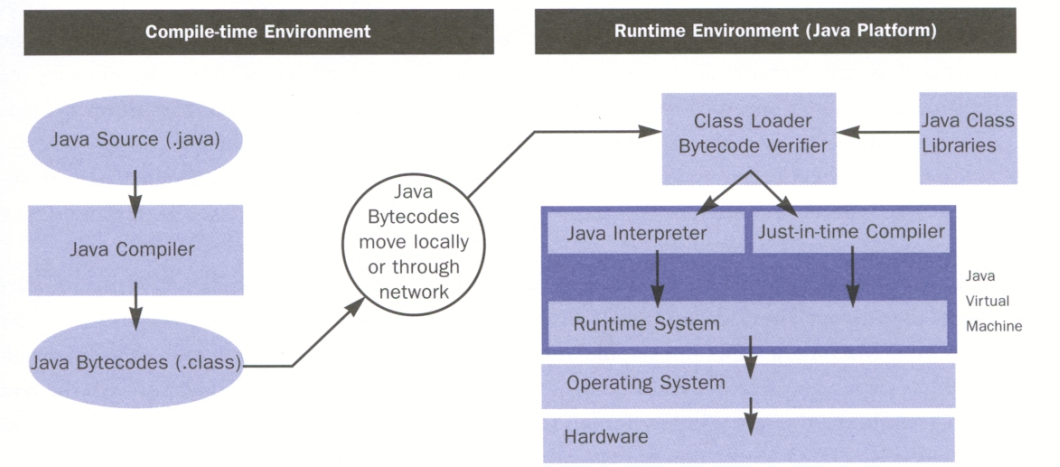
ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ
ΤΜΗΜΑ ΤΕΧΝΟΛΟΓΙΚΗΣ ΕΚΠΑΙΔΕΥΣΗΣ

ΣΗΜΕΙΩΣΕΙΣ ΓΙΑ ΤΟ ΜΑΘΗΜΑ
ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΕΦΗΣ ΣΧΕΔΙΑΣΜΟΣ ΚΑΙ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ

Εισαγωγή στη γλώσσα Java

Δρ. Βασίλειος Βεσκούκης
Δρ. Αναστάσιος Κουτουμάνος



ΠΕΙΡΑΙΑΣ, 2000

Περιεχόμενα

1. Εισαγωγή 7

1.1. Μια νέα γλώσσα προγραμματισμού 7

1.2. Χαρακτηριστικά της Java 8

1.3. Java και World Wide Web 11

1.4. Εκτέλεση προγραμμάτων 12

1.5. Είδη προγραμμάτων Java 13

1.6. Ένα απλό παράδειγμα κώδικα Java 14

1.6.1. Αυτόνομη εφαρμογή Java 14

1.6.2. Java Applet 15

2. Η γλώσσα Java 17

2.1. Κώδικας προγραμμάτων 17

2.2. Σχόλια 17

2.3. Τύποι δεδομένων και σταθερές 18

2.3.1. Βασικοί τύποι 18

2.3.2. Τύποι αναφοράς 20

2.3.3. Ειδικοί τύποι 21

2.4. Εκφράσεις 22

2.4.1. Αριθμητικοί τελεστές και τελεστές σύγκρισης 24

2.4.2. Τελεστές bit προς bit 25

2.4.3. Λογικοί τελεστές 25

2.4.4. Τελεστές εκχώρησης 26

2.5. Εντολές 27

2.5.1. Εντολές εκφράσεων 27

2.5.2. Εντολές επιλογής 28

2.5.3. Εντολές βρόχων 29

2.5.4. Άλλες εντολές 30

3. Αντικείμενα και κλάσεις 31

3.1. Αντικειμενοστρεφής προγραμματισμός 31

3.2. Αντικείμενα και κλάσεις 32

3.2.1. Υπερφόρτωση μεθόδων 35

3.2.2. Κατασκευή αντικειμένων 35

3.2.3. Καταστροφή αντικειμένων 37

3.2.4. Μετατροπείς ορατότητας 38

3.3. Ιεραρχίες κλάσεων 39

3.3.1. Κληρονομικότητα και κατασκευαστές 40

3.3.2. Υπο-τύποι και μετατροπή τύπων 41

3.3.3. Υπερκάλυψη μεταβλητών και υπερίσχυση μεθόδων 42

3.3.4. Αφηρημένες και τελικές κλάσεις 44

3.4. Διαπροσωπείες 46
3.5. Πακέτα 47
3.5.1. Δημιουργία πακέτων 48
3.5.2. Πρόσβαση σε μέλη κλάσεων 48
3.6. Παράδειγμα: Βιβλιοθήκη σχημάτων 50

4. Εξαιρέσεις και νήματα 57
4.1. Χειρισμός εξαιρέσεων 57
4.1.1. Κλάσεις εξαιρέσεων και σφαλμάτων 57
4.1.2. Εντολές χειρισμού εξαιρέσεων 59
4.1.3. Πρόκληση εξαιρέσεων 60
4.1.4. Εξαιρέσεις και κλήση μεθόδων 61
4.1.5. Ανάνηψη από εξαιρέσεις 62
4.2. Πολλαπλά νήματα εκτέλεσης 63
4.2.1. Δημιουργία νημάτων 64
4.2.2. Χειρισμός νημάτων 65
4.2.3. Συγχρονισμός νημάτων 66
4.2.4. Παράδειγμα: Οι φιλόσοφοι που γευματίζουν 68

5. Βοηθητικές κλάσεις 73
5.1. Συμβολοσειρές 73
5.1.1. Κατασκευή και απλές λειτουργίες 73
5.1.2. Μετατροπές από και προς συμβολοσειρές 74
5.1.3. Σύγκριση συμβολοσειρών 75
5.1.4. Επεξεργασία συμβολοσειρών 76
5.1.5. Αποθήκες συμβολοσειρών 76
5.1.6. Διαχωρισμός συμβολοσειρών 77
5.2. Μαθηματικές συναρτήσεις 77
5.2.1. Περιτυλίγματα βασικών τύπων 79
5.2.2. Γεννήτρια τυχαίων αριθμών 80
5.3. Ημερομηνίες 80
5.4. Συλλογές δεδομένων 81
5.4.1. Δυναμικοί πίνακες 82
5.4.2. Απαριθμήσεις 83
5.4.3. Πίνακες κατακερματισμού 83
5.4.4. Πίνακας ιδιοτήτων 84

6. Είσοδος και έξοδος 87
6.1. Εισαγωγή 87
6.2. Ρεύματα 87
6.2.1. Απλή Ε/Ε 89
6.2.2. Φιλτραρισμένα ρεύματα 90
6.2.3. Ρεύματα Δεδομένων 91
6.2.4. Ρεύματα με προσωρινή αποθήκευση 92
6.2.5. Ρεύματα εκτύπωσης (Print Streams) 92

6.2.6. Αγωγοί 92
6.3. Αρχεία 93
6.3.1. Διαχείριση αρχείων 93
6.3.2. Ρεύματα αρχείων 94
6.4. Παράδειγμα: Ανάγνωση αρχείων 95

7. Χειρισμός γραφικών 97
7.1. Εισαγωγή 97
7.2. Applets 97
7.3. Συστατικά 99
7.3.1. Συστατικά, αποθήκες και γεγονότα 100
7.3.2. Χρήση των συστατικών 102
7.3.3. Χαρακτηριστικά των συστατικών 106
7.4. Διαχειριστές διάταξης 106
7.5. Χειρισμός γεγονότων 108
7.6. Γραφικά και ήχοι 109
7.6.1. Μέθοδοι σχεδίασης 110
7.6.2. Χρώματα 110
7.6.3. Γραμματοσειρές 111
7.6.4. Εικόνες 112
7.6.5. Ήχος 113
7.7. Αυτόνομες εφαρμογές 113
7.8. Παράδειγμα: Το πλήρες πρόγραμμα σχεδίασης 114
7.9. Παράδειγμα: Εμφάνιση αρχείων σε παράθυρο 121

1. Εισαγωγή

Στο κεφάλαιο αυτό γίνεται μια εισαγωγή στη γλώσσα Java. Περιγράφονται τα βασικά της χαρακτηριστικά και το ευρύ φάσμα εφαρμογών της σε συνδυασμό με το World Wide Web. Στη συνέχεια, περιγράφεται συνοπτικά η διαδικασία εκτέλεσης προγραμμάτων Java και δίνεται ένα απλό παράδειγμα χρήσης της γλώσσας.

1.1. Μια νέα γλώσσα προγραμματισμού

Οι εφαρμογές υπολογιστών που αναπτύσσονται τα τελευταία χρόνια, οποιοδήποτε και αν είναι το αντικείμενό τους και το είδος χρηστών τους, σχεδόν σίγουρα θα τρέχουν σε πολλές μηχανές, συνδεδεμένες σε κάποιο τοπικό ή ευρύτερο δίκτυο διασύνδεσης. Η αυξανόμενη σημασία των δικτύων υπολογιστών θέτει νέες απαιτήσεις από τα προγραμματιστικά εργαλεία και δημιουργεί απαιτήσεις για ένα νέο και συνεχώς αυξανόμενο σύνολο εφαρμογών.

Το αναπτυσσόμενο λογισμικό απαιτείται συνήθως να λειτουργεί σωστά σε πολλά διαφορετικά περιβάλλοντα και αρχιτεκτονικές υπολογιστών, καθώς και να συνεργάζεται με άλλες εφαρμογές. Το λογισμικό πρέπει να εκμεταλλεύεται τις δυνατότητες της σύγχρονης δικτυακής δομής υπολογιστικών συστημάτων και να είναι σε θέση να προσπελαύνει τις κατανεμημένες πηγές πληροφοριών, να συνδυάζει τις αντλούμενες πληροφορίες και να τις παρουσιάζει στο χρήστη σε κατάλληλη μορφή.

Το πρόβλημα που δημιουργείται είναι ότι οι απαιτήσεις για ταχύτητα και μεταφερσιμότητα είναι συνήθως αντικρουόμενες, ενώ στο θέμα της ασφάλειας δεν έχει δοθεί η απαραίτητη σημασία. Υπάρχουν γλώσσες προγραμματισμού οι οποίες είναι μεν μεταφέρσιμες αλλά και αργές, λόγω του ότι τα προγράμματα ερμηνεύονται αντί να μεταγλωττίζονται. Οι γλώσσες αυτές είναι αρκετά διαδεδομένες, τόσο λόγω της υψηλή λειτουργικότητάς τους όσο και τη μεταφερσιμότητάς τους. Επίσης, υπάρχουν γλώσσες προγραμματισμού οι οποίες είναι γρήγορες, αλλά η ταχύτητά τους απορρέει από το ότι είναι σχεδιασμένες για συγκεκριμένες υπολογιστικές αρχιτεκτονικές.

