληροφόρησης. Υπάρχουν διάφοροι τύποι, όπως `ERROR_MESSAGE`, `WARNING_MESSAGE`, `QUESTION_MESSAGE` κ.λπ. που αλλάζουν την εικόνα που εμφανίζεται στο παράθυρο, δίπλα στο μήνυμα.

2. Διάλογοι επιβεβαίωσης

```
int confirmResult = JOptionPane.showConfirmDialog(null, "Συμφωνείτε;", "Confirm Dialog",JOptionPane.YES_NO_CANCEL_OPTION);
```

Εμφανίζει ένα παράθυρο επιβεβαίωσης με επιλογές Yes, No, Cancel και επιστρέφει την επιλογή του χρήστη:

- `JOptionPane.YES_OPTION`: Αν ο χρήστης επιλέξει Yes.
- `JOptionPane.NO_OPTION`: Αν ο χρήστης επιλέξει No.
- `JOptionPane.CANCEL_OPTION`: Αν ο χρήστης επιλέξει Cancel.

Το αποτέλεσμα αποθηκεύεται στην `confirmResult` και ελέγχεται με τις συνθήκες if-else για να εκτυπωθεί το μήνυμα που αντιστοιχεί στην επιλογή (σε μια πραγματική περίπτωση ο κώδικας πρέπει να διαχειρίζεται τη συνέχεια ανάλογα με την επιλογή του χρήστη, στο παράδειγμα εμείς απλά εμφανίζουμε στην κονσόλα αυτή την επιλογή).

3. Διάλογοι εισαγωγής

```
String inputResult = JOptionPane.showInputDialog(null, "Γράψτε το όνομά σας:", "Input Dialog",JOptionPane.QUESTION_MESSAGE);
```

Εμφανίζει ένα διάλογο εισαγωγής κειμένου, όπου ο χρήστης μπορεί να πληκτρολογήσει κάτι:

- Αν ο χρήστης πληκτρολογήσει κάτι και πατήσει OK, το αποτέλεσμα αποθηκεύεται στη `inputResult`.
- Αν ο χρήστης πατήσει Cancel ή κλείσει το παράθυρο, το `inputResult` είναι `null`.

Στο παράδειγμα, ο κώδικας εμφανίζει στην κονσόλα το κείμενο του χρήστη, ή ένα μήνυμα ακύρωσης αν πάτησε το Cancel.

4. Διάλογοι επιλογών

```
String[] options = { "Περίπτωση 1", "Περίπτωση 2", "Περίπτωση 3" };

int optionResult = JOptionPane.showOptionDialog(null, "Επιλέξτε:", "Option Dialog",  JOptionPane.DEFAULT_OPTION,JOptionPane.QUESTION_MESSAGE,  null, options, options[0]);
```

Εμφανίζει ένα διάλογο επιλογών με προσαρμοσμένες, από τον προγραμματιστή, επιλογές. Αρχικά ορίζουμε έναν πίνακα `options` από αλφαριθμητικά με τις επιλογές που θέλουμε να εμφανιστούν. Στη μέθοδο `showOptionDialog`:

- Η παράμετρος `options` είναι ένας πίνακας με τις διαθέσιμες επιλογές.
- Η μέθοδος επιστρέφει το `index` της επιλεγμένης επιλογής από τον χρήστη ή -1 αν κλείσει το παράθυρο ή πατήσει Cancel, που το αποθηκεύουμε στο `optionResult`.

Αν ο χρήστης επιλέξει κάποια επιλογή, ο δείκτης `optionResult` χρησιμοποιείται στον πίνακα `options` για τον εντοπισμό της επιλογής που έγινε, και στο παράδειγμά μας απλά την εμφανίζουμε στην κονσόλα.

9.5.2. Μενού

Πολλά περιβάλλοντα GUI περιέχουν ένα μενού (ή και περισσότερα, που μπορούν να αλλάζουν δυναμικά καθώς σε ένα `JFrame` κάθε στιγμή ένα μπορεί να είναι το ενεργό), με τα οποία ο χρήστης βρίσκει γρήγορα και εύκολα πρόσβαση στις λειτουργίες της εφαρμογής. Η πλέον συνηθισμένη και de facto καθιερωμένη περίπτωση είναι η γραμμή του μενού να βρίσκεται στο πάνω μέρος του παράθυρου μιας εφαρμογής. Θα δούμε αμέσως τα τρία βασικά συστατικά με τα οποία δημιουργούμε μενού και στη συνέχεια ένα λίγο πιο ολοκληρωμένο παράδειγμα.

`JMenuBar`. Είναι η γραμμή μενού που, όπως είπαμε, συνήθως τοποθετείται στο πάνω μέρος ενός `JFrame`. Ουσιαστικά, είναι το κοντέινερ που περιέχει το μενού.

```
JMenuBar menuBar = new JMenuBar(); // Δημιουργία αντικειμένου

frame.setJMenuBar(menuBar); // Προσθήκη στο JFrame
```

`JMenu`. Είναι το μενού, που τέτοια μπορούμε να προσθέσουμε πολλά σε ένα `JMenuBar`. Ένα `JMenu` μπορεί να περιέχει άλλα `JMenu` ως υπομενού.

```
JMenu fileMenu = new JMenu("Αρχείο"); // Αντικείμενο

menuBar.add(fileMenu); // Προσθήκη του μενού στη γραμμή μενού
```

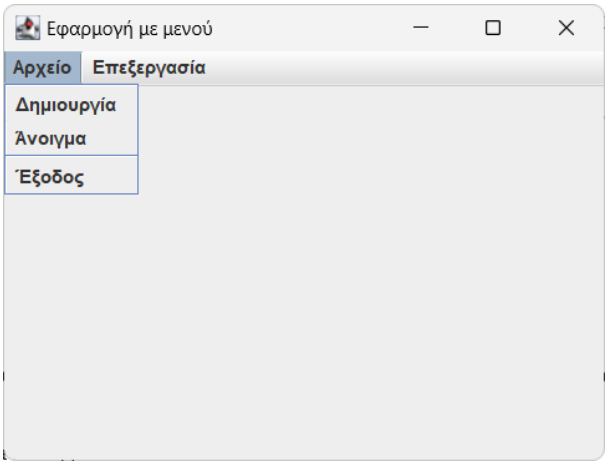
`JMenuItem`. Είναι το στοιχείο του μενού που αντιπροσωπεύει μια επιμέρους επιλογή την οποία μπορεί να επιλέξει ο χρήστης. Προστίθεται σε ένα `JMenu` ως επιλογή.

```
JMenuItem openItem = new JMenuItem("Άνοιγμα"); // Αντικείμενο

fileMenu.add(openItem); // Προσθήκη του στοιχείου στο μενού
```

Στο επόμενο παράδειγμα, δημιουργούμε ένα παράθυρο εφαρμογής `JFrame`, προσθέτουμε ένα `JMenuBar` και σε αυτό δύο αντικείμενα `JMenu` με τίτλο «Αρχείο» και «Επεξεργασία». Στο πρώτο προσθέτουμε τρεις επιλογές `JMenuItem`.

```
import javax.swing.*;

public class Swing15 {
      public static void main(String[] args) {
            // Δημιουργία του frame
            JFrame frame = new JFrame("Εφαρμογή με μενού");
            frame.setSize(400, 300);
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

            // Δημιουργία του JMenuBar
            JMenuBar menuBar = new JMenuBar();

            // Δημιουργία του JMenu
            JMenu fileMenu = new JMenu("Αρχείο");
            JMenu editMenu = new JMenu("Επεξεργασία");
            menuBar.add(fileMenu);
            menuBar.add(editMenu);

            // Δημιουργία στοιχείων JMenuItem
            JMenuItem newItem = new JMenuItem("Δημιουργία");
            JMenuItem openItem = new JMenuItem("Άνοιγμα");
            JMenuItem exitItem = new JMenuItem("Έξοδος");

            // Προσθήκη στοιχείων στο μενού και διαχωριστικού
            fileMenu.add(newItem);
            fileMenu.add(openItem);
            fileMenu.addSeparator(); // Οπτικός διαχωρισμός
            fileMenu.add(exitItem);

            // Προσθήκη της γραμμής μενού στο frame
            frame.setJMenuBar(menuBar);

            frame.setVisible(true);
      }
}
```

Το αποτέλεσμα της εκτέλεσης το βλέπουμε στην Εικόνα 50. Παρατηρήστε τη διαχωριστική γραμμή που προσθέσαμε πριν την τελευταία επιλογή στο μενού «Αρχείο», με χρήση της μεθόδου `JMenu.addSeparator()`.



Εικόνα 50 - δημιουργία μενού σε εφαρμογή GUI

Όπως είναι προφανές, η εμφάνιση του μενού με τις επιλογές του αποκτά αξία όταν συνδέσουμε τα `JMenuItem` με ακροατές γεγονότων (`ActionListener`) που θα χειρίζονται τις ενέργειες που πρέπει να γίνουν ανάλογα με το τι επέλεξε ο χρήστης. Θα δούμε ένα τέτοιο παράδειγμα παρακάτω.

9.6. Σχεδιασμός User Interface με την χρήση Eclipse WindowBuilder Pro

Το Eclipse WindowBuilder Pro, μας δίνει την δυνατότητα οπτικής σχεδίαση GUI για εφαρμογές Java χωρίς να χρειάζεται να γράφουμε απευθείας κώδικα για την τοποθέτηση των συστατικών, τη διάταξή τους κ.λπ. Με αυτό το πρόσθετο του Eclipse τοποθετούμε τα συστατικά στο παράθυρο της εφαρμογής (Σύρε & Άσε) και το μόνο που χρειάζεται είναι να γράψουμε τον κώδικα που θα υλοποιεί τη λειτουργικότητα της συγκεκριμένης εφαρμογής, ανάλογα με τη διάδραση του χρήστη με τα συστατικά (π.χ. κλικ σε κουμπί, επιλογή από μενού, κ.λπ.). Υπάρχουν και άλλες εφαρμογές με παρόμοια λειτουργικότητα, όπως π.χ. τα NetBeans IDE GUI Builder και IntelliJ IDEA GUI Designer, αλλά εμείς θα παραμείνουμε στο πρόσθετο που μας παρέχει το Eclipse.

Αρχικά, θα πρέπει να εγκαταστήσουμε το πρόσθετο στο περιβάλλον μας. Αυτό μπορεί να γίνει μέσα από το Eclipse Marketplace, με αναζήτηση του όρου «WindowBuilder». Κατά την εγκατάσταση θα μας ζητήσει (και πρέπει) να αποδεχτούμε τους όρους χρήσης. Λίγο προσοχή εδώ, θα χρειαστεί να περιμένετε μέχρι να ολοκληρωθεί η εγκατάσταση, που θα το καταλάβουμε όταν μας ζητήσει να επανεκκινήσουμε το Eclipse (εν τω μεταξύ, στο κάτω δεξιά μέρος δείχνει την πρόοδο). Μετά την επανεκκίνηση, τα συνηθισμένα βήματα χρήσης του είναι τα εξής:

- Δημιουργούμε ένα νέο java πρότζεκτ
- Δεξί κλικ στο πρότζεκτ και New>Other..
- WindowBuilder>Swing Designer>JFrame
- Δίνουμε όνομα στην κλάση που θα δημιουργηθεί

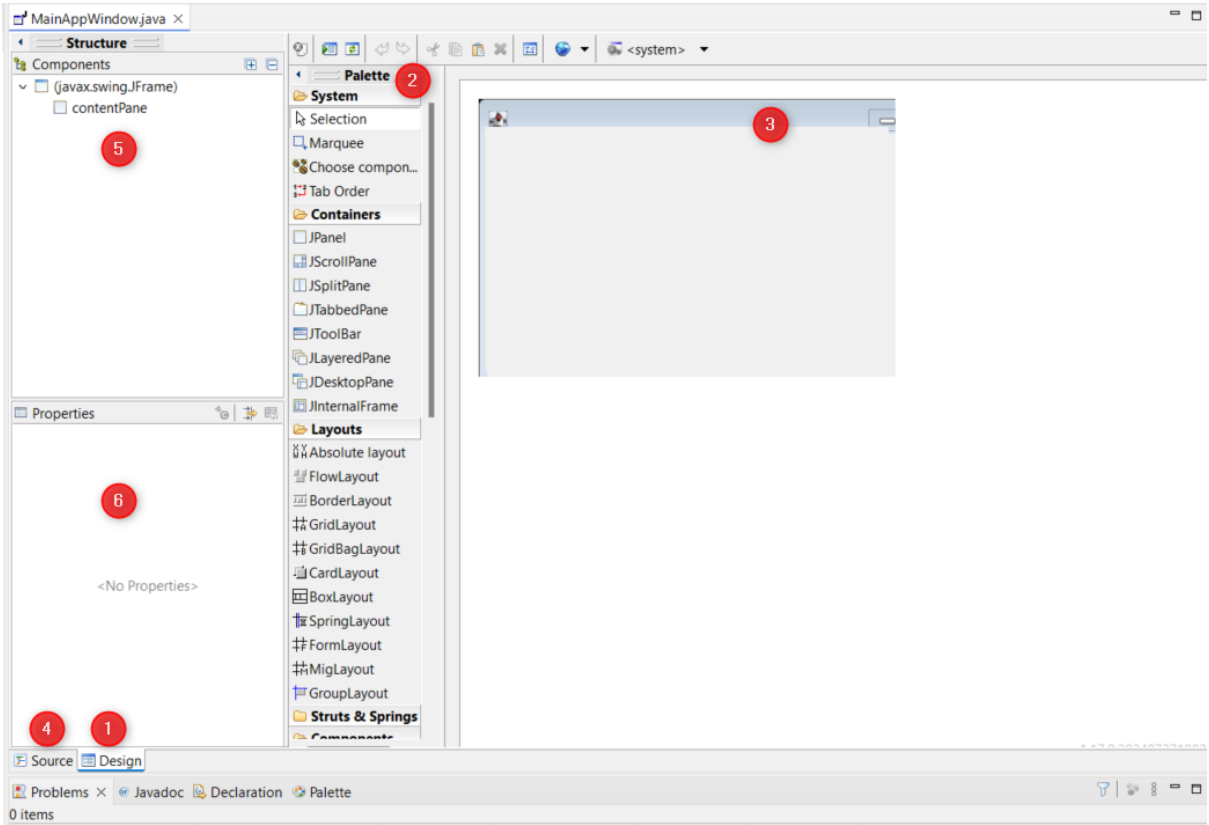
Υπάρχει και η επιλογή Application Window, με γενικά παρόμοια λειτουργικότητα, ωστόσο θα χρησιμοποιήσουμε το `JFrame` για να μείνουμε σε αυτά που έχουμε ήδη μάθει έως τώρα. Ο οδηγός θα σχηματίσει κώδικα που δημιουργεί ένα αντικείμενο `JFrame` (με το όνομα κλάσης που δώσαμε) και επιπλέον εμφανίζει ένα αντικείμενο `JPanel` στο παράθυρο με όνομα `contentPane`.

Μια «λεπτομέρεια», που παραλείψαμε ως εδώ είναι η εξής: όταν δημιουργούμε ένα `JFrame`, αυτό αποκτά δύο συστατικά, ένα χώρο στο πάνω μέρος (κάτω από το όνομα και τα κουμπιά ελαχιστοποίησης, μεγιστοποίησης και κλεισίματος) που εκεί μπορεί να τοποθετηθεί γραμμή μενού και τον υπόλοιπο χώρο παρακάτω που είναι ένα εξ ορισμού `JPanel` (`contentPane`).