Από μεθοδολογικής πλευράς η ανάπτυξη λογισμικού τα τελευταία χρόνια έχει προσανατολιστεί κυρίως προς τον *αντικειμενοστρεφή προγραμματισμό* (object-oriented programming), ο οποίος επιχειρεί να δαμάσει τη συνεχώς αυξανόμενη πολυπλοκότητα ανάπτυξης λογισμικού. Σύμφωνα με τον αντικειμενοστρεφή προγραμματισμό, το λογισμικό δομείται σε αυτόνομες μονάδες οι οποίες έχουν σαφή λειτουργικότητα και διαπροσωπεία.

Η Java είναι μια νέα γλώσσα προγραμματισμού και επιχειρεί να δώσει λύση στα προβλήματα που αναφέρθηκαν παραπάνω. Αναπτύχθηκε πρόσφατα από την εταιρεία Sun και στο λίγο χρόνο από την ανάπτυξή της έχει γνωρίσει αρκετά μεγάλη διάδοση. Αρχικά ήταν προσανατολισμένη στην ανάπτυξη λογισμικού για ηλεκτρονικές συσκευές οικιακής χρήσης, στη συνέχεια όμως εξελίχθηκε σε μια ολοκληρωμένη γλώσσα, η οποία έχει αρκετά από τα χαρακτηριστικά των μοντέρνων γλωσσών προγραμματισμού και υποστηρίζει τον αντικειμενοστρεφή προγραμματισμό. Η επιτυχία της γλώσσας έγκειται στο ότι μπορεί να χρησιμοποιηθεί για προγραμματισμό ασφαλών, υψηλής απόδοσης εφαρμογών σε Internet, οι οποίες μπορούν να τρέξουν αυτούσιες σε διαφορετικά προγραμματιστικά περιβάλλοντα και αρχιτεκτονικές, καθώς και ότι παρέχει τη δυνατότητα μεταφοράς δυναμικού περιεχομένου σε εφαρμογές πολυμέσων.

1.2. Χαρακτηριστικά της Java

Η εταιρεία Sun περιγράφει τη Java ως “μια απλή, αντικειμενοστρεφή, κατανεμημένη, ερμηνευόμενη, συμπαγή, ασφαλή, ανεξάρτητη αρχιτεκτονικής, μεταφέρσιμη, υψηλής απόδοσης, υποστηρίζουσα πολλαπλά νήματα, και δυναμική γλώσσα”. Επίσης, η Java υποστηρίζει εφαρμογές πολυμέσων. Όμως, η έννοια που αποδίδεται στα περισσότερα από αυτά τα χαρακτηριστικά δεν είναι προφανής. Ας τα εξετάσουμε ένα προς ένα.

Απλή

Η Java είναι μια απλή (simple) γλώσσα, με την έννοια ότι η εκμάθησή της είναι σχετικά εύκολη. Περιέχει λίγες προγραμματιστικές δομές με καλά ορισμένη σημασιολογία. Μοιάζει αρκετά με τις ευρέως διαδεδομένες γλώσσες προγραμματισμού C και C++ και, ως εκ τούτου, η μετάβαση από αυτές τις γλώσσες στη Java είναι αρκετά εύκολη. Ορισμένα χαρακτηριστικά των C και C++ έχουν αφαιρεθεί από τη Java, όπως η διαχείριση μνήμης από τον προγραμματιστή. Άλλα πάλι χαρακτηριστικά υπάρχουν στη Java από το σχεδιασμό της, όπως για παράδειγμα η αυτόματη διαχείριση της μνήμης, ο χειρισμός πολλαπλών νημάτων εκτέλεσης, κ.λπ.

Αντικειμενοστρεφής

Ως αντικειμενοστρεφής (object-oriented) γλώσσα, η Java εστιάζει τη δραστηριότητα του προγραμματιστή στον ορισμό αντικειμένων και λειτουργιών πάνω σε αυτά. Στη Java ιδιαίτερη σημασία έχει η έννοια της κλάσης. Μια κλάση περιγράφει μια συλλογή δεδομένων καθώς και τις λειτουργίες που αυτά επιδέχονται. Κάθε κλάση προέρχεται από κάποια άλλη ήδη ορισμένη κλάση μεσω κληρονομικότητας (inheritance). Κατ’ αυτό τον τρόπο, ορίζεται μια ιεραρχία κλάσεων, η οποία έχει στην κορυφή της τη βασική κλάση `Object`. Μια κλάση είναι απλά μια περιγραφή της μορφής των αντικειμένων τα οποία περιέχει. Αντικείμενα μιας κλάσης μπορούν να οριστούν και να χρησιμοποιηθούν σε ένα πρόγραμμα Java. Κάθε αντικείμενο δημιουργείται κατά τη διάρκεια εκτέλεσης ενός προγράμματος Java και υπάρχει μεχρι να καταστραφεί.

Κατανεμημένη

Η Java χαρακτηρίζεται ως κατανεμημένη (distributed) γλώσσα προγραμματισμού γιατί υποστηρίζει την ανάπτυξη κατανεμημένων εφαρμογών δικτύων. Συγκεκριμένα, η Java επιτρέπει

την προσπέλαση αντικειμένων που βρίσκονται σε απομακρυσμένες θέσεις στο δίκτυο, καθώς και τη δικτυακή επικοινωνία με άλλες εφαρμογές.

Ερμηνευόμενη

Η Java είναι ερμηνευόμενη (interpreted) γλώσσα. Ο μεταγλωττιστής δεν παράγει τελικό κώδικα, για κάποιο συγκεκριμένο υπολογιστή, αλλά ενδιάμεσο κώδικα σε μορφή bytes, που ονομάζεται bytecode. Ο ενδιάμεσος κώδικας στη συνέχεια εκτελείται από ένα διερμηνέα (interpreter) της Java. Το πλεονέκτημα είναι ότι ο κώδικας μπορεί να εκτελεστεί σε πολλά διαφορετικά περιβάλλοντα υπολογιστών, αν φυσικά κάποιος διερμηνέας Java είναι διαθέσιμος σε αυτά.

Ο διερμηνέας της γλώσσας μαζί με το σύστημα εκτέλεσης προγραμμάτων (run-time system) υλοποιεί μια εικονική μηχανή Java (Java Virtual Machine, JVM). Ο διερμηνέας έχει αναπτυχθεί για αρκετά περιβάλλοντα, συμβάλλοντας έτσι στη μεταφερσιμότητα του εκτελέσιμου κώδικα της γλώσσας. Προγράμματα Java μπορούν επίσης να εκτελεστούν από τους περισσότερους σύγχρονους περιηγητές του WWW, καθένας από τους οποίους έχει ενσωματωμένη μια εικονική μηχανή Java για την αναγνώριση και εκτέλεση του ενδιάμεσου κώδικα.

Συμπαγής

Η Java έχει σχεδιαστεί ώστε να είναι δυνατή η ανάπτυξη συμπαγούς (compact) και αξιόπιστου λογισμικού. Η γλώσσα έχει ένα ισχυρό σύστημα τύπων, το οποίο επιτρέπει εκτενείς ελέγχους κατά τη διάρκεια της μετάφρασης των προγραμμάτων, βοηθώντας έτσι την ανάπτυξη αξιόπιστου λογισμικού.

Ένα άλλο στοιχείο το οποίο βοηθάει την ανάπτυξη αξιόπιστου λογισμικού είναι το μοντέλο μνήμης. Η Java δεν παρέχει στον προγραμματιστή εντολές για παραχώρηση και αποδέσμευση μνήμης, ή χρησιμοποίηση δεικτών προς τη μνήμη. Η διαχείριση της μνήμης γίνεται εξ ολοκλήρου από το σύστημα εκτέλεσης. Κατά τη δημιουργία ενος αντικειμένου, το σύστημα εκτέλεσης αναλαμβάνει να δεσμεύσει το απαραίτητο ποσό μνήμης γιά την αποθήκευση του αντικειμένου. Επιπλέον, όταν το σύστημα εκτέλεσης εντοπίσει κάποιο αντικείμενο το οποίο είναι αδύνατο να προσπελαστεί από οποιοδήποτε σημείο του εκτελούμενου προγράμματος, αναλαμβάνει να αποδεσμεύσει τη μνήμη που καταλαμβάνει το αντικείμενο και να την κάνει διαθέσιμη στο υπόλοιπο πρόγραμμα.

Ασφαλής

Η Java είναι μια ασφαλής (safe) γλώσσα. Προγράμματα τα οποία έχουν αναπτυχθεί με αυτήν μπορούν να εκτελεστούν από πολλούς χρήστες με διαφορετικά δικαιώματα πρόσβασης στο σύστημα, χωρίς να είναι δυνατό να υπάρξουν ανεπιθύμητες παρενέργειες, εκούσιες ή ακούσιες. Αυτό επιτυγχάνεται με ένα ειδικό πρόγραμμα που ονομάζεται ελεγκτής ενδιάμεσου κώδικα (bytecode verifier), ο οποίος ερευνά τον ενδιάμεσο κώδικα και εντοπίζει “ύποπτα” σημεία, των οποίων η εκτέλεση θα μπορούσε να προκαλέσει ανεπιθύμητες ενέργειες στο υπολογιστικό περιβάλλον του χρήστη. Αυτό είναι αναγκαίο να γίνει, καθ’ όσον ο ενδιάμεσος κώδικας μπορεί να παραχθεί με πολλούς τρόπους και όχι απαραίτητα με μετάφραση πηγαίου κώδικα από έναν “έμπιστο” μεταφραστή.

Ανεξάρτητη αρχιτεκτονικής

Τα προγράμματα Java μεταφράζονται σε ενδιάμεσο κώδικα, ο οποίος δεν είναι προσανατολισμένος σε μια συγκεκριμένη αρχιτεκτονική ή τύπο υπολογιστή. Ο κώδικας είναι μεταφέρσιμος, μια και μπορει να εκτελεστεί από διερμηνείς της γλώσσας που έχουν αναπτυχθεί γιά πολλές αρχιτεκτονικές και περιβάλλοντα υπολογιστών. Γιά παράδειγμα, τέτοιοι διερμηνείς έχουν αναπτυχθεί για μηχανές της Sun και της IBM, καθώς και για τα λειτουργικά συστήματα Unix, Windows 95, Windows NT, κ.λπ.