Οι εντολές τύπου `frame.setLayout()` αλλάζουν το layout του προεπιλεγμένου contentPane. Όταν προσθέτουμε συστατικά στο `JFrame` με `add()`, αυτά στην πράξη τοποθετούνται στο προεπιλεγμένο contentPane. Δηλαδή, μια εντολή `frame.add(component)` είναι ισοδύναμη με την `frame.getContentPane().add(component)`. Όπως αντιλαμβάνεστε, η `getContentPane()` μας επιστρέφει το προεπιλεγμένο για το `JFrame`. Λέμε προεπιλεγμένο, διότι το εξ ορισμού μπορούμε, αν θέλουμε, να το αλλάξουμε, δημιουργώντας ένα `JPanel` π.χ. με όνομα `myPanel` με τις όποιες ρυθμίσεις μας χρειάζονται, και να αντικαταστήσουμε το εξ ορισμού με αυτό: `frame.setContentPane(myPanel);`

9.6.1. Τα βασικά της επεξεργασίας στο Window Builder

Όταν ανοίξουμε την κλάση στον editor, στο κάτω μέρος θα εμφανιστεί η καρτέλα Design (1). Κάντε κλικ επάνω της και θα δείτε κάτι ανάλογο με την Εικόνα 51. Η παλέττα (Palette) (2) περιλαμβάνει τα διαθέσιμα συστατικά και άλλα εργαλεία (π.χ. διαχειριστές διάταξης) για να χτίσουμε το παραθυρικό περιβάλλον.



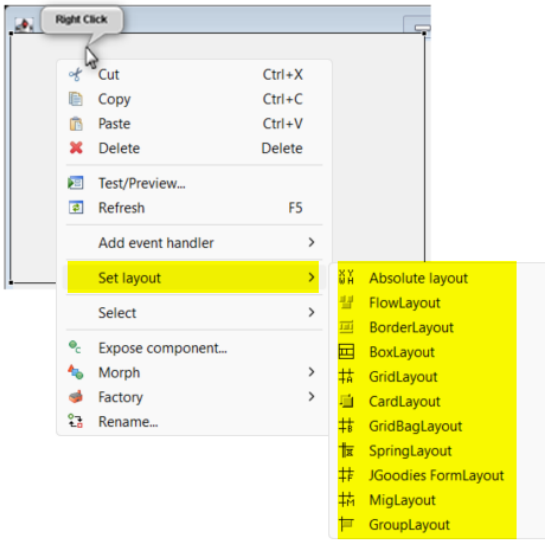
Εικόνα 51 - το περιβάλλον Eclipse WindowBuilder

Στα δεξιά (3) βλέπουμε το παράθυρο της εφαρμογής και μπορούμε, πιάνοντας (τα κατάλληλα για την κάθε περίπτωση) συστατικά από την παλέττα, να τα αποθέσουμε πάνω του. Στη συνέχεια,

μπορούμε να περνάμε στην καρτέλα Source (4) για να επεξεργαστούμε τον σχετικό κώδικα. Στο παράθυρο Components (5) βλέπουμε τα συστατικά που έχουμε χρησιμοποιήσει και κάνοντας κλικ πάνω σε ένα βλέπουμε στο παράθυρο Properties (6) τις σχετικές με αυτό ρυθμίσεις.

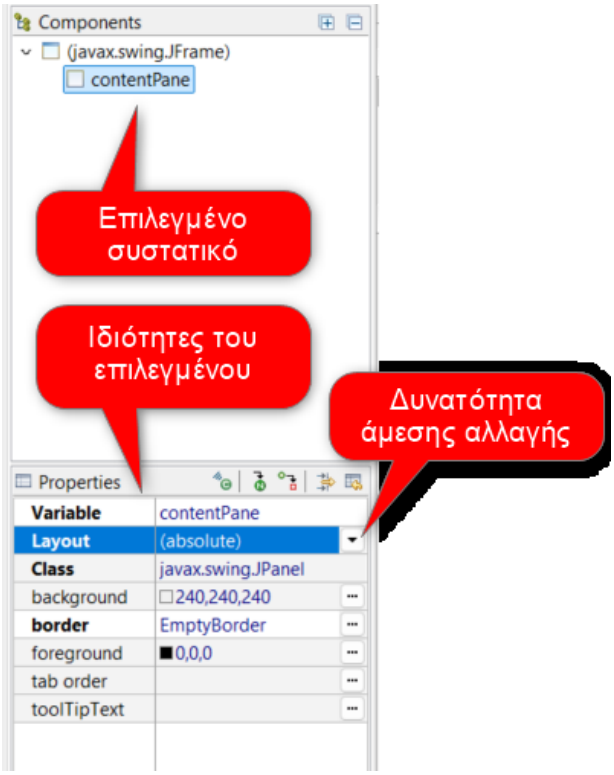
Ας ξεκινήσουμε με ένα απλό παράδειγμα. Θα φτιάξουμε μια απλή φόρμα επικοινωνίας όπου ο χρήστης θα εισάγει το email του και ένα κείμενο και πατώντας το πλήκτρο «Αποστολή» θα «στέλνει» email. Σημειώστε ότι, στην πραγματικότητα, δεν θα αποστέλλεται mail καθώς αυτό απαιτεί ειδικό κώδικα.

Ξεκινάμε καθορίζοντας τον Διαχειριστή Διάταξης. Θα επιλέξουμε το Absolute Layout καθώς είναι το απλούστερο. Στην πράξη, ωστόσο, δεν τον χρησιμοποιούμε καθώς η ακριβής τοποθέτηση έχει πρόβλημα όταν πάμε να αλλάξουμε π.χ. το μέγεθος του παραθύρου, θα χαλάσει η συμμετρία των συστατικών. Ο πιο άμεσος τρόπος καθορισμού διαχειριστή διάταξης είναι να κάνουμε δεξί κλικ πάνω στο παράθυρο και να επιλέξουμε τον επιθυμητό (Εικόνα 52).



Εικόνα 52 - καθορισμός Διαχειριστή Διάταξης

Καθορίζουμε ως διαχειριστή διάταξης το “Absolute layout” και μόλις το επιλέξουμε μπορούμε να δούμε στο παράθυρο Properties (Εικόνα 53) πώς έχει ενημερωθεί η αντίστοιχη ιδιότητα layout σε (absolute). Από εκεί μπορούμε, όποτε το θελήσουμε, να την αλλάξουμε.

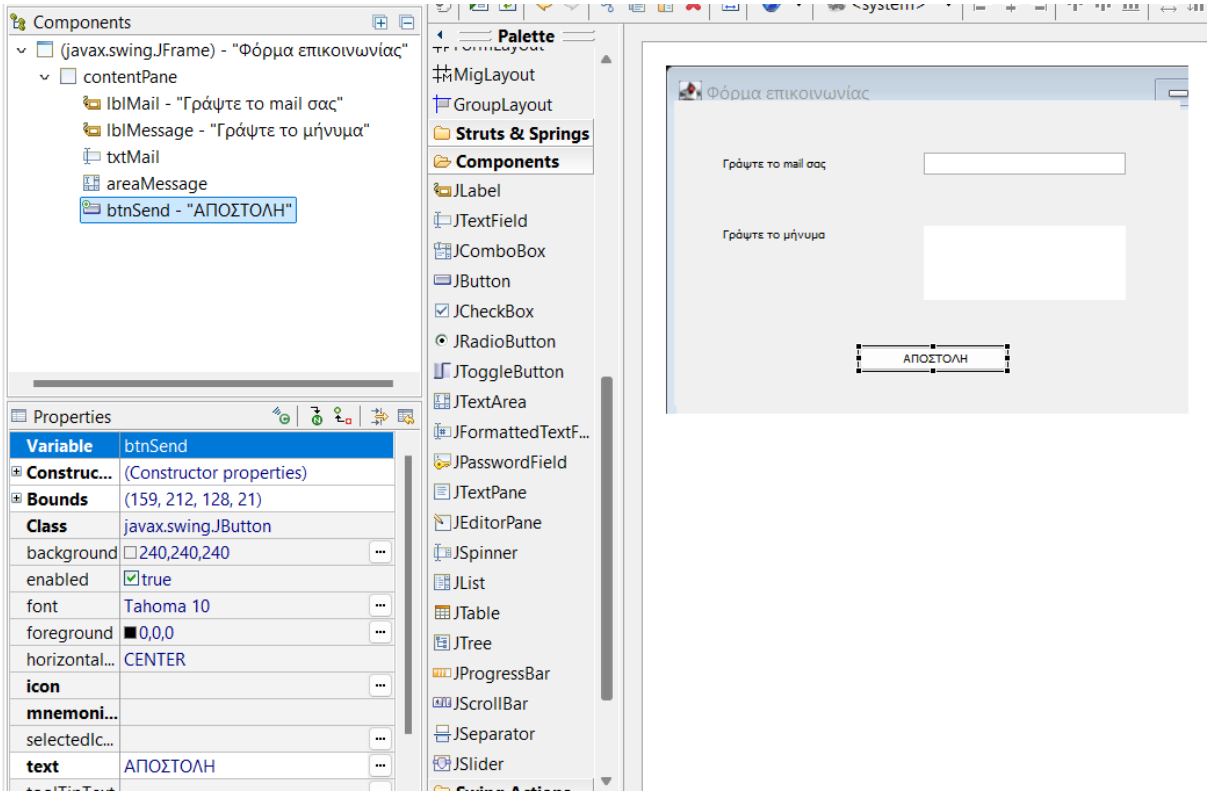


Εικόνα 53 – περιβάλλον Design - συστατικό και ιδιότητες

Στο παράθυρο Components κάνουμε κλικ πάνω στο `JFrame` και στο παράθυρο Properties ρυθμίζουμε την ιδιότητά του title σε «Φόρμα επικοινωνίας». Μόλις φύγουμε από την επεξεργασία (π.χ. κάνοντας κλικ κάπου αλλού), θα δούμε τον τίτλο να περνάει και να ενημερώνει τη φόρμα μας δεξιά.

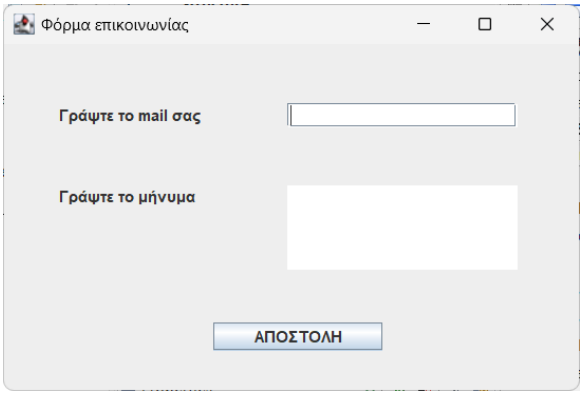
Προσθέτουμε στη φόρμα ένα `JLabel` με το μήνυμα προς τον χρήστη, ένα `JTextField` για την εισαγωγή email, ένα `JTextArea` για την προσθήκη του μηνύματος και ένα `JButton` για την αποστολή των δεδομένων.

Τακτοποιούμε κατάλληλα τα συστατικά στη φόρμα (όποτε εκτελέσετε τον κώδικα, δοκιμάστε να αλλάξετε το μέγεθος του παράθυρου για να δείτε το πρόβλημα της απόλυτης τοποθέτησης). Καθορίζουμε την ιδιότητα `Text` στα `JLabel` και στο `JButton` για να εμφανίζονται τα μηνύματα που θέλουμε. Πάνω στη φόρμα ρυθμίζουμε τα μεγέθη, ώστε το αποτέλεσμα να μας ικανοποιεί (μπορούμε ανά πάσα στιγμή να εκτελέσουμε τον κώδικα). Μπορούμε επίσης να ρυθμίσουμε την ιδιότητα `Variable` σε όλα τα συστατικά όπως θα δηλώναμε το όνομά τους αν γράφαμε κώδικα.



Εικόνα 54 – σχεδίαση μιας εφαρμογής GUI για αποστολή mail

Δείτε στην Εικόνα 54 πώς είναι τώρα το περιβάλλον σχεδίασης και στην Εικόνα 55 το αποτέλεσμα όταν εκτελούμε το πρόγραμμα.



Εικόνα 55 - εκτέλεση εφαρμογής αποστολής mail

Έχουμε καταφέρει να φτιάξουμε μια εφαρμογή με όλα τα συστατικά που θέλουμε, αλλά έχει μια «μικρή» έλλειψη, πατήστε το κουμπί «ΑΠΟΣΤΟΛΗ» και θα δείτε ότι δεν κάνει τίποτε!

Αυτό που χρειάζεται, είναι, όπως ήδη γνωρίζουμε:

- Να προσθέσουμε έναν `ActionListener` στο κουμπί `btnSend` (δείτε στην Εικόνα 54, έτσι έχουμε ονομάσει την ιδιότητα `Variable`, δηλαδή με αυτό το όνομα διαχειριζόμαστε προγραμματιστικά το κουμπί), και

- Να γράψουμε τον κώδικα χειρισμού του γεγονότος «πάτημα πλήκτρου» ώστε να στείλει το μέιλ.

Στο παράδειγμά μας δεν θα στείλουμε μέιλ, αλλά θα χρησιμοποιήσουμε έναν διάλογο για να φανεί ότι γίνεται ο χειρισμός.

Τα ευχάριστα νέα είναι ότι το περιβάλλον μας έχει κάνει ήδη μέρος της δουλειάς. Έχοντας επιλεγμένο συστατικό το κουμπί, κάνουμε κλικ στην καρτέλα Source (στο κάτω μέρος της σχεδίασης, δίπλα στην Design) και θα δούμε ότι έχει προστεθεί αυτόματα ο ακροατής και το μόνο που μας μένει είναι να γράψουμε τον ειδικό κώδικα στη μέθοδο `actionPerformed()`, όπως μπορείτε να δείτε στην Εικόνα 56 (αποστολή μέιλ, αλλά, για το παράδειγμα, ένα απλό μήνυμα διαλόγου).

```
JButton btnSend = new JButton("ΑΠΟΣΤΟΛΗ");
btnSend.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        JOptionPane.showMessageDialog(contentPane, "Το μήνυμά σας εστάλη!");
    }
});
```

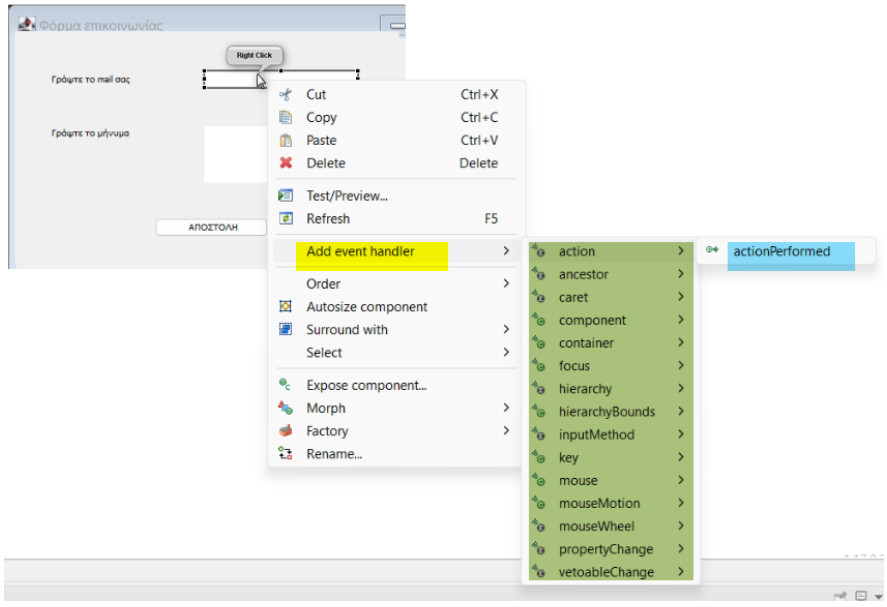
Εδώ ο κώδικας χειρισμού συμβάντος

Εικόνα 56 - εισαγωγή του κώδικα χειρισμού συμβάντος για το κουμπί

Το κλασικό συμβάν από ένα πλήκτρο είναι να πατηθεί (κλικ). Για το λόγο αυτό το περιβάλλον μας είχε ήδη προσθέσει το σχετικό συμβάν και εμείς το επεξεργαστήκαμε.