Μεταφέρσιμη

Το γεγονός ότι η Java είναι ανεξάρτητη αρχιτεκτονικής καθιστά τα προγράμματα Java μεταφέρσιμα. Οι προδιαγραφές της Java καθορίζουν σαφώς τα μεγέθη των διαφόρων πρωταρχικών τύπων δεδομένων καθώς και τα αποτελέσματα των διαφόρων λειτουργιών πάνω σε μεταβλητές αυτών των τύπων. Κατά συνέπεια, ένα πρόγραμμα Java θα δώσει τα ίδια αποτελέσματα όταν δεχθεί τα ίδια δεδομένα, ανεξάρτητα της αρχιτεκτονικής στην οποία θα εκτελεστεί.

Υψηλής απόδοσης

Η Java σαν ερμηνευόμενη γλώσσα δεν μπορεί να φτάσει την απόδοση γλωσσών προγραμματισμού όπως η C και η C++, που υλοποιούνται σε μεταγλωττιστές. Παρ' ολα αυτά, η απόδοση της γλώσσας είναι αρκετά υψηλή. Κατά μέσο όρο η εκτέλεση προγραμμάτων Java είναι περίπου 20% αργότερη από την εκτέλεση των αντίστοιχων προγραμμάτων σε C ή C++.

Σε μερικές περιπτώσεις όμως, η απόδοση θεωρείται σημαντικός παράγοντας. Γι' αυτό το λόγο έχουν αναπτυχθεί μεταγλωττιστές "τελευταίας στιγμής" (just-in-time compilers), που εμπεριέχονται στους διερμηνείς και μεταφράζουν τον ενδιάμεσο κώδικα σε κώδικα εκτελέσιμο από τη συγκεκριμένη μηχανή. Το αποτέλεσμα είναι σημαντική βελτίωση της απόδοσης εκτέλεσης προγραμμάτων.

Υποστηρίζουσα πολλαπλά νήματα

Η Java υποστηρίζει πολλαπλά νήματα εκτέλεσης (multi threaded). Τα πολλαπλά νήματα εκτέλεσης αποτελούν ένα προγραμματιστικό μοντέλο για ανάπτυξη λογισμικού, που απαιτεί την "ταυτόχρονη" εκτέλεση πολλών διεργασιών. Οι διεργασίες μοιράζονται τον ίδιο χώρο μνήμης για την εκτέλεσή τους και τη διαχείριση δεδομένων. Τα πολλαπλά νήματα εκτέλεσης αποτελούν επίσης και μια αποδοτική λύση σε πολλά προβλήματα, αφού συνήθως η εναλλαγή εκτέλεσης ανάμεσα στα διάφορα νήματα δεν είναι ιδιαίτερα δαπανηρή. Παραθυρικές εφαρμογές πού χρησιμοποιούν πολλαπλά νήματα εκτέλεσης παρουσιάζουν βελτιωμένη απόκριση προς το χρήστη και, από προγραμματιστικής άποψης, είναι συνήθως ευκολότερες να αναπτυχθούν.

Η υποστήριξη πολλαπλών νημάτων εκτέλεσης απαιτεί την ύπαρξη μηχανισμών συγχρονισμού μεταξύ των νημάτων. Αυτό είναι απαραίτητο προκειμένου να αποφεύγεται η ταυτόχρονη προσπέλαση του ιδίου αντικειμένου, κάτι που επιβάλλεται συνήθως στις περισσότερες εφαρμογές.

Δυναμική

Η Java είναι μια δυναμική (dynamic) γλώσσα, η οποία έχει σχεδιαστεί για προσαρμογή σε ένα δυναμικά εξελισσόμενο περιβάλλον. Κλάσεις μπορούν να μεταφερθούν δυναμικά από άλλα μέρη του δικτύου και να εκτελεστούν τοπικά. Κατ' αυτό τον τρόπο, προγράμματα Java μπορούν να μεταφέρουν και να εκτελέσουν κλάσεις από άλλα μέρη του δικτύου και δεν είναι απαραίτητο αυτές οι κλάσεις να ενσωματωθούν στο πρόγραμμα κατά τη διάρκεια της μετάφρασής του.

Υποστήριξη πολυμέσων

Η Java καθιστά δυνατή τη μεταφορά εκτελέσιμου περιεχομένου σε εφαρμογές πολυμέσων. Πριν τη Java, οι εφαρμογές πολυμέσων περιείχαν κείμενο, ακίνητη ή κινούμενη εικόνα, ήχο, κ.λπ. Αυτό πού έλειπε ήταν η δυνατότητα εκτέλεσης προγραμμάτων στο περιβάλλον του χρήστη. Αυτό το κενό καλύπτεται από τη Java, η οποία επιτρέπει να μεταφερθεί εκτελέσιμος κώδικας και να εκτελεσθεί στον περιηγητή του χρήστη. Ένα πλεονέκτημα είναι ότι κατ' αυτό τον τρόπο οι δυνατότητες των περιηγητών μπορούν να επεκταθούν χωρίς περιορισμούς, κάτι που απορρέει επίσης από το ότι η γλώσσα είναι δυναμική.

1.3. Java και World Wide Web

Το Internet έχει εξελιχθεί ραγδαία σε έναν άμορφο ωκεανό από δεδομένα, αποθηκευμένα με πολλές διαφορετικές κωδικοποιήσεις σε ένα πλήθος υπολογιστών. Κατά καιρούς, διάφορα πρωτόκολλα αποθήκευσης και μεταφοράς δεδομένων έχουν αναπτυχθεί με σκοπό να επιβάλλουν κάποια μορφή τάξης σε αυτό το χάος. Μια από τις ταχύτερα αναπτυσσόμενες περιοχές του Internet είναι το WWW (World Wide Web), το οποίο χρησιμοποιεί ένα σύστημα οργάνωσης και κωδικοποίησης των πληροφοριών σε υπερκείμενα (hypertext), προκειμένου να διευκολύνει την "πλοήγηση" των χρηστών μέσα στον ωκεανό των δεδομένων.

Η έννοια του υπερκειμένου δεν είναι κατά κανένα τρόπο νέα, αλλά η πραγμάτωση της χρειάστηκε αρκετές δεκαετίες. Η πραγματική δύναμη του υπερκειμένου αποκαλύφθηκε πρόσφατα, με την προσθήκη της δυνατότητας δημιουργίας υπερσυνδέσμων μεταξύ κειμένων που βρίσκονται σε διαφορετικούς υπολογιστές, συνδεδεμένους μέσω δικτύου. Η πρώτη πρακτική, αν και απλοϊκή, υλοποίηση ενός συστήματος υπερμέσων (hypermedia) σε περιβάλλον δικτύου αναπτύχθηκε από τον Tim Berners-Lee στο CERN, στα τέλη της δεκαετίας του 1980. Μέσα σε λίγα χρόνια αναπτύχθηκε η γλώσσα περιγραφής υπερκειμένων HTML (HyperText Markup Language), το πρωτόκολλο μεταφοράς δεδομένων HTTP (HyperText Transport Protocol) και τελικά το WWW.

Οι περιηγητές (browsers) του WWW είναι τα προγράμματα που αναλαμβάνουν την περιήγηση των χρηστών στο σύστημα υπερκειμένων. Εκτός από τη λειτουργία της προσκόμισης δεδομένων στο χρήστη, οι περιηγητές αντιλαμβάνονται το είδος των δεδομένων ώστε να μπορέσουν να τα επιδείξουν με το σωστό τρόπο, αν αυτό είναι δυνατό. Η πιο συχνή κωδικοποίηση που συναντάται σήμερα στο WWW είναι φυσικά αρχεία κειμένου, που περιέχουν εντολές μορφοποίησης στη γλώσσα HTML. Τα αρχεία αυτά είναι γνωστά και ως σελίδες HTML, λόγω της απεικόνισής τους από τους περιηγητές ως μορφοποιημένες σελίδες. Η δύναμη της HTML δεν περιορίζεται στη δυνατότητα μορφοποίησης του κειμένου, αλλά και στον απλό τρόπο ορισμού υπερσυνδέσμων προς άλλα δεδομένα που βρίσκονται αποθηκευμένα σε υπολογιστές του Internet.

Παρά το γεγονός ότι μια μεγάλη ποικιλία διαφορετικών μορφών εγγράφων μπορούν να περιέχονται στις σελίδες HTML, οι σελίδες αυτές στερούνται ακόμα δυναμικής συμπεριφοράς. Τα στοιχεία που περιέχονται σε αυτές είναι παθητικά: πρόκειται για ηλεκτρονικά έγγραφα που μπορεί κανείς να διαβάσει, δει, ακούσει, παρακολουθήσει, αλλά δεν είναι σε θέση να αλληλεπιδρούν με τον αναγνώστη, εκτός από αυτό τον προφανή τρόπο. Η ενσωμάτωση εκτελέσιμων προγραμμάτων στις σελίδες HTML είναι κάτι που επιζητείται εδώ και πολλά χρόνια, εμποδίζεται όμως από μια σειρά δύσκολων προβλημάτων, από τα οποία σημαντικότερα είναι τα εξής δυο:

- *Μεταφερσιμότητα:* Ο εκτελέσιμος κώδικας θα πρέπει να είναι σε μια μορφή που να αναγνωρίζεται από κάθε περιβάλλον υπολογιστή, για τον οποίο υπάρχουν περιηγητές.
- *Ασφάλεια:* Πρέπει να εξασφαλίζεται ότι η εκτέλεση του κώδικα δεν είναι δυνατό να βλάψει, εκούσια ή ακούσια, το υπολογιστικό σύστημα στο οποίο τρέχει ο περιηγητής.