9.6.2. Προσθήκη διαχειριστή συμβάντος σε συστατικό στον Window Builder

Γενικά, στο γραφικό περιβάλλον του Eclipse Window Builder μπορούμε να προσθέσουμε Διαχειριστή γεγονότος κάνοντας δεξί κλικ πάνω στο συστατικό και από το μενού που θα εμφανιστεί πάμε στην επιλογή “Add event handler” (Εικόνα 57). Σε περίπτωση που δεν το βρούμε, θα πρέπει να το εισαγάγουμε «παραδοσιακά» με πληκτρολόγηση στον κώδικα από την καρτέλα Source.



Εικόνα 57 - προσθήκη διαχειριστή συμβάντος σε περιβάλλον σχεδίασης

Ας υποθέσουμε τώρα ότι θέλουμε να προσθέσουμε έναν έλεγχο εγκυρότητας ώστε, όταν ο χρήστης γράφει το email του, να επιβεβαιώνεται ότι έχει συμπεριλάβει το σύμβολο @. Το σενάριο για το παράδειγμά μας είναι πως, όταν αρχίσει να πληκτρολογεί και μέχρι να εισαγάγει τον χαρακτήρα @, το πλαίσιο κειμένου θα αλλάζει χρώμα (π.χ. ροζ) και θα επανέρχεται σε λευκό αν πληκτρολογηθεί το @.

Σκεφτόμαστε ότι θα πρέπει να προσθέσουμε κάποιον ακροατή στο σχετικό `JTextField` που, καθώς ο χρήστης πληκτρολογεί, θα γίνεται η σχετική διαδικασία ελέγχου. Τι ακροατή όμως; Θα πρέπει να δούμε ποιους υποστηρίζει ένα `JTextBox` και να σκεφτούμε ποιος μας εξυπηρετεί. Για παράδειγμα, το `JTextBox` υποστηρίζει `ActionListener`, αλλά το Action στο οποίο γεννάται αυτό το γεγονός, είναι όταν πατηθεί enter, πράγμα που δεν μας εξυπηρετεί.

Ας δοκιμάσουμε με το `KeyEvent`. Κάνουμε δεξί κλικ πάνω στο `JTextBox txtMail` και στο παράθυρο επιλέγουμε Add event handler>key>key typed. Περνάμε στον κώδικα και συμπληρώνουμε στη μέθοδο `keyTyped` που έχει προστεθεί, τον παρακάτω κώδικα, που πιστεύουμε δεν χρειάζεται να τον επεξηγήσουμε ιδιαίτερα:

```
public void keyTyped(KeyEvent e) {
            // Check if the text contains '@'
            if (!txtMail.getText().contains("@")) {
                 txtMail.setBackground(Color.PINK);
            } else {
                 txtMail.setBackground(Color.WHITE);
            }
      }
```

Όταν εκτελέσετε το πρόγραμμα θα δείτε ότι σε γενικές γραμμές κάνει αυτό που θέλουμε όπως το περιγράψαμε παραπάνω. Ίσως θα ήταν καλύτερα αντί για τη μέθοδο `keyTyped` να υλοποιούσαμε την `keyReleased`. Δοκιμάστε να το κάνετε.

Μια ακόμα καλύτερη λύση θα ήταν να χρησιμοποιήσουμε τον `DocumentListener`. Κάθε κείμενο του `Swing` χρησιμοποιεί ένα αντικείμενο `Document` για να κρατά και διαχειρίζεται το περιεχόμενό του. Τα γεγονότα `Document` επέρχονται όταν το αντικείμενο αλλάζει με οποιονδήποτε τρόπο και σε πραγματικό χρόνο και όχι μόνο στα γεγονότα key όπως ο `KeyListener`. Η προσθήκη του θα γινόταν με έναν κώδικα ως εξής:

```
txtMail.getDocument().addDocumentListener(new DocumentListener() {
          @Override
          public void insertUpdate(DocumentEvent e) {
              checkText();
          }
          @Override
          public void removeUpdate(DocumentEvent e) {
              checkText();
          }
          @Override
          public void changedUpdate(DocumentEvent e) {
              checkText();
          }
          private void checkText() {
              if (!txtMail.getText().contains("@")) {
                  txtMail.setBackground(Color.PINK);
              } else {
                  txtMail.setBackground(Color.WHITE);
```

```
            }
        }
    });
```

Η μέθοδος `getDocument()` επιστρέφει το μοντέλο κειμένου (`Document`) που σχετίζεται με το `JtextField` μας που καταχωρίζουμε το μέιλ. Μέσα από το `Document` αυτό, προσθέτουμε τον `DocumentListener`, ένα interface του οποίου υλοποιούμε τις τρεις μεθόδους στον παραπάνω κώδικα. Και στις τρεις καλούμε την βοηθητική μέθοδο `checkText()`, η οποία κάνει τον έλεγχο και καθορίζει ανάλογα το χρώμα του πεδίου κειμένου. Δοκιμάστε το (αντί για το `KeyEvent`) και θα δείτε ότι δουλεύει καλύτερα.

Ακολουθεί ο πλήρης κώδικας που δημιουργήσαμε με τον Window Builder.

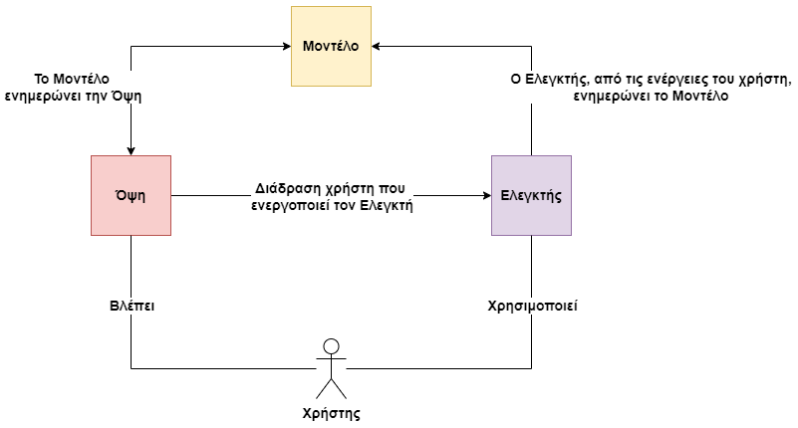
```
import java.awt.Color;
import java.awt.EventQueue;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;
import javax.swing.event.DocumentEvent;
import javax.swing.event.DocumentListener;
import javax.swing.JTextField;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JTextArea;
import javax.swing.JButton;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;

public class MainWindowApp extends JFrame {

	private static final long serialVersionUID = 1L;
	private JPanel contentPane;
	private JTextField txtMail;

	/**
	 * Launch the application.
	 */
	public static void main(String[] args) {
		EventQueue.invokeLater(new Runnable() {
			public void run() {
				try {
					MainWindowApp frame = new MainWindowApp();
					frame.setVisible(true);
				} catch (Exception e) {
					e.printStackTrace();
				}
			}
		});
	}

	/**
	 * Create the frame.
	 */
	public MainWindowApp() {
		setTitle("Φόρμα επικοινωνίας");
		setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		setBounds(100, 100, 450, 300);
		contentPane = new JPanel();
		contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
```

```
(contentPane);
		contentPane.setLayout(null);

		JLabel lblMail = new JLabel("Γράψτε το mail σας");
		lblMail.setBounds(42, 48, 163, 13);
		contentPane.add(lblMail);

		JLabel lblMessage = new JLabel("Γράψτε το μήνυμα");
		lblMessage.setBounds(42, 109, 163, 13);
		contentPane.add(lblMessage);

		txtMail = new JTextField();
		txtMail.getDocument().addDocumentListener(new
DocumentListener() {
			@Override
			public void insertUpdate(DocumentEvent e) {
				checkText();
			}

			@Override
			public void removeUpdate(DocumentEvent e) {
				checkText();
			}

			@Override
			public void changedUpdate(DocumentEvent e) {
				checkText();
			}

			private void checkText() {
				if (!txtMail.getText().contains("@")) {
					txtMail.setBackground(Color.PINK);
				} else {
					txtMail.setBackground(Color.WHITE);
				}
			}
		});

		txtMail.setBounds(215, 45, 174, 19);
		contentPane.add(txtMail);
		txtMail.setColumns(10);

		JTextArea areaMessage = new JTextArea();
		areaMessage.setBounds(215, 108, 174, 64);
		contentPane.add(areaMessage);

		JButton btnSend = new JButton("ΑΠΟΣΤΟΛΗ");
		btnSend.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent e) {
				JOptionPane.showMessageDialog(contentPane,	"Το
μήνυμά σας εστάλη!");
			}
		});
		btnSend.setBounds(159, 212, 128, 21);
		contentPane.add(btnSend);
	}
}
```

9.7. Αρχιτεκτονική Model-View-Controller (MVC)

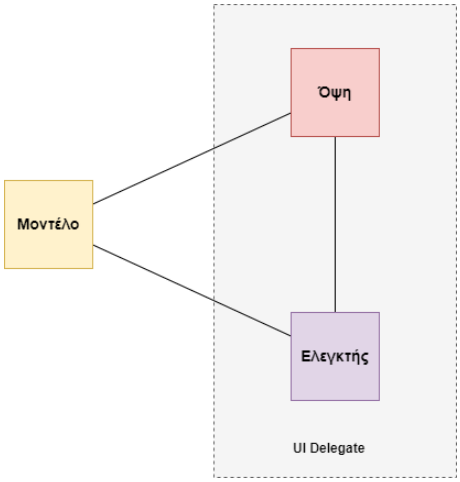
Στην ανάπτυξη πληροφοριακών συστημάτων, το μοντέλο MVC (Model-View-Controller) είναι μια αρχιτεκτονική σχεδίασης που επιτυγχάνει καλύτερη συντήρηση και ευκολότερη ανάπτυξη εφαρμογών. Ο λόγος είναι ότι διαχωρίζει την εφαρμογή σε τρία (κύρια) συστατικά μέρη (Εικόνα 58):

- Μοντέλο (Model). Αντιπροσωπεύει τη **λογική** των δεδομένων της εφαρμογής. Διαχειρίζεται και επεξεργάζεται τα δεδομένα, συνδέεται με τη βάση δεδομένων και φροντίζει για την επεξεργασία τους. Γενικά, το μοντέλο περιέχει τις κλάσεις που αντιπροσωπεύουν τα δεδομένα και τους σχετικούς αλγόριθμους.
- Όψη (View). Είναι υπεύθυνη για την **εμφάνιση** των δεδομένων στον χρήστη. Παίρνει δεδομένα από το μοντέλο και τα εμφανίζει (π.χ. μέσω γραφικού περιβάλλοντος ή ιστοσελίδας). Περιλαμβάνει τον κώδικα που είναι υπεύθυνος για την παρουσίαση των δεδομένων και την αλληλεπίδραση με τον χρήστη.
- Ελεγκτής (Controller). Συνδέει το μοντέλο με την όψη παρέχοντας τον **έλεγχο**. Είναι υπεύθυνος για τη διαχείριση των εισόδων του χρήστη και την επικοινωνία μεταξύ των άλλων δύο τμημάτων. Παίρνει είσοδο από τον χρήστη μέσω της όψης, την επεξεργάζεται και καλεί κατάλληλες λειτουργίες στο μοντέλο για να πραγματοποιηθεί η επιθυμητή ενέργεια. Ο ελεγκτής διαχειρίζεται τα γεγονότα και συντονίζει τη ροή των δεδομένων από το μοντέλο προς την όψη και αντίστροφα.



Εικόνα 58 - μοντέλο MVC στην ανάπτυξη λογισμικού

Ο σχεδιασμός του πακέτου Swing έγινε από τους μηχανικούς της Oracle με το βλέμμα προς το μοντέλο MVC και στόχο να διαχωριστεί η λογική, από την εμφάνιση και τον έλεγχο. Συνάντησαν όμως προβλήματα στην υλοποίηση, κύρια λόγω της στενής σχέσης μεταξύ του ελεγκτή και της όψης. Αυτό έφερε ως αποτέλεσμα την τροποποίηση του σχήματος ώστε οι δύο αυτοί παράγοντες να ενοποιηθούν σε ένα αντικείμενο, το UI delegate object (Εικόνα 59).



Εικόνα 59 - UI delegate: τροποποιημένο MVC για το Swing

Στην πράξη, κάθε συστατικό του Swing περιέχει ένα Model και ένα UI delegate. Το μοντέλο είναι επιφορτισμένο με την διατήρηση πληροφορίας για την κατάσταση (state) του συστατικού. Το UI delegate είναι επιφορτισμένο με την διατήρηση πληροφορίας σχετικά με το πώς θα σχεδιαστεί το συστατικό στην οθόνη (όψη) καθώς και με την αντίδραση στα διάφορα γεγονότα (συμβάντα, events) που δημιουργούνται.

Το τροποποιημένο μοντέλο MVC που χρησιμοποιείται στην ανάπτυξη του Swing αναφέρεται ως αρχιτεκτονική Model-Delegate ή αρχιτεκτονική Separable Model. Η σχεδίαση αυτή αντιμετωπίζει το μοντέλο ενός Component ως ξεχωριστό στοιχείο (όπως στο MVC) αλλά συμπτύσσει την όψη και τον ελεγκτή σε ένα ενιαίο αντικείμενο διεπαφής χρήστη.

Μπορούμε να δούμε ένα παράδειγμα, εξετάζοντας το συστατικό `JTable` που χρησιμοποιείται για την εμφάνιση και επεξεργασία πινάκων δύο διαστάσεων:

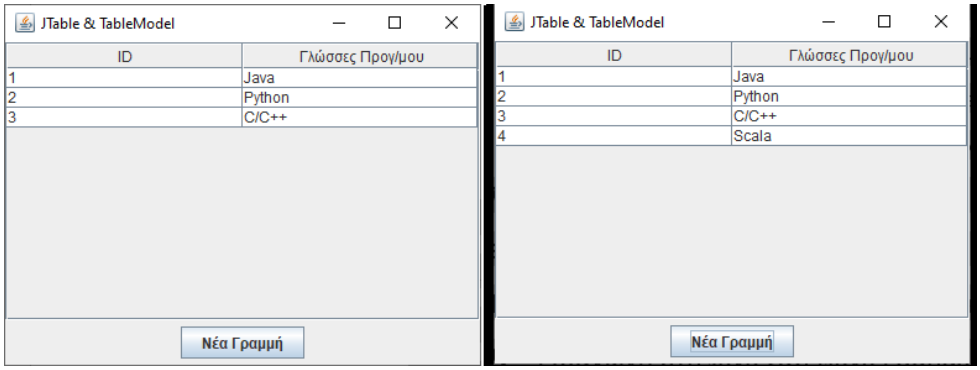
- Model: Στο `JTable`, το μοντέλο είναι το `TableModel`, το οποίο αποθηκεύει δεδομένα και διαχειρίζεται την κατάσταση (state) του πίνακα π.χ. τα περιεχόμενα και τις αλλαγές.
- UI Delegate: Το UI delegate είναι το αντικείμενο που αναλαμβάνει τη σχεδίαση του `JTable` και την αντίδραση σε γεγονότα (π.χ. επιλογή κελιών). Αυτό συνδυάζει τον ρόλο της όψης και του ελεγκτή.