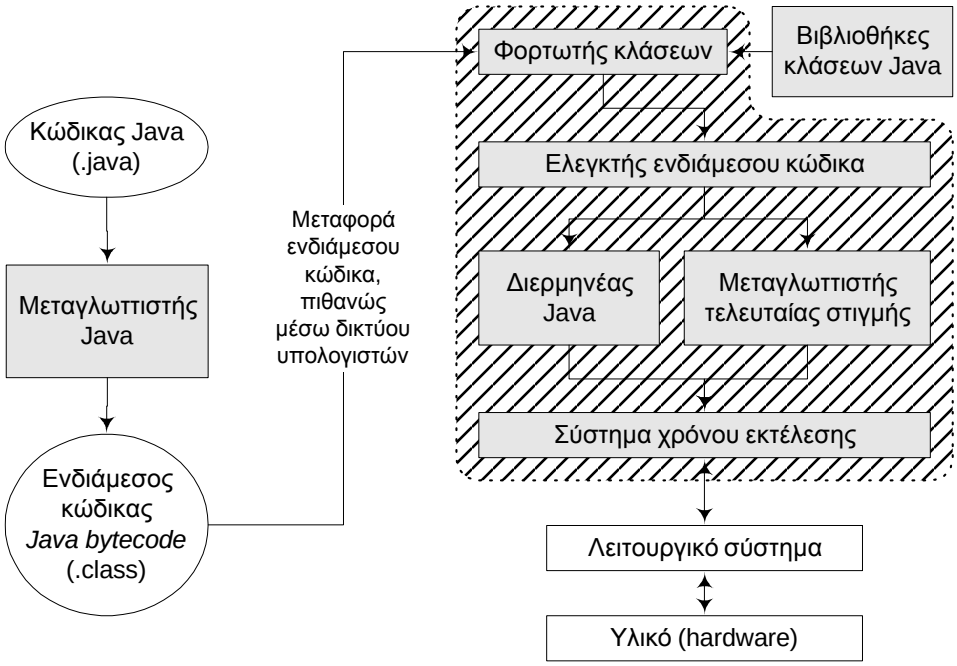
Η γλώσσα Java αποτελεί μια λύση στο πρόβλημα δυναμικής συμπεριφοράς, επιχειρώντας να ξεπεράσει αυτά τα δυο προβλήματα, με αρκετά μεγάλη επιτυχία. Για το λόγο αυτό, η Java συνδέεται άμεσα με το WWW.

1.4. Εκτέλεση προγραμμάτων

Στο Σχήμα 1.1 απεικονίζεται η διαδικασία μεταγλώττισης και εκτέλεσης προγραμμάτων Java, πολλά στοιχεία της οποίας έχουν ήδη αναφερθεί στη συζήτηση για τα χαρακτηριστικά της γλώσσας. Στο σχήμα αυτό, με γκρι χρώμα παριστάνονται τα τμήματα του συστήματος που σχετίζονται με την υλοποίηση της γλώσσας Java. Τα τμήματα αυτά μπορεί κανείς να τα προμηθευθεί στο εμπόριο ή στο Internet.

Στο πρώτο στάδιο γίνεται η μεταγλώττιση του πηγαίου κώδικα Java, που έχει γράψει ο προγραμματιστής. Για το σκοπό αυτό χρησιμοποιείται ο *μεταγλωττιστής* Java (Java compiler), που μεταφράζει τον πηγαίο κώδικα σε ενδιάμεσο κώδικα (bytecode). Ο ενδιάμεσος κώδικας αποθηκεύεται σε ένα ή περισσότερα αρχεία, τα οποία είναι δυνατό να είτε να χρησιμοποιηθούν τοπικά, είτε να μεταφερθούν σε άλλους υπολογιστές μέσω του δικτύου.

Στο δεύτερο στάδιο γίνεται η εκτέλεση του ενδιάμεσου κώδικα από *την εικονική μηχανή* της Java (Java Virtual Machine), που φαίνεται διαγραμμισμένη στο Σχήμα 1.1, μέσα σε πλαίσιο. Αν και ο διερμηνέας της Java δεν είναι παρά ένα μόνο τμήμα της εικονικής μηχανής, συχνά όλη η μηχανή ονομάζεται καταχρηστικά διερμηνέας Java. Το πρώτο τμήμα της εικονικής μηχανής είναι ο *φορτωτής κλάσεων* (class loader), που αναλαμβάνει το φόρτωμα του ενδιάμεσου κώδικα που υλοποιεί κάποια ζητούμενη κλαση. Στην περίπτωση που αυτή η κλάση δε βρίσκεται τοπικά, στην *βιβλιοθήκη κλάσεων* της Java, ο φορτωτής κλάσεων αναλαμβάνει τη μεταφορά της μέσω του δικτύου. Αμέσως μετά το φόρτωμα μιας κλάσης, ακολουθεί ο έλεγχος ορθότητας του κώδικά της, που πραγματοποιείται από τον *ελεγκτή ενδιάμεσου κώδικα* (bytecode verifier). Αυτός εξασφαλίζει ότι ο ενδιάμεσος κώδικας που φορτώθηκε δε θα θέσει σε κίνδυνο την ασφάλεια του υπολογιστικού συστήματος.



Σχήμα 1.1. Διαδικασία μεταγλώττισης και εκτέλεσης προγραμμάτων Java.

Στη συνέχεια, ακολουθεί η κυρίως φάση της εκτέλεσης του ενδιάμεσου κώδικα, που πραγματοποιείται εναλλακτικά από το *διερμηνέα* (interpreter) της Java, ή τον *μεταγλωττιστή τελευταίας στιγμής* (just-in-time compiler). Και τα δυο αυτά τμήματα επικοινωνούν με το *σύστημα χρόνου εκτέλεσης* (runtime system), που αναλαμβάνει την κατασκευή και την καταστροφή αντικειμένων, τη διαχείριση της μνήμης, και άλλες σημαντικές λειτουργίες. Το σύστημα χρόνου εκτέλεσης αλληλεπιδρά με το *λειτουργικό σύστημα* (operating system) του υπολογιστή, μέσω του οποίου αποκτά πρόσβαση στο *υλικό* (hardware) και επικοινωνεί με το χρήστη.

1.5. Είδη προγραμμάτων Java

Τα προγράμματα που αναπτύσονται σε Java μπορούν να εκτελεστούν είτε ως αυτόνομες εφαρμογές, είτε ως τμήματα σελίδων HTML, μέσα από έναν περιηγητή. Παρακάτω περιγράφουμε τα δύο αυτά είδη προγραμμάτων.

Αυτόνομες εφαρμογές μπορούν να εκτελεστούν απ' ευθείας από το διερμηνέα της γλώσσας. Αυτό φυσικά προϋποθέτει την ύπαρξη του διερμηνέα, αλλά διερμηνείς της Java έχουν αναπτυχθεί γιά πολλά περιβάλλοντα. Από προγραμματιστικής πλευράς ο προγραμματιστής αρκεί να ορίσει μια κλάση η οποία να περιέχει μια μέθοδο με όνομα `main`. Όταν ο διερμηνέας κληθεί με το όνομα αυτής της κλάσης, η εκτέλεση θα αρχίσει από αυτή τη μέθοδο.

Εφαρμογές που εκτελούνται μέσω ενός περιηγητή ονομάζονται applets. Ο περιηγητές πρέπει να είναι σε θέση να αναγνωρίσει και να εκτελέσει ενδιάμεσο κώδικα Java. Ενα applet είναι ένα αντικείμενο που προέρχεται από κάποια υποκλάση της `Applet`. Σελίδες HTML μπορούν να περιέχουν applets, που μπορούν να μεταφέρονται μέσω του δικτύου και να εκτελούνται. Οι σελίδες προσδιορίζουν τις συντεταγμένες της οθόνης όπου θα εμφανιστεί και θα εκτελεστεί το

applet. Ένα applet περιέχει ορισμένες μεθόδους που καλούνται όταν συμβούν κάποια γεγονότα, όπως π.χ. το πάτημα του πλήκτρου του ποντικιού πάνω στο applet.

1.6. Ένα απλό παράδειγμα κώδικα Java

Στη συνέχεια δίνεται ένα παράδειγμα προγράμματος Java. Το πρόγραμμα είναι ένα από τα απλούστερα που μπορούν να γραφούν και εμφανίζει στο χρήστη το μήνυμα “Hello world!”. Στις επόμενες δυο παραγράφους δίνεται και εξηγείται ο κώδικας του προγράμματος, κατ’ αρχήν ως ανεξάρτητη εφαρμογή και στη συνέχεια ως applet.

1.6.1. Αυτόνομη εφαρμογή Java

Ο κώδικας του απλού προγράμματος Java που θα αποτελέσει αυτόνομη εφαρμογή είναι ο ακόλουθος:

```
public class HelloWorld
{
   public static void main (String [] args)
   {
      System.out.println("Hello world!");
   }
}
```

Όπως κάθε πρόγραμμα Java, το παραπάνω αποτελείται από τη δήλωση μιας κλάσης. Το όνομα της κλάσης που ορίζεται εδώ είναι `HelloWorld`. Η κλάση περιέχει τον ορισμό μιας μεθόδου με όνομα `main`. Το σώμα της `main` περιέχει μια μόνο εντολή, που εκτυπώνει το μήνυμα “Hello World!”. Η σύνταξη του προγράμματος θυμίζει τη σύνταξη προγραμμάτων C και C++. Οι υπόλοιπες λεπτομέρειες του προγράμματος θα γίνουν κατανοητές στα επόμενα κεφάλαια.

Τα προγράμματα Java αποθηκεύονται σε αρχεία, των οποίων το όνομα τελειώνει σε `.java`. Ας υποθέσουμε ότι ο παραπάνω κώδικας είναι αποθηκευμένος σε ένα αρχείο με όνομα `hello.java`. Προκειμένου να εκτελεσθεί το πρόγραμμα, το επόμενο βήμα είναι η εκτέλεση της εντολής:

```
javac hello.java
```

με την οποία καλείται ο μεταγλωττιστής Java, το όνομα του οποίου είναι `javac`. Ο μεταγλωττιστής ελέγχει την ορθότητα του προγράμματος. Αν αυτό είναι σωστό, τότε δημιουργείται ένα αρχείο με όνομα `HelloWorld.class`, το οποίο περιέχει τον ενδιάμεσο εκτελέσιμο κώδικα της κλάσης `HelloWorld`. Ένα ενδιαφέρον σημείο είναι ότι το όνομα του παραγόμενου αρχείου προέρχεται από το όνομα της κλάσης που ορίζεται και όχι από το όνομα του αρχείου που περιέχει τον ορισμό της.

Το αρχείο `HelloWorld.class` περιέχει ενδιάμεσο κώδικα και χρειάζεται τον διερμηνέα της Java για να εκτελεστεί. Με την εντολή:

```
java HelloWorld
```

εκτελείται το αρχείο `HelloWorld.class` και το μήνυμα τυπώνεται στη οθόνη. Στην παραπάνω εντολή, `java` είναι το όνομα του διερμηνέα της Java. Ο διερμηνέας δέχεται ως παράμετρο το

όνομα μιας κλάσης και όχι το όνομα του αρχείου που την περιέχει. Με άλλα λόγια, ο διερμηνέας της Java εκτελεί κλάσεις ξεκινώντας πάντα από τη μέθοδο `main`. Το αρχείο `HelloWorld.class` μπορεί να μεταφερθεί και να εκτελεστεί σε διαφορετικό περιβάλλον υπολογιστή χωρίς να χρειαστεί άλλη επεξεργασία. Το μόνο που απαιτείται είναι η ύπαρξη ενός διερμηνέα της γλώσσας.

1.6.2. Java Applet

Ο παρακάτω κώδικας υλοποιεί το ίδιο πρόγραμμα και αποτελεί παράδειγμα ορισμού κλάσης που θα χρησιμοποιηθεί ως applet:

```
import java.awt.*;
import java.applet.*;

public class HelloWorld extends Applet
{
   public void paint (Graphics g)
   {
      g.drawstring("Hello World!");
   }
}
```

Η γενική δομή του προγράμματος είναι παρόμοια με αυτή του πρώτου παραδείγματος. Οι εντολές `import` κάνουν διαθέσιμους στο μεταφραστή διάφορους ορισμούς κλάσεων από τη βιβλιοθήκη κλάσεων της Java, όπως π.χ. η κλάση `Applet`. Η κλάση `HelloWorld` που ορίζεται στο παράδειγμα κληρονομεί την κλάση `Applet`, με χρήση της λέξης `extends`. Περιέχει μια μέθοδο με όνομα `paint`, που “ζωγραφίζει” το μήνυμα “Hello World!”. Το πρόγραμμα αυτό πρόκειται να εκτελεστεί μέσα από έναν περιηγητή, ο οποίος θα καλεί τη μέθοδο `paint` κάθε φορά που θα πρέπει να σχεδιάσει το applet στην οθόνη.

Η μετάφραση γίνεται όπως στο προηγούμενο παράδειγμα. Ας υποθέσουμε ότι ο παραπάνω κώδικας είναι αποθηκευμένος στο αρχείο `hello.java`. Με την εντολή:

```
javac hello.java
```

μεταφράζεται ο κώδικας που περιέχεται σε αυτό. Όμως, αυτή τη φορά, το αρχείο `HelloWorld.class` που προκύπτει δεν μπορεί να εκτελεστεί αυτόνομα, γιατί δεν περιέχει μέθοδο `main`. Προκειμένου να εκτελεσθεί, χρειάζεται να ενσωματωθεί σε μια σελίδα HTML. Ο παρακάτω κώδικας περιγράφει μια απλή σελίδα HTML που περιέχει το applet:

```
<HTML>

<HEAD>
<TITLE>My first Java applet</TITLE>
</HEAD>

<BODY>
<APPLET CODE=HelloWorld WIDTH=200 HEIGHT=100></APPLET>
</BODY>

</HTML>
```

Η εντολή `<APPLET>` προσδιορίζει το applet που περιέχεται, του οποίου το όνομα δίνεται στην παράμετρο `CODE`, καθώς και τις διαστάσεις που θα έχει στην σελίδα HTML. Όταν το αρχείο που

περιέχει τον παραπάνω κώδικα HTML διαβασθεί από έναν περιηγητή με δυνατότητα εκτέλεσης Java applets, όπως π.χ. ο Netscape Navigator, το μήνυμα “Hello world!” θα εμφανιστεί στη σελίδα HTML.

2. Η γλώσσα Java

Στο κεφάλαιο αυτό περιγράφεται ο σκελετός της γλώσσας προγραμματισμού Java και ορισμένα από τα βασικά της χαρακτηριστικά. Τα περισσότερο προχωρημένα χαρακτηριστικά της γλώσσας περιγράφονται αναλυτικά σε επόμενα κεφάλαια.

2.1. Κώδικας προγραμμάτων

Η γλώσσα Java προέρχεται από την κοινότητα του Internet. Προκειμένου να καλύψει τις ανάγκες αυτής της κοινότητας, η οποία απαρτίζεται από άτομα διαφορετικών εθνοτήτων που μιλούν διαφορετικές γλώσσες, η Java πρέπει να είναι σε θέση να χειρίζεται έναν μεγάλο αριθμό γλωσσών. Για το λόγο αυτό, η Java υποστηρίζει το διεθνές σύστημα κωδικοποίησης χαρακτήρων Unicode (βλ. http://www.unicode.org/). Τα αρχεία προγραμμάτων Java μπορούν να είναι γραμμένα σε Unicode. Αυτό φυσικά δεν αποκλείει τη χρήση απλών χαρακτήρων ASCII, που αποτελούν ούτως ή άλλως τμήμα του συστήματος Unicode.

2.2. Σχόλια

Η Java υποστηρίζει δυο ειδών σχόλια. Το πρώτο είδος είναι αυτό της γλώσσας C. Τα σχόλια αυτά αποτελούνται από μια ή περισσότερες γραμμές κειμένου και περικλείονται από τις ακολουθίες χαρακτήρων `/*` και `*/` όπως στο ακόλουθο παράδειγμα:

```
/* This is a first comment */
/* This is a second comment
     that is written
       in three lines */
```

Τα σχόλια αυτού του είδους δεν μπορούν να είναι φωλιασμένα, δηλαδή δεν επιτρέπονται σχόλια μέσα σε σχόλια.

Το δεύτερο είδος είναι αυτό των σχολίων μιας γραμμής της γλώσσας C++. Τα σχόλια αυτά αρχίζουν με την ακολουθία χαρακτήρων `//` και εκτείνονται ως το τέλος της γραμμής. Παραδείγματα τέτοιων σχολίων είναι τα:

```
// This is a first single line comment
// And this is a // second one //
```

Κατά σύμβαση, ένα σχόλιο του πρώτου είδους που αρχίζει με την ακολουθία χαρακτήρων `/**` επισημαίνει την ύπαρξη πληροφοριών τεκμηρίωσης. Κατάλληλα προγράμματα που ονομάζονται *γεννήτορες τεκμηρίωσης*, όπως το *javadoc* της Sun, επεξεργάζονται τέτοια σχόλια και κατασκευάζουν αυτόματα την τεκμηρίωση του κώδικα. Στο εσωτερικό τέτοιου είδους σχολίων, τυχόν κενά στην αρχή των γραμμών ακολουθούμενα από το χαρακτήρα `*` αγνοούνται από το γεννήτορα. Επίσης, γραμμές που αρχίζουν με το χαρακτήρα `@` ερμηνεύονται ως ειδικές οδηγίες προς το γεννήτορα. Παράδειγμα σχολίου τεκμηρίωσης είναι αυτό που ακολουθεί:

```
/**
 * This class represents a client of the system.
 *
 * @author Nikolaos Papaspyrou
 * @version 1.0 alpha, January 6, 1996
 * @see Server
 */
```

Ανάλογα με το γεννήτορα που χρησιμοποιείται, τα σχόλια τεκμηρίωσης μπορούν επίσης να περιέχουν οδηγίες σε κατάλληλη γλώσσα μορφοποίησης (π.χ. HTML). Οι οδηγίες αυτές ενσωματώνονται στην τεκμηρίωση από το γεννήτορα.

2.3. Τύποι δεδομένων και σταθερές

Το σύστημα τύπων μιας γλώσσας προγραμματισμού αποσκοπεί στην πρόληψη σφαλμάτων που μπορούν να παρουσιαστούν κατά την εκτέλεση των προγραμμάτων. Η βασική λειτουργία του συστήματος τύπων είναι η συσχέτιση τύπων με τις μεταβλητές ενός προγράμματος και με τις εκφράσεις που συναντώνται σε αυτό.

Το σύστημα τύπων της Java συνδυάζει χαρακτηριστικά που συναντούνται σε γλώσσες με στατικά αλλά και με δυναμικά συστήματα τύπων. Είναι *στατικό* με την έννοια ότι κάθε μεταβλητή συσχετίζεται με ένα συγκεκριμένο και μη τετριμμένο τύπο, ο οποίος είναι γνωστός κατά τη μεταγλώττιση. Είναι *δυναμικό* με την έννοια ότι κατά το χρόνο εκτέλεσης διατηρούνται πληροφορίες σχετικές με τον τύπο των αντικειμένων, ώστε να υποστηρίζεται ο πολυμορφισμός με ασφαλή τρόπο.

Οι τύποι στη Java χωρίζονται σε δυο μεγάλες κατηγορίες: τους *βασικούς τύπους* και τους *τύπους αναφοράς*. Στην ενότητα αυτή θα γίνει μια εισαγωγή επίσης στον *τύπο συμβολοσειράς* και στους *τύπους πινάκων* που, αν και στην πραγματικότητα είναι τύποι αντικειμένων, χαίρουν ιδιαίτερης μεταχείρισης από τους μεταγλωττιστες της Java.

2.3.1. Βασικοί τύποι

Οι βασικοί τύποι της Java χρησιμοποιούνται για την αναπαράσταση θεμελιωδών τύπων δεδομένων, όπως είναι οι αριθμοί, οι χαρακτήρες και οι λογικές τιμές. Συνοπτικά δίνονται στον Πίνακα 2.1. Στον ίδιο πίνακα περιγράφονται επίσης οι δυνατές τιμές που μπορούν να πάρουν τα αντίστοιχα δεδομένα. Αξίζει να σημειωθεί ότι όλοι οι τύποι ακεραίων αφορούν προσημασμένους αριθμούς σε αναπαράσταση συμπληρώματος ως προς 2. Επίσης, οι αριθμοί κινητής υποδιαστολής ακολουθούν το πρότυπο IEEE 754.

Πίνακας 2.1. Βασικοί τύποι της Java.

Τύπος	***Περιγραφή***	***Αρχική τιμή***
`boolean`	Λογικές τιμές: true ή false	`false`
`char`	Χαρακτήρες, 16-bit Unicode	`'\0'`
`byte`	Ακέραιος αριθμός 8-bit	`0`
`short`	Ακέραιος αριθμός 16-bit	`0`
`int`	Ακέραιος αριθμός 32-bit	`0`
`long`	Ακέραιος αριθμός 64-bit	`0`
`float`	Αριθμός κινητής υποδιαστολής 32-bit	`0.0`
`double`	Αριθμός κινητής υποδιαστολής 64-bit	`0.0`

Μεταβλητές μπορούν να ορισθούν στο εσωτερικό μεθόδων ή κλάσεων (που θα περιγραφούν στο επόμενο κεφάλαιο) με τρόπο παρόμοιο όπως στη γλώσσα C. Για παράδειγμα, ο κώδικας:

```
int x;
double d1, d2;
```

δηλώνει τρεις μεταβλητές με ονόματα `x`, `d1` και `d2`. Η πρώτη έχει τύπο `int`, ενώ οι δυο επόμενες έχουν τύπο `double`. Επίσης, οι μεταβλητές μπορούν να αρχικοποιηθούν κατά τη δήλωσή τους, όπως για παράδειγμα στον παρακάτω κώδικα:

```
int x = 1;
double d1 = 3.14, d2 = -8.0;
```

Αν μια μεταβλητή δηλωθεί χωρίς να αρχικοποιηθεί, τότε η αρχικοποίηση γίνεται συχνά με αυτόματο τρόπο.[1] Οι τιμές που χρησιμοποιούνται για τις αυτόματες αρχικοποιήσεις δίνονται στην τελευταία στήλη του Πίνακα 2.1.