Στον παρακάτω κώδικα, δημιουργούμε έναν πίνακα με δεδομένα και επικεφαλίδες. Με κατάλληλα `JFrame` και `JScrollPane` (για κύλιση, αν χρειαστεί) εμφανίζουμε τα περιεχόμενά του. Έχουμε ενσωματώσει και ένα κουμπί που, όταν πατηθεί, προστίθεται μια ακόμα γραμμή δεδομένων στον πίνακα. Το κουμπί τοποθετείται σε ένα `JPanel` και ενσωματώνεται στη νότια περιοχή του παραθύρου (`South`) μέσω του διαχειριστή διάταξης `BorderLayout`. Με τον `ActionListener` στο κουμπί διαχειριζόμαστε το συμβάν κλικ και εκτελούμε τη λογική για την προσθήκη γραμμών. Ο αριθμός ID υπολογίζεται δυναμικά με βάση τον αριθμό των υπαρχουσών γραμμών.

```
import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class Swing16 {
```

```
	public static void main(String[] args) {
		// Δημιουργία του TableModel με δεδομένα και επικεφαλίδες στηλών
		String[] columnNames = { "ID", "Γλώσσες Προγ/μου" };
		Object[][] data = { { "1", "Java" }, { "2", "Python" }, { "3", "C/C++" } };

		DefaultTableModel tableModel = new DefaultTableModel(data, columnNames);

		// Δημιουργία JTable με το TableModel
		JTable table = new JTable(tableModel);

		// Δημιουργία JScrollPane για κύλιση
		JScrollPane scrollPane = new JScrollPane(table);

		// Δημιουργία JButton για προσθήκη γραμμών
		JButton addButton = new JButton("Νέα Γραμμή");

		// Προσθήκη ActionListener στο κουμπί
		addButton.addActionListener(new ActionListener() {
			@Override
			public void actionPerformed(ActionEvent e) {
				// Προσθήκη νέας γραμμής
				int newId = tableModel.getRowCount() + 1; // Αυτόματη αρίθμηση ID
				tableModel.addRow(new Object[] { String.valueOf(newId), "Scala " });
			}
		});

		// Δημιουργία JPanel για διάταξη
		JPanel panel = new JPanel();
		panel.add(addButton);

		// Δημιουργία JFrame
		JFrame frame = new JFrame("JTable & TableModel");
		frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		frame.add(scrollPane, "Center"); // Πίνακας στο κέντρο
		frame.add(panel, "South"); // Κουμπί στο κάτω μέρος
		frame.setSize(400, 300); // Ορισμός μεγέθους παραθύρου
		frame.setVisible(true); // Εμφάνιση παραθύρου
	}
}
```

Παρατηρήστε τον τρόπου που χειριζόμαστε το `JTable`. Δημιουργήσαμε ένα αντικείμενο `TableModel` στο οποίο αποθηκεύσαμε τα δεδομένα και στη συνέχεια το αντικείμενο `JTable` όπου του περάσαμε το μοντέλο ως παράμετρο. Με το πάτημα προσθήκης «μιλήσαμε» στο `TableModel` προσθέτοντας τη νέα γραμμή, καθώς αυτό έχει την ευθύνη διαχείρισης των δεδομένων. Για τον ίδιο λόγο ανακτήσαμε το πλήθος των γραμμών και πάλι από το `TableModel`. Τα υπόλοιπα (π.χ. εμφάνιση, ή μαύρισμα κελιού όταν κάνουμε κλικ μέσα του) είναι δουλειά του UI delegate. Δείτε στην Εικόνα 60, το παράθυρο με την εκτέλεση του κώδικα (αριστερά) και το παράθυρο μετά το πάτημα του πλήκτρου (δεξιά).

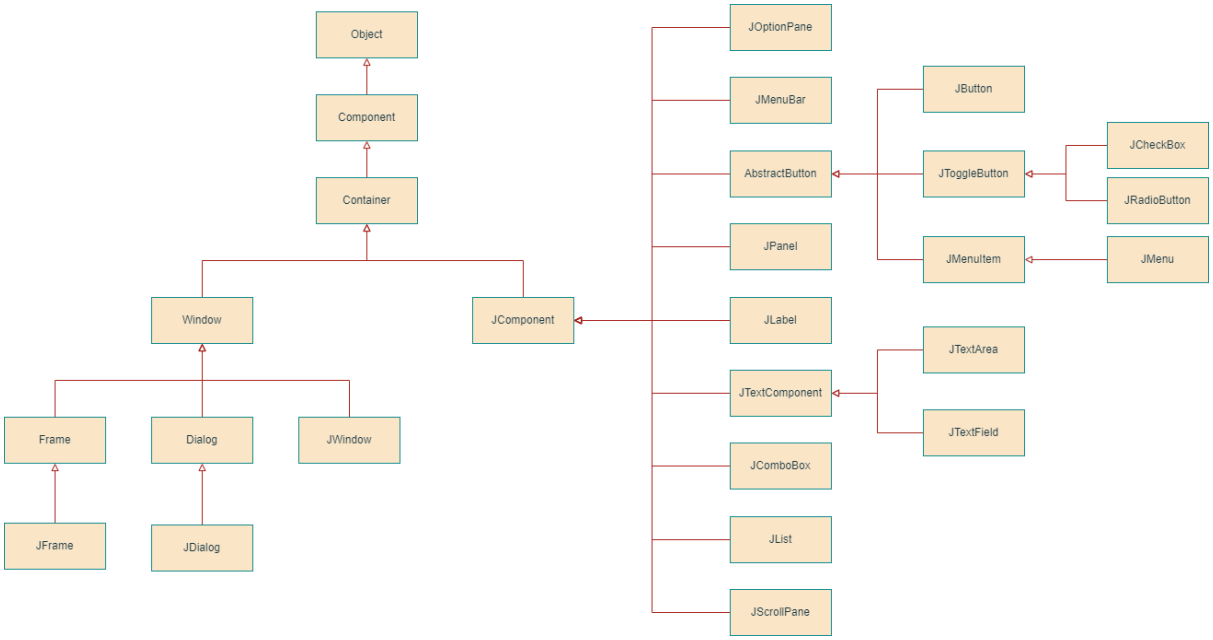


Εικόνα 60 - χρήση του JTable για διαχείριση και εμφάνιση δεδομένων

9.7.1. Μια μερική ιεραρχία των κλάσεων Swing

Καθώς πλησιάζουμε προς το τέλος της παρουσίασης του Java Swing και έχοντας πλέον δει και εξοικειωθεί με μια μεγάλη ομάδα από τα συστατικά του, παραθέτουμε εδώ μια (μερική και όχι πλήρη) ιεραρχία των σχετικών κλάσεων, που μπορείτε να δείτε στην Εικόνα 61.

Σημειώστε ότι είναι μέρος της συνολικής ιεραρχίας Swing και επίσης δεν αποτελεί πλήρες διάγραμμα κλάσεων, αλλά μόνο κληρονομικότητας.



Εικόνα 61 - Swing: μια (απλοποιημένη) ιεραρχία κλάσεων

9.7.2. Παράδειγμα χρησιμότητας του διαγράμματος ιεραρχίας κλάσεων

Γενικά, είναι πολύ χρήσιμο ο προγραμματιστής να γνωρίζει και να χρησιμοποιεί τις ιεραρχίες των κλάσεων, πράγμα που γίνεται φανερό και στην περίπτωση του Swing. Διαβάζοντας το διάγραμμα, ο προγραμματιστής μπορεί να κατανοήσει πώς τα διαφορετικά συστατικά σχετίζονται μέσω της κληρονομικότητας και να επιλέξει το σωστό επίπεδο αφαίρεσης για τις λειτουργίες που επιθυμεί.

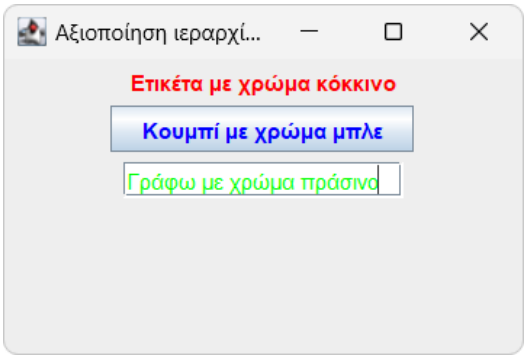
Ας δούμε ένα παράδειγμα, με τις κλάσεις `Jlabel`, `JButton` και `JTextField`.

- Κληρονομικότητα. Και τα τρία συστατικά κληρονομούν από την κλάση `JComponent`, επομένως μπορούν να χρησιμοποιούν κοινές μεθόδους όπως, για παράδειγμα τις `setForeground()` και `setFont()`.
- Πολυμορφισμός. Και τα τρία είναι («is-a») αντικείμενα `JComponent` και επομένως μπορούμε να τα χειριστούμε με ομοιόμορφο και άρα πιο εύκολο τρόπο.

Στον κώδικα που ακολουθεί, έχουμε δημιουργήσει μια μέθοδο `changeForegroundColor` η οποία δέχεται ως όρισμα ένα αντικείμενο `JComponent` και ένα αντικείμενο `Color` για το χρώμα. Η μέθοδος χρησιμοποιεί τη μέθοδο `setForeground()` της κλάσης `JComponent` για να καθορίσει το χρώμα κειμένου στο αντικείμενο που της περνάμε, ασχέτως του ειδικότερου τύπου του καθενός.

```
import javax.swing.*;
import java.awt.*;

public class InheritanceExample {
      public static void main(String[] args) {
            // Δημιουργία JFrame
            JFrame frame = new JFrame("Αξιοποίηση ιεραρχίας Swing");
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            frame.setSize(300, 200);
            frame.setLayout(new FlowLayout());

            // Δημιουργία συστατικών Swing
            JLabel label = new JLabel("Ετικέτα με χρώμα κόκκινο");
            JButton button = new JButton("Κουμπί με χρώμα μπλε");
            JTextField textField = new JTextField(15);

            // Αλλαγή του χρώματος κειμένου μέσω της κοινής γονικής κλάσης
JComponent
            changeForegroundColor(label, Color.RED);
            changeForegroundColor(button, Color.BLUE);
            changeForegroundColor(textField, Color.GREEN);

            // Προσθήκη components στο frame
            frame.add(label);
            frame.add(button);
            frame.add(textField);

            // Εμφάνιση του παραθύρου
            frame.setVisible(true);
      }

      // Μέθοδος που λειτουργεί για οποιοδήποτε αντικείμενο JComponent
      private static void changeForegroundColor(JComponent component, Color
color) {
            component.setForeground(color);
      }
}
```

Με την παραπάνω λογική, διαχειριζόμαστε τρία αντικείμενα `Jlabel`, `JButton` και `JtextField` στα οποία θέλουμε να καθορίσουμε το χρώμα κειμένου με ενιαίο και συνεκτικό τρόπο.



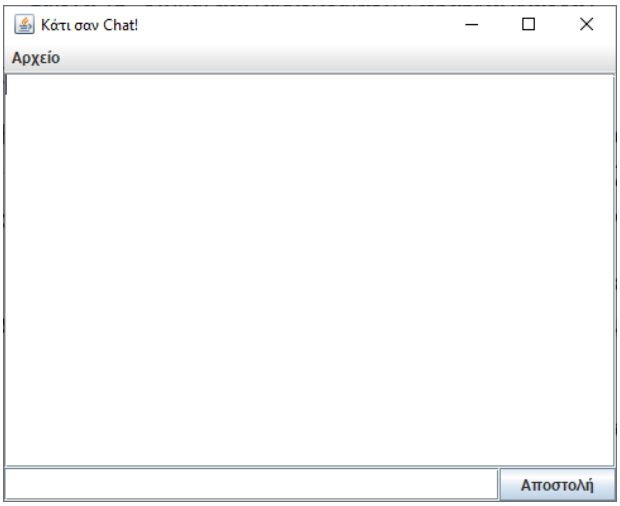
Εικόνα 62 - χρήση JComponent.setForeground() σε αντικείμενα-απογόνους της κλάσης

Δείτε το αποτέλεσμα της εκτέλεσης στην Εικόνα 62.

9.8. Πρακτική Εξάσκηση - δημιουργία γραφικού περιβάλλοντος

Σε μια εφαρμογή Text Chat, ο χρήστης διαθέτει ένα πλαίσιο κειμένου για να γράψει το μήνυμά του. Στη συνέχεια, πατώντας σε ένα πλήκτρο «Αποστολή», το μήνυμα φεύγει από το πλαίσιο που το έγραψε και εμφανίζεται σε έναν χώρο παραπάνω, όπου υπάρχουν τα μηνύματα που ανταλλάσσονται, ενώ ταυτόχρονα το μήνυμα αποστέλλεται. Μια πραγματική εφαρμογή χρησιμοποιείται από περισσότερους του ενός χρήστες, και ο χώρος με τα ανταλασσόμενα μηνύματα είναι ορατός σε όλους τους χρήστες. Η εφαρμογή μας δεν θα στέλνει μήνυμα, απλά θα προσομοιώνει αυτή τη λειτουργία.

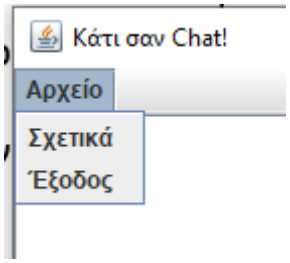
Όταν εκτελεστεί το πρόγραμμα, θα φαίνεται το παράθυρο που βλέπουμε στην Εικόνα 63.



Εικόνα 63 - πρακτική άσκηση - το αρχικό παράθυρο

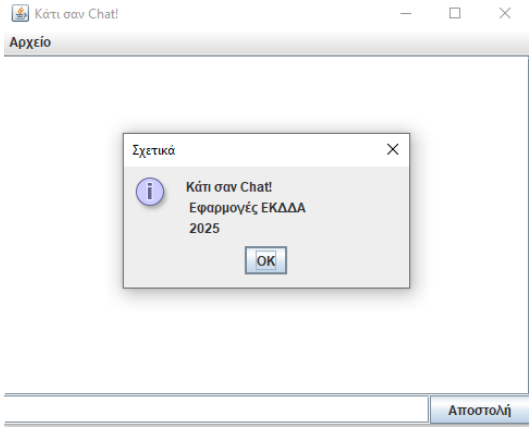
Ειδικότερα (Εικόνα 63), θα έχει τα εξής χαρακτηριστικά:

- Όνομα εφαρμογής: «Κάτι σαν Chat!».
- Ένα μενού με τίτλο «Αρχείο» που θα περιλαμβάνει δύο επιλογές (Εικόνα 64): «Σχετικά» και «Έξοδος». Στην δεύτερη περίπτωση, το παράθυρο κλείνει και το πρόγραμμα τερματίζει.



Εικόνα 64 - πρακτική άσκηση - μενού και επιλογές

Στην περίπτωση που ο χρήστης επιλέξει «Σχετικά», θα βλέπει ένα μήνυμα, όπως αυτό στην Εικόνα 65.



Εικόνα 65 - πρακτική άσκηση - μήνυμα από επιλογή στο μενού

- Ένα πλαίσιο κειμένου και δίπλα ένα πλήκτρο με τίτλο «Αποστολή». Ο χρήστης θα γράφει το μήνυμα και θα πατά το πλήκτρο.
- Μια περιοχή κειμένου (πολλών γραμμών) πάνω από το ζευγάρι πλαισίου κειμένου-κουμπιού. Εκεί, σε νέα γραμμή κάθε φορά, θα εμφανίζεται το μήνυμα όταν ο χρήστης πατάει το «Αποστολή» (και ταυτόχρονα το πλαίσιο κειμένου θα καθαρίζει).

Οδηγίες – συστάσεις

Χωρίστε τον κώδικα σε τρεις κλάσεις. Μια για το κύριο πρόγραμμα, μια για τον χειρισμό συμβάντων μενού και μια για τον χειρισμό συμβάντων κουμπιού.