Οι μοναδικές τιμές του τύπου `boolean` είναι οι σταθερές `true` και `false`. Οι σταθερές τύπου χαρακτήρα είτε περιέχονται μεταξύ απλών εισαγωγικών, είτε δίνονται βάσει του κωδικού Unicode σε δεκαεξαδική μορφή. Με τη χρήση του ειδικού χαρακτήρα `\` (ανάστροφη κάθετος, backslash) προκύπτουν οι ειδικές *ακολουθίες διαφυγής* (escape sequences), που έχουν θέση ενός χαρακτήρα όταν βρίσκονται μεταξύ εισαγωγικών. Μερικές χρήσιμες ακολουθίες διαφυγής δίνονται στον Πίνακα 2.2. Παραδείγματα σταθερών τύπου χαρακτήρα είναι τα ακόλουθα:

```
'a'
'\n'
\u00ff
```

[1] Η αυτόματη αρχικοποίηση δε συμβαίνει μόνο στην (πολύ συνήθη) περίπτωση των μεταβλητών που είναι τοπικές σε μια μέθοδο. Στην περίπτωση αυτή, η αρχικοποίηση πρέπει να είναι ρητή, είτε μαζί με τη δήλωση είτε σε επόμενη εντολή. Αν δεν προηγηθεί αρχικοποίηση, κάθε χρήση της μεταβλητής θεωρείται σφάλμα κατά τη μεταγλώττιση.

Πίνακας 2.2. Μερικές ακολουθίες διαφυγής.

Ακολουθία διαφυγής	***Ερμηνεία***
`\n`	Ο χαρακτήρας τερματισμού γραμμής (LF)
`\r`	Ο χαρακτήρας επιστροφής (CR)
`\0`	Ο κενός (μηδενικός) χαρακτήρας
`\'`	Ο χαρακτήρας ' (απλό εισαγωγικό)
`\"`	Ο χαρακτήρας " (διπλό εισαγωγικό)
`\unnnn`	Ο χαρακτήρας με κωδικό Unicode *nnnn* (σε δεκαεξαδικό)

Οι ακέραιες σταθερές μπορούν να γραφούν στο δεκαδικό σύστημα, το οκταδικό ή το δεκαεξαδικό. Οι δεκαδικές σταθερές αποτελούνται από μια ακολουθία δεκαδικών ψηφίων, από τα οποία το πρώτο δεν είναι το `0`. Για παράδειγμα:

```
9327
```

Οι οκταδικές σταθερές αρχίζουν με το ψηφίο `0`, του οποίου έπεται μια ακολουθία οκταδικών ψηφίων. Για παράδειγμα:

```
03477          1855 στο δεκαδικό
```

Οι δεκαεξαδικές σταθερές αρχίζουν με την ακολουθία `0x`, της οποίας έπεται μια ακολουθία δεκαεξαδικών ψηφίων. Για παράδειγμα:

```
0xFF77         65399 στο δεκαδικό
```

Όλες οι ακέραιες σταθερές είναι τύπου `int`, εκτός αν ακολουθούνται από το γράμμα L. Στην περίπτωση αυτή είναι τύπου `long`. Για παράδειγμα:

```
563L
```

Τέλος, οι σταθερές κινητής υποδιαστολής μπορούν να γραφούν σε δεκαδική μορφή, με ή χωρίς εκθέτη. Είναι τύπου `double`, εκτός αν ακολουθούνται από το γράμμα `F`. Στην περίπτωση αυτή είναι τύπου `float`. Παραδείγματα:

```
3.14159        τύπου double
3.14159F       τύπου float
9.05e8         τύπου double, ίση με: 9.05 × 10^8
42e-14F        τύπου float, ίση με: 42 × 10^-14
```

2.3.2. Τύποι αναφοράς

Όλοι οι τύποι που δεν είναι βασικοί ανήκουν στην κατηγορία των τύπων αναφοράς. Στην κατηγορία αυτή ανήκουν όλοι οι τύποι αντικειμένων και, κατά συνέπεια, όλοι οι τύποι δεδομένων που μπορεί να ορίσει ο προγραμματιστής. Οι τιμές που μπορεί να πάρει ένα δεδομένο που ανήκει σε ένα τύπο αναφοράς είναι στην ουσία *αναφορές* σε αντικείμενα: όχι τα ίδια τα αντικείμενα αλλά δείκτες σε αυτά. Μπορούν να παραλληλισθούν με τους δείκτες στη C, με τη

διαφορά ότι δεν απαιτείται ειδική μεταχείριση προκειμένου να προσπελάσει κανείς τα αντικείμενα στα οποία δείχνουν. Η Java δεν επιτρέπει την κατασκευή αναφορών παρά μόνο με την εκχώρηση ή το πέρασμα αντικειμένων.

Το όνομα “τύποι αναφοράς” έχει προέρθει από το διαφορετικό τρόπο με τον οποίο τα δεδομένα που ανήκουν σε αυτούς τους τύπους διακινούνται κατά την εκτέλεση των προγραμμάτων.[2] Όταν ένα δεδομένο που ανήκει σε κάποιο βασικό τύπο πρόκειται να εκχωρηθεί σε μια μεταβλητή ή να περαστεί ως όρισμα σε μια μέθοδο, η τιμή του απλώς αντιγράφεται. Αντίθετα, όταν ένα δεδομένο που ανήκει σε κάποιο τύπο αναφοράς πρόκειται να εκχωρηθεί σε μια μεταβλητή ή να περαστεί ως όρισμα σε μια μέθοδο, αυτό που αντιγράφεται είναι η αναφορά στο αντικείμενο, όχι το ίδιο το αντικείμενο. Με αυτό τον τρόπο, δημιουργούνται περισσότερες αναφορές στο ίδιο αντικείμενο, το οποίο μπορεί να προσπελασθεί μέσω οποιασδήποτε από αυτές.

Η ειδική τιμή `null` υποδηλώνει ότι ένα δεδομένο που ανήκει σε κάποιο τύπο αναφοράς δεν αναφέρεται σε κανένα αντικείμενο. Αυτή είναι και η τιμή που χρησιμοποιείται για την αυτόματη αρχικοποίηση δεδομένων που ανήκουν σε τύπους αναφοράς. Πριν χρησιμοποιηθούν αυτά τα δεδομένα για οτιδήποτε μη τετριμμένο, είναι απαραίτητο να ληφθεί μέριμνα ώστε να αναφέρονται σε υπαρκτά αντικείμενα. Εκτενέστερη συζήτηση θα γίνει στο επόμενο κεφάλαιο.

2.3.3. Ειδικοί τύποι

Οι τύποι που θα περιγραφούν στην ενότητα αυτή ανήκουν στην κατηγορία των τύπων αναφοράς. Πρόκειται στην πραγματικότητα για τύπους που έχουν ιδιαίτερη σημασία για τη γλώσσα Java και για τους οποίους χρησιμοποιείται ιδιαίτερη σύνταξη. Πάντως, αν παραβλέψει κανείς την ιδιαίτερη σύνταξη, οι τύποι αυτοί δεν είναι παρά απλοί τύποι αντικειμένων, όπως εκείνοι που θα περιγραφούν στο επόμενο κεφάλαιο.

Ο τύπος συμβολοσειράς ονομάζεται `String`. Οι συμβολοσειρές στη Java είναι ακολουθίες χαρακτήρων Unicode. Σταθερές τύπου συμβολοσειράς μπορούν να γραφούν ανάμεσα σε διπλά εισαγωγικά. Ο μεταγλωττιστής της Java τις μετατρέπει αυτόματα σε κατάλληλα αντικείμενα τύπου `String`. Εκτός από κοινούς χαρακτήρες, οι σταθερές συμβολοσειρές μπορούν να περιέχουν ακολουθίες διαφυγής. Παραδείγματα:

```
"Java"
"Hello world!\n"
"So long \"Java\" world...\n"
```

Οι τύποι πινάκων ανήκουν επίσης στην κατηγορία των ειδικών τύπων. Πρόκειται για τύπους αντικειμένων με ειδική σύνταξη που διευκολύνει τη δήλωση και τη χρήση των πινάκων. Ένας τύπος πίνακα προκύπτει από κάποιο έγκυρο τύπο, με την προσθήκη δυο κενών αγκυλών `[]`. Οι αγκύλες μπορούν να τοποθετηθούν εναλλακτικά είτε μετά το όνομα του τύπου, ή μετά το όνομα της μεταβλητής που ορίζεται. Παραδείγματα δήλωσης πινάκων είναι τα ακόλουθα:

```
int myArray [];
int [] yourArray;
String hisArray [];
```

[2] Τα δεδομένα που ανήκουν στους βασικούς τύπους περνώνται *κατ’ αξία* (by value), ενώ τα αντικείμενα που ανήκουν στους τύπους αναφοράς περνώνται *κατ’ αναφορά* (by reference).

Πολυδιάστατοι πίνακες μπορούν να δηλωθούν με παρόμοιο τρόπο:

```
double [][] myMatrix2D;
int yourMatrix3D [][][];
```

Περισσότερα για την αρχικοποίηση και τη χρήση των πινάκων γράφονται στο επόμενο κεφάλαιο.

Στο σημείο αυτό αξίζει να παρατηρηθεί ότι οι μεταβλητές των ειδικών τύπων που περιγράφηκαν στην ενότητα αυτή πρέπει να αρχικοποιούνται πριν χρησιμοποιηθούν. Σε αντίθετη περίπτωση, όπως όλες οι μη αρχικοποιημένες μεταβλητές τύπων αναφοράς, έχουν την (άχρηστη) τιμή `null`. Επίσης, αξίζει να σημειωθεί ότι οι διαστάσεις των πινάκων δεν καθορίζονται κατά τη δήλωσή τους.

2.4. Εκφράσεις

Οι εκφράσεις στη Java είναι τμήματα του κώδικα που, όταν εκτελούνται, παράγουν κάποιο αποτέλεσμα. Το αποτέλεσμα που προκύπτει από την εκτέλεση μιας έκφρασης ονομάζεται *τιμή της έκφρασης*, και για το λόγο αυτό η εκτέλεση μιας έκφρασης ονομάζεται συχνά και *αποτίμηση της έκφρασης*.