Ο διαχειριστής συμβάντων του κουμπιού θα πρέπει να δέχεται (ο κατασκευαστής της κλάσης) το πεδίο κειμένου και την περιοχή κειμένου για να κάνει τις ενέργειες (καθαρισμός πεδίου, πέρασμα στην περιοχή).

Ο διαχειριστής συμβάντων του μενού θα πρέπει να δέχεται (ο κατασκευαστής της κλάσης) το γονικό frame, για να κεντράρει μέσα του το μήνυμα διαλόγου (Εικόνα 65).

Η περιοχή κειμένου δεν πρέπει να μπορεί να υποστεί επεξεργασία και αλλαγές, ενώ πρέπει να έχει δυνατότητα εμφάνισης ράβδων κύλισης αν χρειαστεί.

Χρησιμοποιήστε Διαχειριστή Διάταξης τον `BorderLayout` για το Frame και βάλτε την περιοχή κειμένου `CENTER`.

Δημιουργήστε ένα πάνελ με BorderLayout και βάλτε το πλαίσιο κειμένου `CENTER` και το κουμπί `EAST`.

Βάλτε το πάνελ στο Frame σε θέση `SOUTH`.

Στην περίπτωση του κουμπιού επειδή είναι ένα, ξέρουμε ότι μόνο αυτό μπορεί να δημιουργήσει σχετικό συμβάν. Στην περίπτωση του μενού, το συμβάν μπορεί να έρθει από δύο επιλογές του χρήστη, «Σχετικά» ή «Έξοδος». Για να τα ξεχωρίζει ο διαχειριστής συμβάντος, χρησιμοποιήστε την μέθοδο `setActionCommand()`, για παράδειγμα στο κύριο πρόγραμμα:

```
JMenuItem exitItem = new JMenuItem("Έξοδος");

exitItem.setActionCommand("EXIT");

exitItem.addActionListener(new MenuActionListener(frame));
```

και στην μέθοδο χειρισμού του συμβάντος:

```
String command = e.getActionCommand(); // Από ποια επιλογή στο μενού ήρθαμε

switch (command) {

case "EXIT": κ.λπ.
```

Απάντηση. Παραθέτουμε παρακάτω μια ενδεικτική επίλυση της άσκησης, με την παρατήρηση ότι ο κώδικας δίνεται ως δείγμα για εκπαιδευτικούς λόγους.

```
// Κλάση 1
import javax.swing.*;
import java.awt.*;

public class PraktikiAskisi {
      public static void main(String[] args) {
            // Δημιουργία παραθύρου
            JFrame frame = new JFrame("Κάτι σαν Chat!");
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            frame.setSize(500, 400);
            frame.setLayout(new BorderLayout());

            // Δημιουργία του μενού
            JMenuBar menuBar = new JMenuBar();
            JMenu fileMenu = new JMenu("Αρχείο");

            // Δημιουργία στοιχείων μενού
            JMenuItem exitItem = new JMenuItem("Έξοδος");
            exitItem.setActionCommand("EXIT");
            exitItem.addActionListener(new MenuActionListener(frame));

            JMenuItem aboutItem = new JMenuItem("Σχετικά");
            aboutItem.setActionCommand("ABOUT");
            aboutItem.addActionListener(new MenuActionListener(frame));

            // Προσθήκη των στοιχείων στο μενού
            fileMenu.add(aboutItem);
            fileMenu.add(exitItem);
            menuBar.add(fileMenu);
```

```
            frame.setJMenuBar(menuBar);

            // Δημιουργία TextArea και ScrollPane
            JTextArea textArea = new JTextArea();
            textArea.setEditable(false);
            JScrollPane scrollPane = new JScrollPane(textArea);
            frame.add(scrollPane, BorderLayout.CENTER);

            // Δημιουργία TextField και Button
            JPanel inputPanel = new JPanel();
            inputPanel.setLayout(new BorderLayout());
            JTextField textField = new JTextField();
            JButton addButton = new JButton("Αποστολή");

            // Προσθήκη event handler στο κουμπί
            addButton.addActionListener(new ButtonActionListener(textField,
textArea));

            // Προσθήκη στο inputPanel
            inputPanel.add(textField, BorderLayout.CENTER);
            inputPanel.add(addButton, BorderLayout.EAST);

            frame.add(inputPanel, BorderLayout.SOUTH);
            // Κεντράρισμα frame στην οθόνη
        frame.setLocationRelativeTo(null);
            // Εμφάνιση παραθύρου
            frame.setVisible(true);
      }
}
// Κλάση 2
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.*;

class ButtonActionListener implements ActionListener {
      private JTextField textField;
      private JTextArea textArea;

      public ButtonActionListener(JTextField textField, JTextArea textArea)
{
            this.textField = textField;
            this.textArea = textArea;
      }

      @Override
      public void actionPerformed(ActionEvent e) {
            String text = textField.getText();
            if (!text.isEmpty()) {
                  textArea.append(text + "\n");
                  textField.setText("");
            }
      }
}
// Κλάση 3
import javax.swing.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

class MenuActionListener implements ActionListener {
```

```
	private JFrame parentFrame;

	// Περνάμε το γονικό frame για να κεντράρουμε μέσα του το πλαίσιο διαλόγου "Σχετικά"
	MenuActionListener(JFrame pf) {
		this.parentFrame = pf;
	}

	@Override
	public void actionPerformed(ActionEvent e) {
		String command = e.getActionCommand(); // Από ποια επιλογή στο μενού ήρθαμε
		switch (command) {
		case "EXIT":
			System.exit(0);
			break;
		case "ABOUT":
			JOptionPane.showMessageDialog(parentFrame, "Κάτι σαν Chat!\n Εφαρμογές ΕΚΔΔΑ\n 2025", "Σχετικά",
					JOptionPane.INFORMATION_MESSAGE);
			break;
		default:
			System.err.println("Άγνωστη εντολή: " + command);
		}
	}
}
```

9.9. Ερωτήσεις αυτο-αξιολόγησης

1. Ποια από τις παρακάτω είναι μέρος των Java Foundation Classes (JFC);

α) AWT

β) Swing

γ) Java 2D

δ) Όλα τα παραπάνω

2. Ποιο από τα παρακάτω δεν ανήκει στα βασικά συστατικά του AWT;

α) Button

β) Canvas

γ) GridBagLayout

δ) FXML

3. Ποιος Layout Manager τοποθετεί τα στοιχεία σε γραμμές και στήλες;

α) GridLayout

β) FlowLayout

γ) BorderLayout

δ) CardLayout

4. Ποιο από τα παρακάτω χαρακτηριστικά περιγράφει καλύτερα το FlowLayout;

α) Τοποθετεί τα στοιχεία σε γραμμές, μία μετά την άλλη

β) Τοποθετεί τα στοιχεία σε απόλυτες συντεταγμένες

γ) Τοποθετεί τα στοιχεία σε σχήμα κάρτας

δ) Τοποθετεί τα στοιχεία σε πλέγμα με προκαθορισμένες διαστάσεις

5. Ποια μέθοδος χρησιμοποιείται για την προσθήκη ενός ActionListener σε ένα JButton;

α) addListener()

β) addActionListener()

γ) setListener()

δ) setActionListener()

6. Ποια μέθοδος καλείται για να πάρουμε το κείμενο από ένα JTextField;

α) getText()

β) getValue()

γ) retrieveText()

δ) getInput()

7. Ποια είναι η βασική αρχή του Event-Driven προγραμματισμού;

α) Ο συγχρονισμός νημάτων

β) Η ενεργοποίηση με βάση τα συμβάντα

γ) Η στατική εκτέλεση

δ) Η απουσία διαχείρισης συμβάντων

8. Ποια μέθοδος μπορεί να χρησιμοποιηθεί για να συνδεθεί ένα MouseListener σε ένα JButton;

α) addMouseListener()

β) addClickListener()

γ) addHoverListener()

δ) setMouseListener()

9. Ποια κλάση χρησιμοποιείται για τη δημιουργία μενού στο Swing;

α) JMenuBar

β) JMenu

γ) JMenuItem

δ) Όλα τα παραπάνω

10. Ποια μέθοδος του JOptionPane χρησιμοποιείται για την εμφάνιση παραθύρου επιβεβαίωσης;

α) showConfirmDialog()

β) showInputDialog()

γ) showMessageDialog()

δ) showOptionDialog()

11. Στο πρότυπο MVC, τι αντιπροσωπεύει το "View";

α) Την παρουσίαση δεδομένων

β) Τη λογική της εφαρμογής

γ) Τη διαχείριση δεδομένων

δ) Τη σύνδεση με τον χρήστη

12. Ποια μέθοδος στο Swing χρησιμοποιείται για την ενημέρωση του View όταν αλλάζουν τα δεδομένα;

α) revalidate()

β) repaint()

γ) updateUI()

δ) Όλα τα παραπάνω

13. Ποια από τις παρακάτω κλάσεις είναι τμήμα του πακέτου java.awt αλλά όχι του Swing;

α) JPanel

β) Frame

γ) JButton

δ) JLabel

14. Ποιο πακέτο περιέχει την υλοποίηση του Swing;

α) java.awt.swing

β) javax.swing

γ) org.swing

δ) java.swing

15. Ποιος Layout Manager προσφέρει τη δυνατότητα τοποθέτησης στοιχείων με διαφορετικά μεγέθη σε προκαθορισμένα "βάρη";

α) GridBagLayout

β) FlowLayout

γ) BorderLayout

δ) CardLayout

16. Ποιος Layout Manager είναι ο προεπιλεγμένος για την κλάση JPanel;

α) BorderLayout

β) FlowLayout

γ) GridLayout

δ) CardLayout

17. Ποια κλάση χρησιμοποιείται για την υλοποίηση scrollbars σε μια Swing εφαρμογή;

α) JScrollPane

β) JScrollBar

γ) JPanel

δ) JScrollBox

18. Ποια μέθοδος καλείται για την προσθήκη στοιχείων σε μια λίστα JList;

α) addElement()

β) addItem()

γ) setListData()

δ) insert()

19. Ποιο event είναι κατάλληλο για να ανιχνεύσουμε κλικ σε ένα κουμπί;

α) MouseEvent

β) KeyEvent

γ) ActionEvent

δ) FocusEvent

20. Ποια μέθοδος καλείται για να κλείσει ένα JDialog;

α) dispose()

β) hide()

γ) close()

δ) end()

9.10. Απαντήσεις στις ερωτήσεις Αυτο-αξιολόγησης

1. δ	2. δ	3. α	4. α	5. β
6. α	7. β	8. α	9. δ	10. α
11. α	12. δ	13. β	14. β	15. α
16. β	17. α	18. γ	19. γ	20. α

ΚΕΦΑΛΑΙΟ 10: Ολοκληρωμένο παράδειγμα - JAVA PROJECT

10.1. Εκπαιδευτικοί στόχοι του παρόντος κεφαλαίου

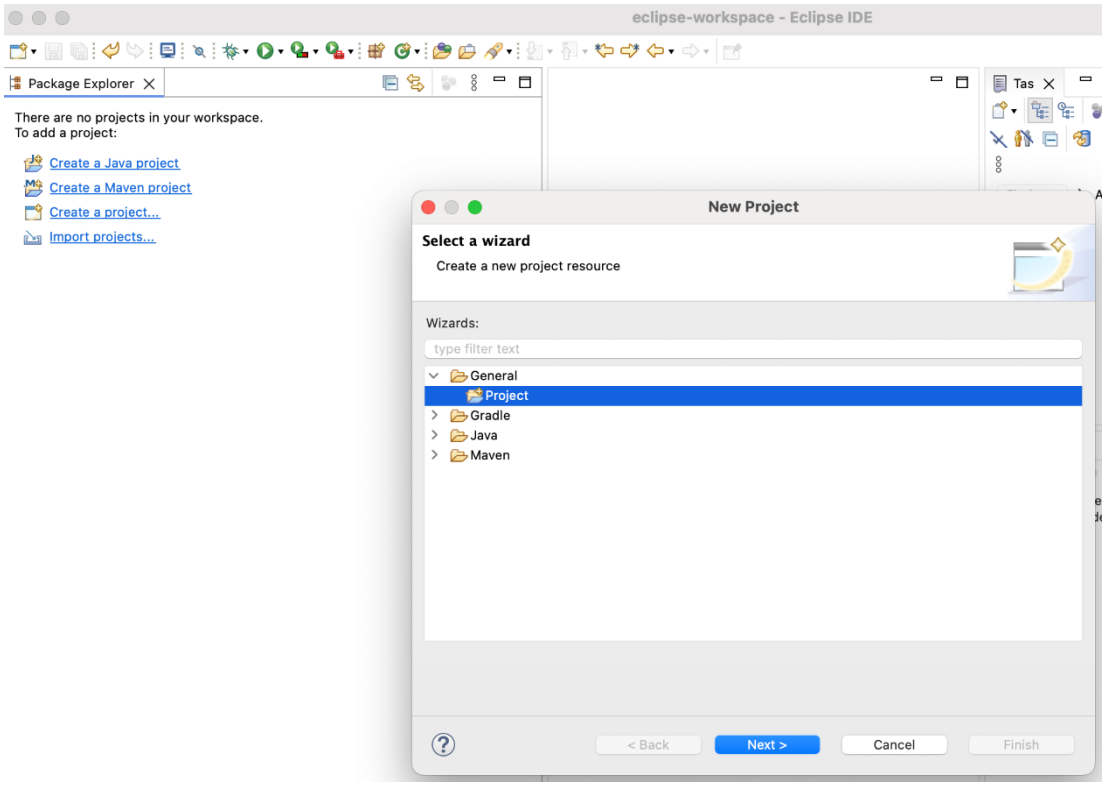
Οι εκπαιδευτικοί στόχοι του παρόντος κεφαλαίου, συνοψίζονται στα κάτωθι σημεία. Οι εκπαιδευόμενοι:

- θα κάνουν deploy ένα ολοκληρωμένο παράδειγμα σε μορφή jar στο Eclipse,
- θα δημιουργήσουν jar με χρήση του Eclipse,
- θα κάνουν έλεγχο και debugging.