Η τιμή μιας έκφρασης στη Java μπορεί να ανήκει σε κάποιον από τους βασικούς τύπους, π.χ. να είναι αριθμητική, να ανήκει σε κάποιο τύπο αναφοράς, ή να ανήκει στον ειδικό τύπο `void`. Ο τύπος κάθε έκφρασης είναι πάντα γνωστός κατά τη μεταγλώττιση του κώδικα και η τιμή που θα προκύψει κατά την αποτίμηση στο χρόνο εκτέλεσης θα ανήκει υποχρεωτικά στον ίδιο τύπο.[3] Ο τύπος `void` είναι κενός, δεν περιέχει δηλαδή καμιά τιμή. Η αποτίμηση μιας έκφρασης με τύπο `void` γίνεται μόνο για τις παρενέργειες που μπορεί να έχει. Το αποτέλεσμα δεν μπορεί να χρησιμοποιηθεί γιατί δεν υπάρχει.

Η σύνταξη των εκφράσεων της Java μοιάζει πολύ με αυτή της γλώσσας C. Ιδιαίτερη έμφαση έχει δοθεί στην απλότητα και τη συνέπεια των εκφράσεων. Οι περισσότεροι τελεστές της γλώσσας C υποστηρίζονται, με την ίδια σημασία και την ίδια σειρά *προτεραιότητας*, όπως φαίνεται στον Πίνακα 2.3. Στον πίνακα αυτό, μεγαλύτεροι αριθμοί προτεραιότητας συμβολίζουν στενότερο "δέσιμο" ενός τελεστή με τα τελούμενά του. Για παράδειγμα, η έκφραση:

```
c = a + b * 2
```

ερμηνεύεται ως:

```
c = (a + (b * 2))
```

γιατί ο τελεστής `+` έχει μεγαλύτερη προτεραιότητα από τον τελεστή `=` (11 έναντι 1) και ο τελεστής `*` έχει μεγαλύτερη προτεραιότητα από τον τελεστή `+` (12 έναντι 11).

Στην περίπτωση τελεστών με δυο τελούμενα, ο Πίνακας 2.3 δίνει εκτός από την προτεραιότητα και την *προσεταιριστικότητα* του τελεστή. Η προσεταιριστικότητα ενός τελεστή μπορεί να είναι

[3] Μοναδική εξαίρεση οι τύποι αναφοράς. Στην περίπτωση αυτή, η τιμή που θα προκύψει μπορεί να ανήκει σε κάποιο τύπο που είναι συμβατός για ανάθεση με τον πρώτο. Περισσότερη συζήτηση για αυτό το θέμα στο κεφάλαιο 3.

από αριστερά προς τα δεξιά ή από τα δεξιά προς τα αριστερά, και καθορίζεται από το βέλος που φαίνεται δίπλα στον αριθμό προτεραιότητας. Υποδεικνύει τον τρόπο “δεσίματος” των ορισμάτων στην περίπτωση τελεστών με την ίδια προτεραιότητα. Για παράδειγμα, η έκφραση:

```
a + b - 3
```

ερμηνεύεται ως:

```
(a + b) - 3
```

γιατί οι τελεστές + και -, παρότι έχουν την ίδια προτεραιότητα, έχουν προσεταιριστικότητα από αριστερά προς τα δεξιά. Αντίθετα η έκφραση:

```
a += b = 0
```

ερμηνεύεται ως:

```
a += (b = 0)
```

Η σειρά αποτίμησης των υπο-εκφράσεων μιας σύνθετης έκφρασης, όταν δεν υπάρχουν άλλοι περιορισμοί που να υπαγορεύονται από την προτεραιότητα και την προσεταιριστικότητα των τελεστών, είναι από αριστερά προς τα δεξιά.

Πίνακας 2.3. Προτεραιότητα τελεστών στη Java.

Προτεραιότητα	***Τελεστής***	***Περιγραφή***
13	`++ --`	Αύξηση / μείωση κατά 1
	`+ -`	Πρόσημο συν / πλήν
	`~`	Συμπλήρωμα bit προς bit
	`!`	Λογική άρνηση
	`( type )`	Μετατροπή τύπου
12 →	`* / %`	Πολλαπλασιασμός, διαίρεση, υπόλοιπο
11 →	`+ -`	Πρόσθεση, αφαίρεση
	`+`	Παράθεση συμβολοσειρών
10 →	`<<`	Αριστερή ολίσθηση
	`>>`	Δεξιά ολίσθηση με επέκταση προσήμου
	`>>>`	Δεξιά ολίσθηση χωρίς επέκταση προσήμου
9 →	`< > =< >=`	Αριθμητική σύγκριση
	`instanceof`	Σύκριση τύπου
8 →	`== !=`	Ισότητα, ανισότητα
7 →	`&`	Σύζευξη, λογική ή bit προς bit
6 →	`^`	Αποκλειστική διάζευξη, λογική ή bit προς bit
5 →	`|`	Διάζευξη, λογική ή bit προς bit
4 →	`&&`	Λογική σύζευξη υπό συνθήκη

Προτεραιότητα	*Τελεστής*	*Περιγραφή*
3 →	`||`	Λογική διάζευξη υπό συνθήκη
2 ←	`?:`	Εναλλακτική αποτίμηση υπό συνθήκη
1 ←	`=`	Εκχώρηση
	`*= /= %= += -= <<= >>= >>>= &= ^= |=`	Εκχώρηση με πράξη

Από τον Πίνακα 2.3 λείπουν ο τελεστής κατασκευής αντικειμένων `new` και ο τελεστής πρόσβασης μεταβλητών και κλήσης μεθόδων `.` (τελεία). Και οι δυο, καθώς και οι υπόλοιποι τελεστές που δέχονται τελούμενα τύπων αναφοράς, θα συζητηθούν στο επόμενο κεφάλαιο.

2.4.1. Αριθμητικοί τελεστές και τελεστές σύγκρισης

Οι αριθμητικοί τελεστές `+`, `-`, `*`, `/` και `%` χρησιμοποιούνται για τις συνήθεις αριθμητικές πράξεις. Τα τελούμενα πρέπει να ανήκουν σε κάποιο αριθμητικό τύπο (ακέραιο ή κινητής υποδιαστολής). Το ίδιο συμβαίνει και με το αποτέλεσμα.

Ο τελεστής `+` χρησιμοποιείται, εκτός από την πρόσθεση αριθμητικών τιμών, και για την παράθεση (συνένωση) συμβολοσειρών. Για παράδειγμα, το αποτέλεσμα από την αποτίμηση της έκφρασης:

```
"Ja" + "va"
```

είναι η συμβολοσειρά:

```
"Java"
```

Οι τελεστές `++` και `--` υλοποιούν αντίστοιχα την αύξηση και τη μείωση κατά ένα μεταβλητών αριθμητικού τύπου. Κάθε ένας από αυτούς μπορεί να τοποθετηθεί είτε πρίν, είτε μετά το όνομα της μεταβλητής. Η λειτουργία είναι λίγο διαφορετική στις δυο αυτές περιπτώσεις. Αν ο τελεστής τοποθετηθεί μετά τη μεταβλητή, η τιμή της έκφρασης είναι η αρχική τιμή της μεταβλητής, πριν την αύξηση ή τη μείωση. Αν τοποθετηθεί πριν, η τιμή της έκφρασης είναι η τελική τιμή της μεταβλητής, μετά την αύξηση ή τη μείωση. Και στις δυο περιπτώσεις, πάντως, η αύξηση ή η μείωση πραγματοποιούνται. Για παράδειγμα, αν η μεταβλητή `a` περιέχει την τιμή `42`, τότε η τιμή της έκφρασης `a++` είναι `42` ενώ η τιμή της έκφρασης `++a` είναι `42`. Και στις δυο περιπτώσεις, μετά την αποτίμηση της έκφρασης, η μεταβλητή `a` θα περιέχει την τιμή `43`.

Οι τελεστές σύγκρισης ισότητας `==` και ανισότητας `!=` χρησιμοποιούνται τόσο για τελούμενα που ανήκουν σε βασικούς τύπους, όσο και για τελούμενα που ανήκουν σε τύπους αναφοράς. Στην πρώτη περίπτωση, οι τελεστές αυτοί εξετάζουν την ισότητα (ανισότητα) των τιμών των τελουμένων. Στη δεύτερη περίπτωση, εξετάζουν το αν τα δυο τελούμενα αναφέρονται στο ίδιο αντικείμενο. Οι υπόλοιποι τελεστές σύγκρισης `<`, `>`, `<=` και `>=` χρησιμοποιούνται μόνο για αριθμητικά τελούμενα.

2.4.2. Τελεστές bit προς bit

Οι τελεστές αυτοί χειρίζονται τελούμενα που ανήκουν σε αριθμητικούς τύπους όχι ως αριθμητικές τιμές αλλά ως ακολουθίες από bit. Για παράδειγμα, αν έχει προηγηθεί η δήλωση:

```
short a = 42;
```

τότε η τιμή της μεταβλητής `a` παριστάνεται από την παρακάτω ακολουθία bit, στο δυαδικό σύστημα:

```
0000000000101010
```

Το πλήθος των bit είναι 16 γιατί πρόκειται για ακέραιο αριθμό τύπου `short`.

Οι τελεστές `~`, `&`, `|` και `^` υλοποιούν αντίστοιχα το συμπλήρωμα, τη σύζευξη, τη διάζευξη και την αποκλειστική διάζευξη bit προς bit, όταν τα τελούμενα είναι αριθμητικά. Για παράδειγμα, η τιμή της έκφρασης:

```
~a
```

με βάση την παραπάνω δήλωση είναι `-43`, που σε παράσταση συμπληρώματος ως προς 2 αντιστοιχεί στην παρακάτω ακολουθία:

```
1111111111010101
```

Ας σημειωθεί ότι η τιμή κάθε bit της ακολουθίας έχει αντιστραφεί, δηλαδή έχει αντικατασταθεί από το συμπλήρωμά της.

Οι τελεστές `<<`, `>>` και `>>>` υλοποιούν την ολίσθηση ακολουθιών bit. Το πρώτο τελούμενο παρέχει την ακολουθία bit ενώ το δεύτερο τελούμενο τον αριθμό των ολισθήσεων. Για παράδειγμα, το αποτέλεσμα της έκφρασης:

```
a << 3
```

με βάση την παραπάνω δήλωση είναι 336, που αντιστοιχεί στην παρακάτω ακολουθία:

```
0000000101010000
```

Ας σημειωθεί ότι η ακολουθία αυτή προήρθε από την αρχική με ολίσθηση τριων θέσεων προς τα αριστερά.