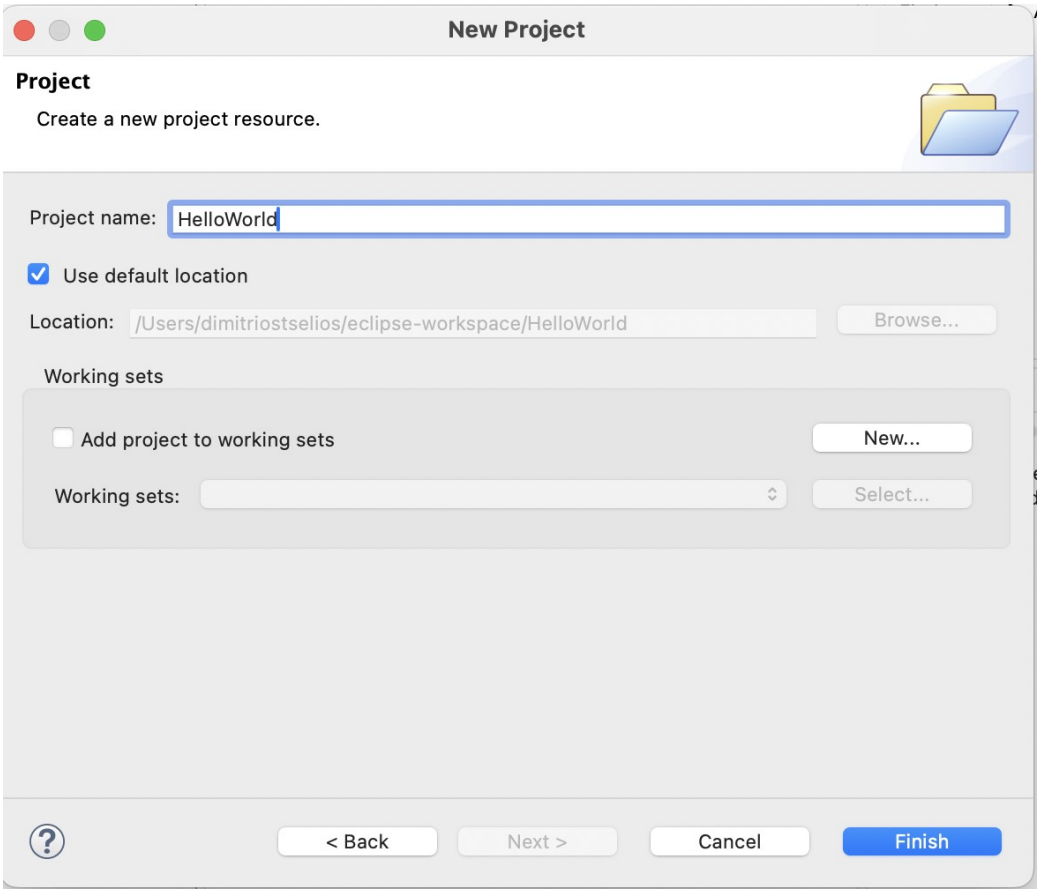
10.2. Deploy ένα ολοκληρωμένο παράδειγμα σε μορφή jar στο Eclipse

Ένας χρήσιμος τρόπος για να διαχειριστούμε τα αρχεία ενός έργου (project) στην Java είναι η δημιουργία ενός αρχείου με επέκταση **.jar**. Σύμφωνα με την Oracle, ένα αρχείο jar είναι μία άλλη μορφή της δημοφιλούς μορφής συμπιεσμένων αρχείων zip. Στην πραγματικότητα αρχείο jar είναι αρχείο zip το οποίο έχει μία προαιρετική πληροφορία για τον κατάλογο META-INF. Το αρχείο jar μπορεί να δημιουργηθεί από τη γραμμή εντολών του εργαλείου jar ή με χρήση του **API java.util.jar**. Επιπλέον, τα **IDE** εργαλεία διαθέτουν κατάλληλες επιλογές μενού για την επίτευξη του ίδιου στόχου. Θα παρουσιαστεί η τελευταία επιλογή για την εισαγωγή ενός αρχείου jar που είναι αποθηκευμένο σε κάποιο μέσο αποθήκευσης. Το παράδειγμα που ακολουθεί δείχνει σε στιγμιότυπα της οθόνης μας πως μπορεί να εισαχθεί ένα τέτοιο αρχείο.

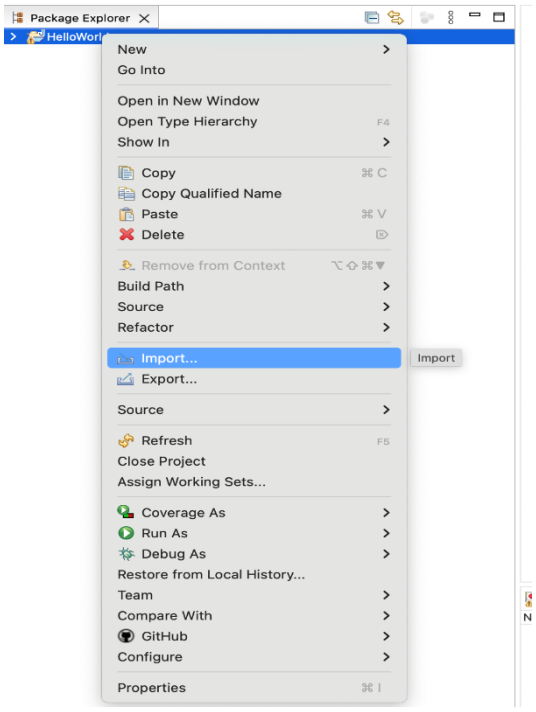
Πρώτα δημιουργούμε ένα νέο Project όπως ήδη γνωρίζουμε μέσα στο Eclipse.



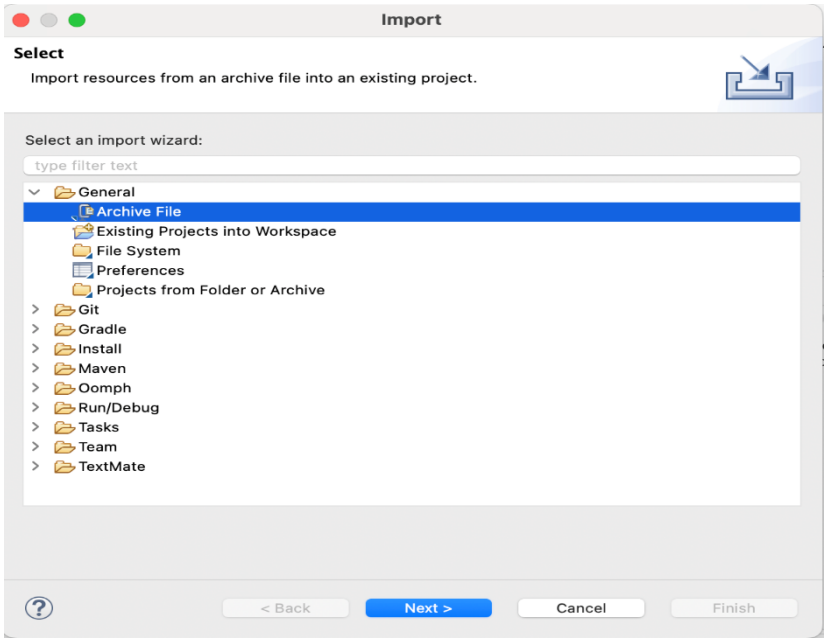
Δίνουμε το όνομα του project.



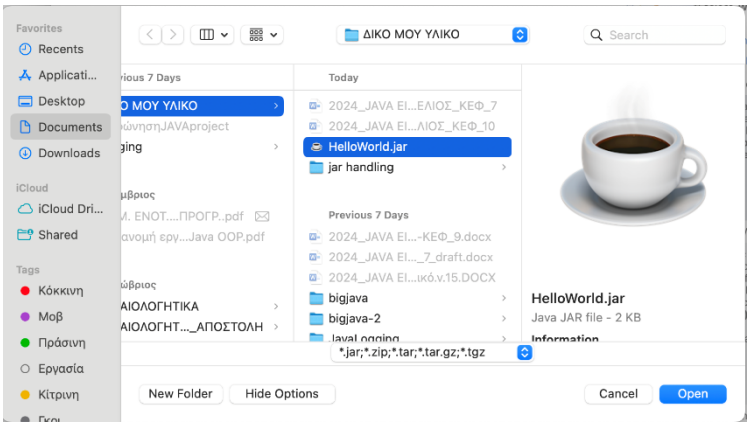
Έπειτα επιλέγουμε μετά από ένα δεξί κλικ στην περιοχή των projects την επιλογή **Import**.

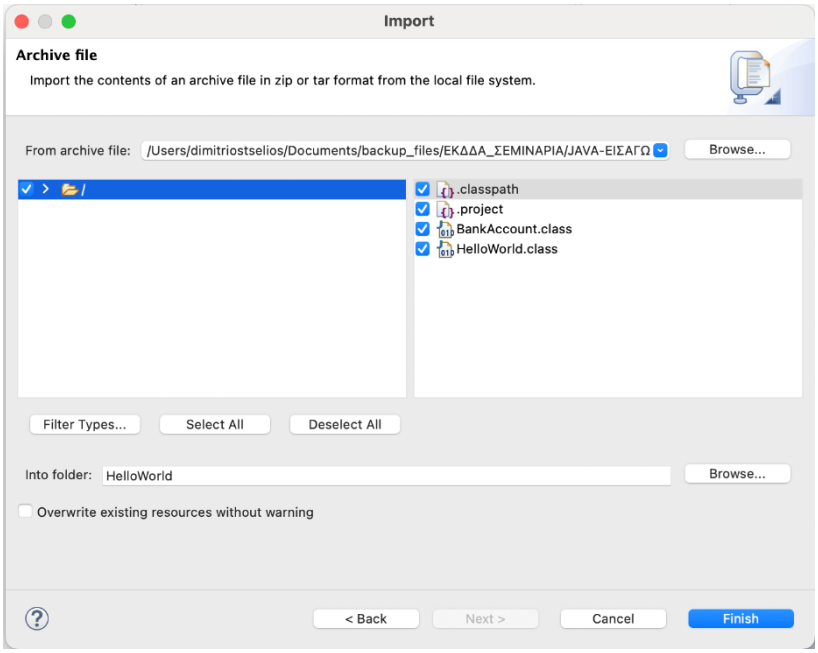


Στη συνέχεια επιλέγουμε ως μορφή του αρχείου που θα εισαχθεί το **Archive File** όπως φαίνεται στην ακόλουθη εικόνα.



Αναζητούμε στο σύστημα αρχείων του υπολογιστή μας το αρχείο jar που μας ενδιαφέρει καθώς και τα επιμέρους περιεχόμενα αρχεία και έπειτα επιλέγουμε το **Finish**.





Το αποτέλεσμα θα είναι η εισαγωγή των αρχείων που εμπεριέχονται στο αρχείο **jar** στο project που είχε δημιουργηθεί. Με αυτόν τον τρόπο προσπαθήστε να ανοίξετε το αρχείο με όνομα **MyBnbProject_v1.jar** που βρίσκεται στα συνοδευτικά αρχεία του εκπαιδευτικού οδηγού.

10.3. Πρακτική Εξάσκηση Επέκταση του Παραδείγματος

Πρώτη έκδοση του project MyBnb. Το αρχείο που ανοίχθηκε στην προηγούμενη ενότητα περιλαμβάνει τα αρχεία μίας μικρής εφαρμογής (χωρίς την υλοποίηση του κώδικα τους) για μία υπηρεσία ενοικίασης ακινήτων. Η MyBnb, μια startup κοινής χρήσης ακινήτων, σχεδιάζει να ξεκινήσει νέες υπηρεσίες ενοικίασης τμηματοποιώντας τα ακίνητα σύμφωνα με τα πρότυπα τους. Πιο συγκεκριμένα, τα ακίνητα χωρίζονται σε τρεις κατηγορίες, δηλαδή διαμερίσματα, σπίτια και πολυτελείς βίλες. Όλα τα ακίνητα έχουν κοινά στοιχεία, δηλαδή **μοναδικό** αριθμό μητρώου, όνομα ιδιοκτήτη, ταχυδρομική διεύθυνση, κόστος ενοικίασης ανά ημέρα, συνολικό αριθμό ημερών ενοικίασης ανά σεζόν. Το διαμέρισμα έχει ως επιπλέον πληροφορίες τον αριθμό του ορόφου και τον αριθμό των κλινών. Το σπίτι έχει ως επιπλέον πληροφορίες τον συνολικό αριθμό των ορόφων και τα τέλη καθαριότητας. Η πολυτελής βίλα έχει ως επιπλέον πληροφορίες το συνολικό αριθμό των δωματίων, το κόστος της υπηρεσίας δωματίου ανά ημέρα και το φόρο πολυτελείας ανά ημέρα.

Η **MyBnb** θέλει να αναπτύξει ένα πρόγραμμα Java που θα χειρίζεται ολόκληρη την επιχείρηση. Επιπλέον, ο αναλυτής της απαιτεί από τους προγραμματιστές να αναπτύξουν τρεις κλάσεις. π.χ. **Property**, **Apartment**, **Villa** εντός των απαραίτητων χαρακτηριστικών και των κατάλληλων constructors, mutators και accessors. Ο αναλυτής χρειάζεται την ύπαρξη array lists για να καλύψει τους τρεις διαφορετικούς τύπους ιδιοκτησιών.

Απώτερος στόχος είναι ο υπολογισμός του συνολικού εισοδήματος από όλα τα ακίνητα, συμπεριλαμβανομένων των πρόσθετων δαπανών και φόρων. Πιο αναλυτικά, θα πρέπει να παραδώσετε τις ακόλουθες κλάσεις και μεθόδους.

Μέρος Α

Property class: Αυτή η κλάση είναι η υπερ-κλάση όλων των τύπων ακινήτων. Η κλάση θα πρέπει να περιέχει τα απαραίτητα χαρακτηριστικά (μοναδικό αριθμό μητρώου, όνομα ιδιοκτήτη, ταχυδρομική διεύθυνση, κόστος ενοικίασης ανά ημέρα, συνολικό αριθμό ημερών ενοικίασης σεζόν) και ολόκληρο το σύνολο των μεθόδων constructor, mutator και accessor. Αυτό το τμήμα αξίζει το **15%** της τελικής εργασίας.

Apartment class: Αυτή η κατηγορία είναι υποκατηγορία του ακινήτου. Η κλάση πρέπει να περιέχει τα απαραίτητα πρόσθετα χαρακτηριστικά (αριθμός ορόφων και αριθμός κλινών) και ολόκληρο το σύνολο των μεθόδων constructor, mutator και accessor. Αυτό το τμήμα αξίζει το **15%** της τελικής εργασίας.

House class: Αυτή η κλάση είναι υποκατηγορία του ακινήτου. Η κλάση πρέπει να περιέχει τα απαραίτητα πρόσθετα χαρακτηριστικά (συνολικός αριθμός ορόφων και τέλη εκκαθάρισης) και ολόκληρο το σύνολο των μεθόδων constructor, mutator και accessor. Αυτό το τμήμα αξίζει το **15%** της τελικής εργασίας.

Villa class: Αυτή η κατηγορία είναι υποκατηγορία του ακινήτου. Η κλάση θα πρέπει να περιέχει τα απαραίτητα πρόσθετα χαρακτηριστικά (το κόστος υπηρεσίας δωματίου ανά ημέρα και τον φόρο πολυτελείας ανά ημέρα) και ολόκληρο το σύνολο των μεθόδων constructor, mutator και accessor. Αυτό το τμήμα αξίζει το **15%** της τελικής εργασίας.

Μέρος Β

Μέθοδος FillInProperties: Αυτή η μέθοδος θα πρέπει να εισάγει δεδομένα σε τρεις **ArrayLists**, δηλαδή διαμερίσματα, σπίτια και βίλες. Θα πρέπει να εισαγάγετε τρία αντικείμενα σε κάθε λίστα (3 διαμερίσματα, 3 σπίτια, 3 βίλες). Στη συνέχεια, η μέθοδος θα καλέσει τη μέθοδο **RentProperty(int RentalDays)** 3 φορές για κάθε ακίνητο (συνολικά 27 φορές). Η μέθοδος **RentProperty** θα προσθέσει τις ημέρες ενοικίασης στις συνολικές μέρες ενοικίασης του καταλύματος. Μπορείτε να εισαγάγετε τα δεδομένα με μη αυτόματο τρόπο. Αυτή η ενότητα αξίζει **10%** τη εργασίας.

Μέθοδος PrintAllProperties: Αυτή η μέθοδος θα πρέπει να εμφανίζει το περιεχόμενο όλων των λιστών πινάκων μετά τη συμπλήρωσή τους με δεδομένα. Αυτό το τμήμα αξίζει το **10%** της τελικής εργασίας.

Μέθοδος CalculateTotalIncome: Αυτή η μέθοδος θα υπολογίσει το συνολικό εισόδημα από όλα τα ακίνητα, συμπεριλαμβανομένων των πρόσθετων δαπανών και φόρων. Αυτό το τμήμα αξίζει το **10%** της τελικής εργασίας.

Μέθοδος Main: Η κύρια κλάση που καλεί τις παραπάνω μεθόδους. Αυτή η ενότητα αξίζει **10%** του τελικού έργου.

Όλες οι μέθοδοι δίνονται στο αρχείο **MyBnbProject_v1.jar** χωρίς να είναι υλοποιημένες. Συνεπώς θα πρέπει να εισαγάγετε το αρχείο στο IDE Eclipse και αφού ολοκληρώσετε την υλοποίηση τους να κάνετε εξαγωγή σε ένα αρχείο **MyBnbProject_v2.jar** που θα είναι το παραδοτέο της εργασίας. Θα πρέπει να σχεδιάσετε τις δημόσιες κλάσεις όπως έχουν περιγραφεί παραπάνω, χρησιμοποιώντας τις έννοιες του πολυμορφισμού και της κληρονομικότητας. Χρειάζεστε επίσης την υλοποίηση των

κατάλληλων **ArrayLists**. Για λόγους απλότητας, υποθέστε όλες τις αριθμητικές μεταβλητές ως ακέραιους αριθμούς.

Η ενδεικτική λύση βρίσκεται στο αρχείο με όνομα **MyBnbProject_v2.jar** που βρίσκεται στα συνοδευτικά αρχεία του εκπαιδευτικού οδηγού.

Δεύτερη έκδοση του project MyBnb. Ως συνέχεια του προηγούμενου project η νεοφυής εταιρεία **MyBnb**, προκειμένου να βελτιώσει το λογισμικό της και τη διαδικασία εισόδου/εξόδου, απαιτεί από τους προγραμματιστές να υλοποιήσουν την όλη διαδικασία χρησιμοποιώντας αρχεία κειμένου εισόδου και εξόδου. Πιο συγκεκριμένα, το αρχείο **InputProperty.txt** περιέχει τις αρχικές πληροφορίες για κάθε ακίνητο σε μία γραμμή. Το πρώτο ψηφίο της γραμμής θα αντιπροσωπεύει τον τύπο της ιδιότητας και η υπόλοιπη συμβολοσειρά τις τιμές των πεδίων ιδιοτήτων. Οι ακόλουθες γραμμές δίνουν ένα παράδειγμα του **InputProperty.txt**:

```
1 1 2 50 0 George Paradise_3

1 2 3 50 0 John Somewhere_4

1 2 4 50 0 Jane Devil's_place_3

2 1 50 60 0 George Paradise_13

2 2 50 60 0 John Somewhere_14

2 4 60 60 0 Jane Devil's_place_13

3 5 10 10 50 0 George Paradise_23

3 4 15 20 40 0 John Somewhere_24

3 8 20 30 55 0 Jane Devil's_place_23
```

Θα πρέπει να γράψετε κώδικα για τη μέθοδο **ReadInputPropertyFile** που θα σαρώσει το αρχείο εισόδου και θα συμπληρώσει τα κατάλληλα **ArrayLists**. Επιπλέον, θα πρέπει να τροποποιήσετε τη μέθοδο **PrintAllProperties** του αρχικού έργου, προκειμένου να ανακατευθύνει την έξοδο σε ένα αρχείο κειμένου εξόδου με όνομα **OutputProperty.txt**. Η **MyBnb** απαιτεί επίσης από τους προγραμματιστές να χρησιμοποιούν χειρισμό εξαιρέσεων για τη διαδικασία εισαγωγής των δεδομένων.

Επιπλέον, θα πρέπει να τροποποιήσετε τον κώδικα σας για το πρώτο project, προκειμένου να αλλάξετε τη διαδικασία ενοικίασης. Η τροποποιημένη μέθοδος θα πρέπει να ζητήσει από τον χρήστη τον κωδικό ακινήτου και τις ημέρες ενοικίασης και στη συνέχεια ο κωδικός αναζητά την κατάλληλη **ArrayList**, προκειμένου να εντοπίσει το ακριβές ακίνητο και να ενημερώσει το συνολικό αριθμό ημερών ενοικίασης.

Η **MyBnb** σχεδιάζει επίσης να επεκτείνει τις δραστηριότητές της στην ενοικίαση οχημάτων. Η επιχείρηση πρέπει να προσφέρει υπηρεσίες ενοικίασης για τυπικά αυτοκίνητα και φορτηγά. Ο επικεφαλής αναλυτής προτείνει τη δήλωση ενός **interface** που ονομάζεται **RentItem** και την υλοποίηση του ακινήτου και του οχήματος με βάση αυτό. Πιο συγκεκριμένα, προτείνει το ακόλουθο σχήμα.

Τα ακίνητα και τα οχήματα υλοποιούν το **RentItem**. Το διαμέρισμα, το σπίτι και η βίλα είναι επεκτάσεις του ακινήτου (όπως περιγράφεται στο πρώτο project). Το αυτοκίνητο και το φορτηγό

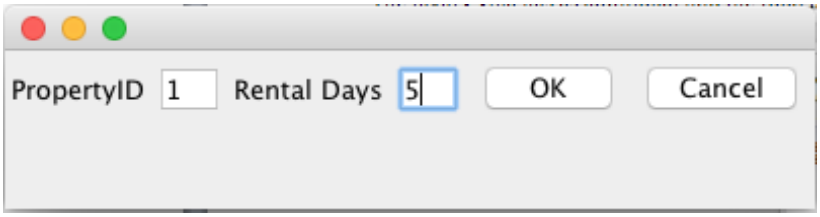
είναι επεκτάσεις του οχήματος. Η κατηγορία οχήματος έχει ως πρόσθετα χαρακτηριστικά τα **Owner**, **TotalRentalDays** και **RentalCostPerDay**. Το αυτοκίνητο έχει ως πρόσθετο χαρακτηριστικό το **PassengersNumber** και το **Truck** το **CargoWeight**. Πιο αναλυτικά, θα πρέπει να παραδώσετε τις ακόλουθες κλάσεις και μεθόδους.

Μέρος Α

Μέθοδος ReadInputPropertyFile: Αυτή η μέθοδος θα σαρώσει το αρχείο εισόδου και θα συμπληρώσει τις κατάλληλες λίστες πινάκων. Πιο συγκεκριμένα, το αρχείο **InputProperty.txt** περιέχει τις αρχικές πληροφορίες για κάθε ακίνητο σε μία γραμμή. Το πρώτο ψηφίο της γραμμής θα αντιπροσωπεύει τον τύπο της ιδιότητας και η υπόλοιπη συμβολοσειρά τις τιμές των πεδίων ιδιοτήτων. Αυτό το τμήμα αξίζει το **15%** της τελικής εργασίας.

Μέθοδος PrintAllProperties: Θα πρέπει να τροποποιήσετε αυτήν τη μέθοδο (μέθοδος πρώτου project), προκειμένου να ανακατευθύνετε την έξοδο σε ένα αρχείο κειμένου εξόδου με όνομα **OutputProperty.txt**. Αυτό το τμήμα αξίζει το **10%** της τελικής εργασίας.

Μέθοδος GiveRentalProperty: Η μέθοδος **GiveRentalProperty** εμφανίζει ένα πλαίσιο (χρησιμοποιώντας το **swing** packet), όπως φαίνεται στην ακόλουθη εικόνα, και ο χρήστης μπορεί να εισαγάγει μια νέα συναλλαγή ενοικίασης δίνοντας τον κωδικό ακινήτου και τις ημέρες ενοικίασης. Η μέθοδος αναζητά το αναγνωριστικό ιδιότητας στις τρεις λίστες πίνακα. Αυτό το τμήμα αξίζει το **35%** της τελικής εργασίας.



Μέρος Β

Interface RentalItem: Αυτή η διεπαφή αντιπροσωπεύει όλα όσα μπορούν να ενοικιαστούν. Έχει μόνο μία δηλωμένη μέθοδο, **RentalItem(int RentalDays)**. Αυτό το τμήμα αξίζει το **5%** της τελικής εργασίας.

Κλάση Property: Θα πρέπει να τροποποιήσετε αυτήν την κλάση, προκειμένου να είναι η υλοποίηση του interface. Αυτό το τμήμα αξίζει **το 5%** της τελικής εργασίας.

Κλάσεις Vehicle, Car και Truck: Αυτή η κατηγορία θα είναι μια νέα εφαρμογή του **interface RentItem**. Η μόνη διαφορά από το Property είναι η εφαρμογή της μεθόδου **RentalItem**() (πολυμορφισμός). Πιο συγκεκριμένα, αυξάνει το συνολικό αριθμό ημερών ενοικίασης μείον τη μία ελεύθερη ημέρα (ειδική προσφορά). Πρέπει να γράψετε μεθόδους **ReadInputVehicleFile**, **FillInVehicles** και **PrintAllVehicles** (ανάλογες με τις μεθόδους **ReadInputPropertyFile, FillInVehicles, PrintAllProperties**). Το αρχείο εισόδου πρέπει να ονομάζεται **InputVehicle.txt** και το αρχείο εξόδου **OutputVehicle.txt**. Αυτό το τμήμα αξίζει το **20%** της τελικής εργασίας.

Μέθοδος CalculateVehicleTotalIncome: Αυτή η μέθοδος θα υπολογίσει το συνολικό εισόδημα από όλα τα οχήματα, συμπεριλαμβανομένου του πρόσθετου κόστους και των φόρων. Αυτό σημαίνει ότι

για κάθε ενοικίαση φορτηγού υπάρχει ένα επιπλέον κόστος το οποίο είναι ίσο με **το CargoWeight**. Αυτό το τμήμα αξίζει το **5%** της τελικής εργασίας.

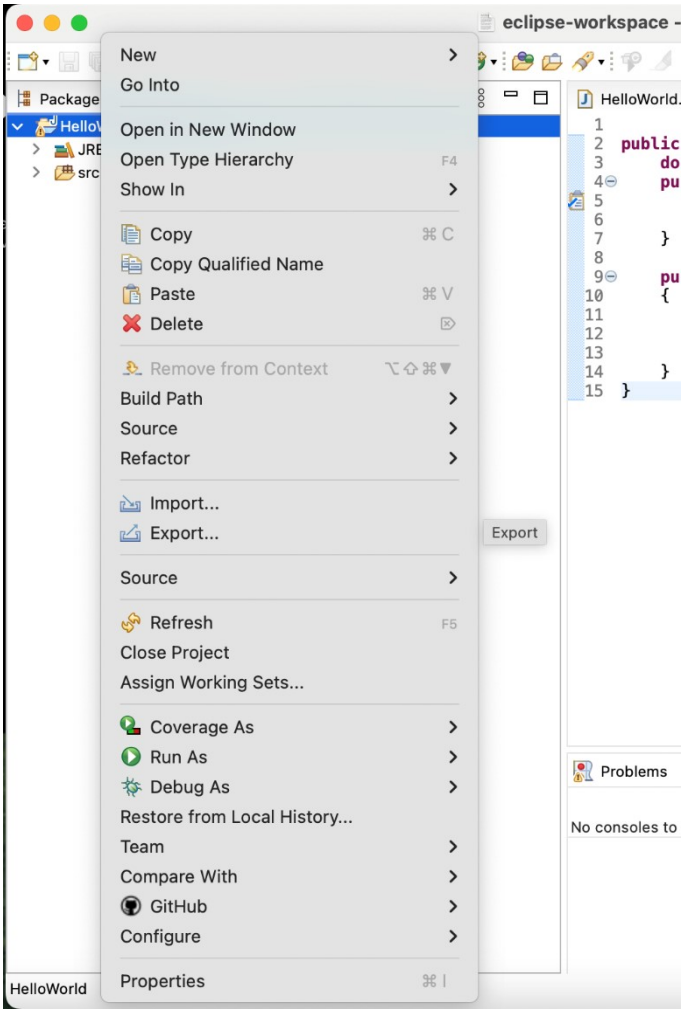
Μέθοδος Main: Η κύρια κλάση που καλεί όλες τις άλλες μεθόδους. Αυτή η ενότητα αξίζει το **5%** του τελικού έργου.

Θα πρέπει να σχεδιάσετε τις δημόσιες κλάσεις όπως έχουν περιγραφεί παραπάνω, χρησιμοποιώντας τις έννοιες του πολυμορφισμού και της κληρονομικότητας. Χρειάζεστε επίσης την υλοποίηση των κατάλληλων **ArrayLists**. Θα πρέπει επίσης να χρησιμοποιήσετε τεχνική χειρισμού εξαιρέσεων για τη διαδικασία εισαγωγής δεδομένων. Για λόγους απλότητας, υποθέστε όλες τις αριθμητικές μεταβλητές ως ακέραιους αριθμούς.

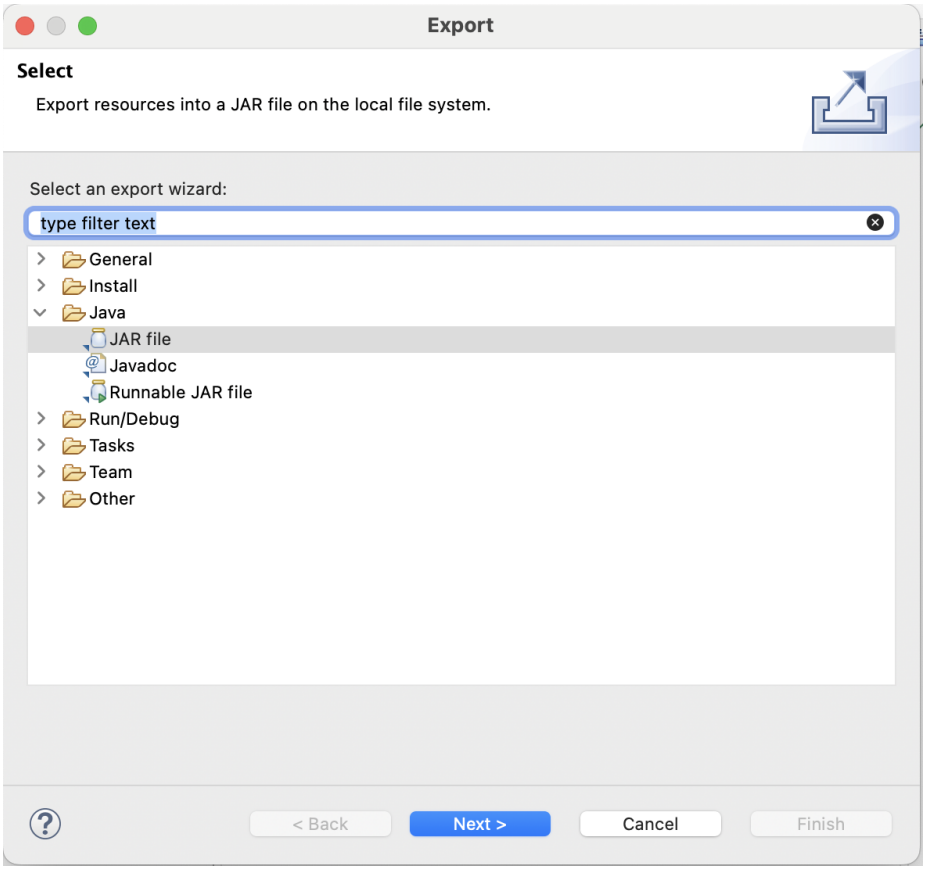
Η ενδεικτική λύση της δεύτερης έκδοσης του project βρίσκεται στο αρχείο με όνομα **MyBnbProject_v3.jar** που βρίσκεται στα συνοδευτικά αρχεία του εκπαιδευτικού οδηγού.

10.4. Δημιουργία jar με χρήση του Eclipse

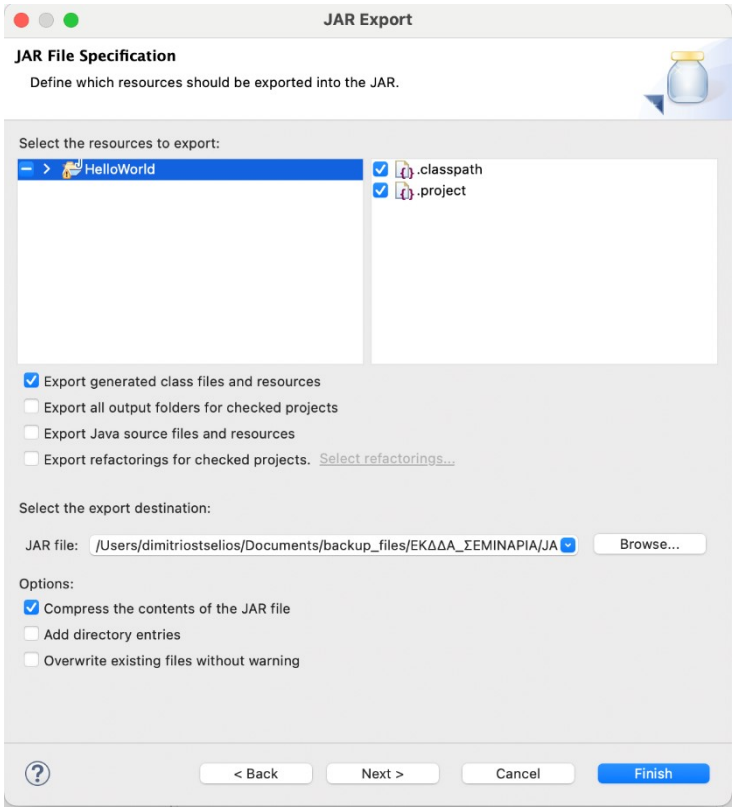
Προφανώς όταν ένα μεγάλο project αποτελείται από πολλά αρχεία κλάσεων η διακίνηση τους δεν μπορεί να γίνει αποσπασματικά αλλά είναι ευκολότερη η συνένωση τους σε ένα αρχείο **jar**. Το **Eclipse** δίνει αυτή τη δυνατότητα μέσω της εξαγωγής του project από την επιλογή **Export** όπως φαίνεται στην παρακάτω εικόνα.



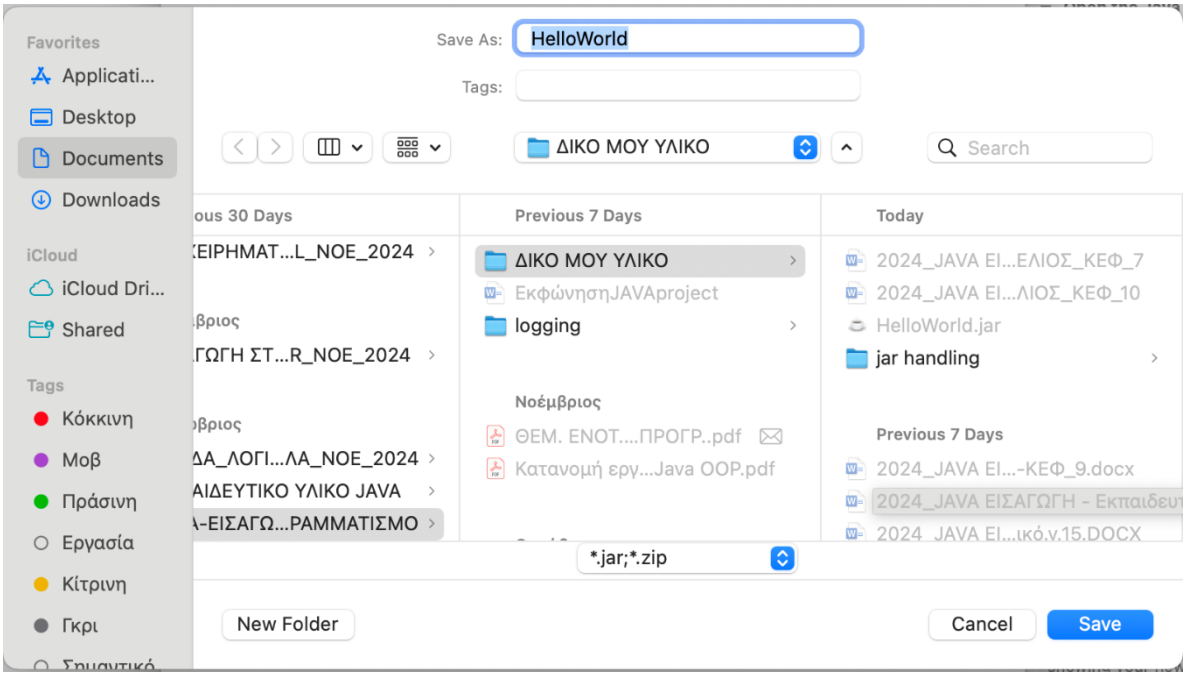
Στη συνέχεια επιλέγεται ο τύπος του αρχείου που είναι ο **JAR file** όπως παρουσιάζεται στην επόμενη εικόνα.



Έπειτα επιλέγεται το περιεχόμενο που θα αποθηκευτεί στο αρχείο jar.



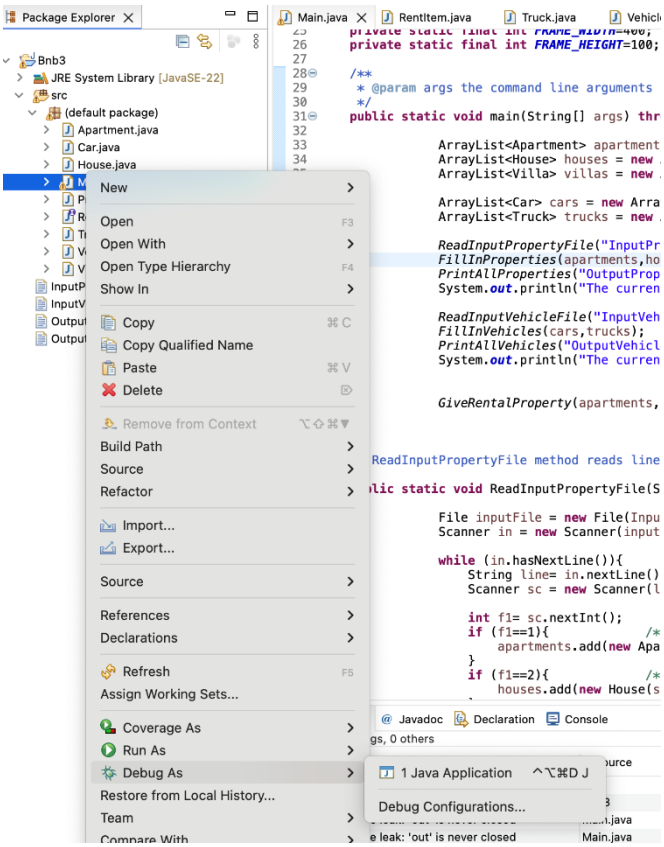
Τέλος, επιλέγεται ο αποθηκευτικός χώρος και το όνομα αρχείου όπως φαίνεται στην ακόλουθη εικόνα.



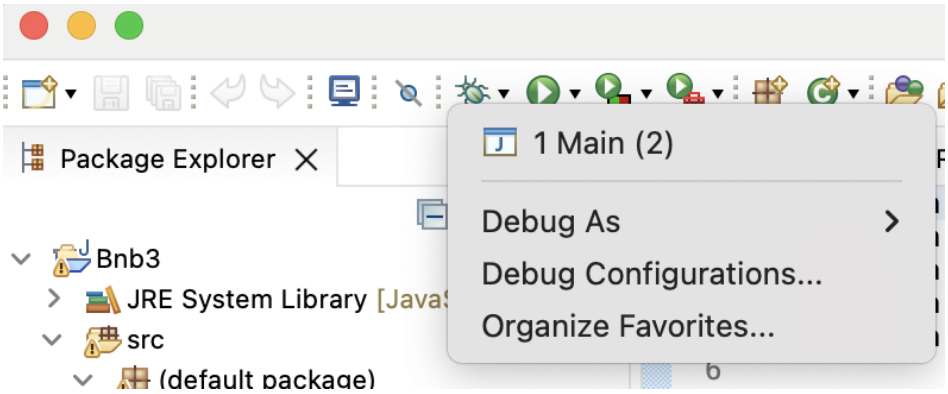
10.5. Έλεγχος και Debugging

Ο έλεγχος της ορθότητας του κώδικα γίνεται προφανώς άμεσα με την προσπάθεια εκτέλεσης του. Όμως τα σύγχρονα προγραμματιστικά εργαλεία προσφέρουν και αυτοματοποιημένες δυνατότητες εντοπισμού των λαθών και μετά τη διόρθωση τους. Το **Debugging** είναι η διαδικασία αναζήτησης, απομόνωσης και επίλυσης των λαθών του κώδικα τα οποία έχουν ονομαστεί ως **bugs** στα προγράμματα λογισμικού. Το **Debugging** βοηθά στην αποκάλυψη των αιτιών των λαθών του κώδικα, λειτουργεί προληπτικά και αυξάνει την απόδοση του. Σε αντίθεση με τη δοκιμή του λογισμικού που επιτρέπει στον προγραμματιστή να δει απλώς τις συνέπειες των λαθών, το **Debugging** αναζητά την αιτία τους και προχωρά στην επιδιόρθωση.

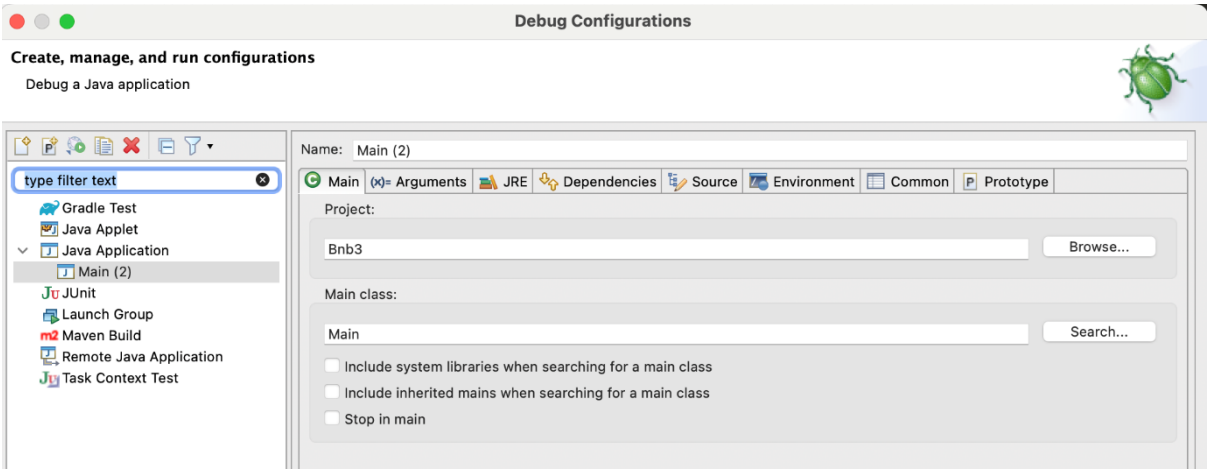
Όπως κάθε **IDE** έτσι και το **Eclipse** παρέχει την αυτοματοποιημένη διαδικασία **Debugging** η οποία ενεργοποιείται με τις εξής ενέργειες: αφού επιλεγεί το αρχείο της κλάσης που θα ελεγχθεί με δεξί κλικ και μέσα από την επιλογή του μενού **Debug As →Java Application** όπως φαίνεται στην ακόλουθη εικόνα.



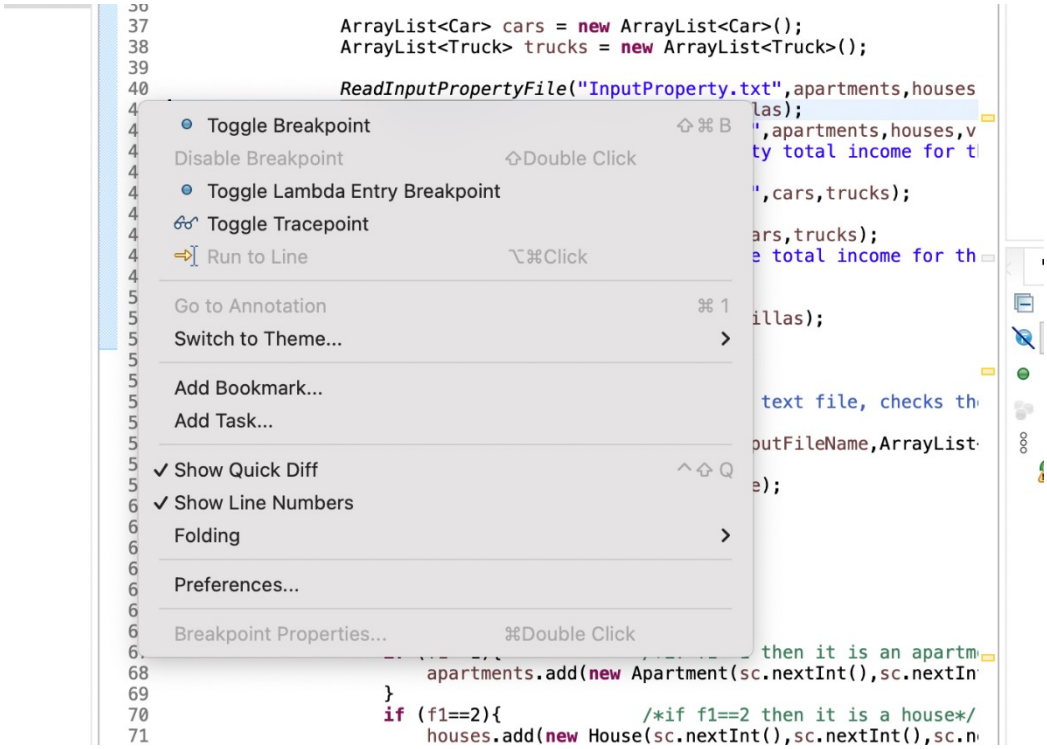
Αυτή η επιλογή θα δημιουργήσει ένα νέο **Debug Configuration** το οποίο είναι διαχειρίσιμο από το κατάλληλο εικονίδιο του **Debugger** στην κορδέλα εργαλείων του Eclipse.



Εκεί υπάρχει η επιλογή **Debug Configuration** το οποίο θα ανοίξει το κατάλληλο παράθυρο του Configuration όπως φαίνεται στην επόμενη εικόνα.



Μία δυνατότητα που δίνεται από τον **Debugger** είναι η υπόδειξη κάποιων σημείων στον κώδικα ως **breakpoints** για να σταματάει προσωρινά η εκτέλεση του προγράμματος έτσι ώστε να είναι δυνατή η παρακολούθηση των πληροφοριών του προγράμματος. Η ρύθμιση αυτή γίνεται εύκολα με την επιλογή του δεξιού κλικ στον αριθμό της γραμμής του κώδικα όπως φαίνεται στην παρακάτω εικόνα.


ArrayList<Car> cars = new ArrayList<Car>();
ArrayList<Truck> trucks = new ArrayList<Truck>();
ReadInputPropertyFile("InputProperty.txt",apartments,houses
Toggle Breakpoint
Disable Breakpoint
Toggle Lambda Entry Breakpoint
Toggle Tracepoint
Run to Line
Go to Annotation
Switch to Theme...
Add Bookmark...
Add Task...
Show Quick Diff
Show Line Numbers
Folding
Preferences...
Breakpoint Properties...
Double Click
text file, checks th
then it is an apartm
apartments.add(new Apartment(sc.nextInt(),sc.nextIn
if (f1==2){ /*if f1==2 then it is a house*/
houses.add(new House(sc.nextInt(),sc.nextInt(),sc.n

ΒΙΒΛΙΟΓΡΑΦΙΑ

Cay S. Horstmann, Big Java 4th Edition, John Wiley and Sons (Asia) Pte Ltd, New York 2010

Michael T. Goodrich and Roberto Tamassia, Data Structures & Algorithms in Java, John Wiley and Sons (Asia) Pte Ltd, New York 2011