2.4.3. Λογικοί τελεστές

Οι λογικοί τελεστές χρησιμοποιούνται για τελούμενα τύπου `boolean`. Οι κυριότεροι από αυτούς είναι οι τελεστές `!`, `&` και `|` που υλοποιούν αντίστοιχα τη λογική άρνηση, τη λογική σύζευξη και τη λογική διάζευξη. Για παράδειγμα, η τιμή της έκφρασης:

```
true & false
```

είναι `false`. Οι τελεστές `&&` και `||` υλοποιούν επίσης τη λογική σύζευξη και τη λογική διάζευξη, με τη διαφορά ότι η αποτίμηση των ορισμάτων διακόπτεται μόλις γίνει γνωστό το αποτέλεσμα. Με άλλα λόγια, ο τελεστής `&&` αποτιμά το δεύτερο όρισμα μόνο αν το πρώτο όρισμα έχει τιμή `true` (ειδάλλως το αποτέλεσμα θα είναι `false`) και ο τελεστής `||` αποτιμά το δεύτερο όρισμα μόνο αν το πρώτο όρισμα έχει τιμή `false` (ειδάλλως το αποτέλεσμα θα είναι `true`). Αυτό είναι σημαντικό στην περίπτωση που η αποτίμηση του δεύτερου ορίσματος εμπεριέχει παρενέργειες. Για παράδειγμα, η αποτίμηση των εκφράσεων:

```
false &  (b = true)
false && (b = true)
```

οδηγεί και στις δυο περιπτώσεις στην τιμή `false`. Στην πρώτη περίπτωση, όμως, γίνεται η εκχώρηση της τιμής `true` στη μεταβλητή `b`, ενώ στη δεύτερη δε γίνεται, γιατί το δεύτερο όρισμα δε χρειάζεται να αποτιμηθεί.

Ο πιο πολύπλοκος λογικός τελεστής είναι αυτός της επιλογής υπό συνθήκη. Ο τελεστής αυτός δέχεται τρία ορίσματα, που διαχωρίζονται με τους χαρακτήρες `?` (ερωτηματικό) και `:` (άνω και κάτω τελεία). Το πρώτο από αυτά πρέπει να είναι τύπου boolean, ενώ τα άλλα δυο πρέπει να ανήκουν στον ίδιο τύπο. Αφού αποτιμηθεί το πρώτο όρισμα, επιλέγεται ένα από τα άλλα δυο με τον εξής τρόπο: αν η τιμή του πρώτου ορίσματος είναι true επιλέγεται το δεύτερο όρισμα, διαφορετικά επιλέγεται το τρίτο όρισμα. Στη συνέχεια, η τιμή της συνολικής έκφρασης είναι η τιμή του ορίσματος που επιλέχθηκε. Το όρισμα που δεν επιλέγεται δεν αποτιμάται καθόλου. Για παράδειγμα, η έκφραση:

```
(1 + 1 == 2) ? 42 : a++
```

έχει τιμή `42`, γιατί το πρώτο όρισμα έχει τιμή true, πράγμα που οδηγεί στην επιλογή του δεύτερου ορίσματος. Η τιμή της μεταβλητής `a` δεν αυξάνεται γιατί το τρίτο όρισμα δεν αποτιμάται.

2.4.4. Τελεστές εκχώρησης

Με τον όρο *εκχώρηση* εννοούμε την ανάθεση μιας τιμής σε μια μεταβλητή. Ο τελεστής απλής εκχώρησης `=` χρησιμοποιείται για να αναθέσει την τιμή του δεύτερου ορίσματός του στη μεταβλητή που δηλώνεται από το πρώτο όρισμα. Για παράδειγμα, αν υποτεθεί ότι η μεταβλητή `x` έχει δηλωθεί με τύπο `int` και περιέχει την τιμή `5`, τότε, μετά την αποτίμηση της έκφρασης:

```
x = 42
```

η τιμή της μεταβλητής `x` θα είναι `42`.

Οι τελεστές εκχώρησης με πράξη χρησιμοποιούν για τον υπολογισμό της τιμής που θα ανατεθεί, εκτός από το δεύτερο όρισμα, και την τρέχουσα τιμή της μεταβλητής. Για παράδειγμα, η έκφραση:

```
x += 5
```

είναι ισοδύναμη με την έκφραση:

```
x = x + 5
```

με τη διαφορά ότι το όρισμα `x` αποτιμάται μόνο μια φορά.

2.5. Εντολές

Οι εντολές στη γλώσσα Java, όπως και στη C ή τη C++, εμφανίζονται μέσα σε τμήματα κώδικα που περικλείονται από άγκιστρα `{}`. Τέτοια τμήματα κώδικα επιτρέπονται μόνο στο κύριο σώμα μεθόδων, όπως θα περιγραφεί αναλυτικά στο επόμενο κεφάλαιο. Τα τμήματα κώδικα μπορούν να περιέχουν δηλώσεις μεταβλητών. Στην περίπτωση αυτή, οι μεταβλητές είναι ορατές μόνο στο εσωτερικό του τμήματος: δεν είναι δυνατό να χρησιμοποιηθούν έξω από αυτό. Επίσης, τα τμήματα κώδικα είναι δυνατό να περιέχουν φωλιασμένα άλλα τμήματα κώδικα. Παράδειγμα τμήματος κώδικα είναι το επόμενο, που εμπεριέχει τη δήλωση δυο μεταβλητών `x` και `d`, καθώς και ένα άλλο τμήμα κώδικα, στο πρώτο σκέλος της εντολής `if`.

```
{
   int x = 1;
   double d = 3.14159;

   if (x <= 5) {
      System.out.println(x);
      return d + 2.5;
   }
   else
      return 8.1 - d;
}
```

Η σύνταξη των εντολών στη Java δε διαφέρει πολύ από αυτή στη γλώσσα C. Στις επόμενες ενότητες περιγράφονται οι εντολές της Java, με τη βοήθεια παραδειγμάτων.

2.5.1. Εντολές εκφράσεων

Κάθε έκφραση μπορεί να θεωρηθεί ως μια εντολή που, όταν εκτελείται, παράγει κάποιο αποτέλεσμα. Με την προσθήκη του χαρακτήρα `;` (semicolon) στο τέλος μιας έκφρασης, αυτή μετατρέπεται σε απλή εντολή. Όταν αυτή η εντολή εκτελεσθεί, θα γίνει η αποτίμησή της και το αποτέλεσμα θα αγνοηθεί. Οι εντολές εκφράσεων χρησιμοποιούνται κυρίως για τις παρενέργειες που προκαλούν, π.χ. για την αλλαγή τιμών σε μεταβλητές του προγράμματος. Παραδείγματα εντολών εκφράσεων είναι τα ακόλουθα:

```
x = 1;
a = flag ? 5 + a : 8 + x;
```

Ειδική περίπτωση εκφράσεων είναι η κλήση μεθόδων, που θα περιγραφεί στο επόμενο κεφάλαιο. Όμως, αν και πρωθύστερη, κρίνεται σκόπιμη μια μικρή εισαγωγή στη μέθοδο `System.out.println`, που χρησιμοποιείται για την εκτύπωση τιμών στην οθόνη. Η μέθοδος αυτή δέχεται ως παράμετρο μια τιμή που μπορεί να ανήκει σε μια ποικιλία τύπων, συμπεριλαμβανομένων όλων των βασικών τύπων. Παράδειγμα χρήσης αυτής της μεθόδου σε μια εντολή έκφρασης είναι το ακόλουθο:

```
System.out.println("Hello world!");
```

που εκτυπώνει τη συμβολοσειρά “Hello world!”. Η μέθοδος αυτή αποτελεί τμήμα του συστήματος εισόδου-εξόδου της Java, που θα περιγραφεί με λεπτομέρειες στο κεφάλαιο 6.

2.5.2. Εντολές επιλογής

Η συχνότερα χρησιμοποιούμενη εντολή επιλογής είναι η εντολή `if`. Η εντολή αυτή αποτιμά μια λογική συνθήκη και, ανάλογα με την τιμή της, εκτελεί ένα από τα δυο πιθανά σκέλη της. Η σύνταξή της είναι:

```
if ( συνθήκη )
   εντολή
else
   εντολή
```

Στην περίπτωση που η συνθήκη είναι μια έκφραση τύπου `boolean`, αν αυτή έχει τιμή `true` εκτελείται η πρώτη εντολή, διαφορετικά η δεύτερη εντολή. Αν η συνθήκη είναι μια έκφραση διαφορετικού βασικού τύπου, τότε θεωρείται αληθής αν είναι μη μηδενική, διαφορετικά ψευδής. Επίσης, αν είναι μια έκφραση τύπου αναφοράς, θεωρείται ψευδής αν έχει τιμή `null`, αληθής διαφορετικά.

Η ύπαρξη του σκέλους `else` είναι προαιρετική: αν αυτό λείπει τότε θεωρείται κενό. Σημειώνεται επίσης ότι οποιοδήποτε από τα σκέλη της εντολής `if` μπορεί να αποτελείται από μια *σύνθετη εντολή*, να είναι δηλαδή ένα τμήμα κώδικα που περικλείεται από άγκιστρα. Παράδειγμα τέτοιας εντολής είναι το ακόλουθο:

```
if (x > 0)
   System.out.println("It was positive.");
else {
   System.out.println("It was not positive.");
   x = 1 - x;
   System.out.println("But now it is!");
}
```

Μερικές φορές η επιλογή δεν καθορίζεται από μια λογική συνθήκη, αλλά γίνεται βάση της τιμής μιας ακέραιας τιμής. Στην περίπτωση αυτή μπορεί να χρησιμοποιηθεί η εντολή `switch`, που εξηγείται αμέσως με το ακόλουθο παράδειγμα:

```
switch (count) {
   case 1:
      System.out.println("I have one.");
      break;
   case 2:
   case 3:
      System.out.println("I have two or three.");
      break;
   default:
      System.out.println("I have a few...");
      break;
}
```

Στο παραπάνω παράδειγμα, η έκφραση `count` αποτιμάται. Αν η τιμή της είναι `1`, τότε ο έλεγχος μεταφέρεται στην εντολή που ακολουθεί την ετικέττα `case 1`. Από το σημείο αυτό, εκτελούνται οι επόμενες εντολές μέχρι να συναντηθεί η εντολή `break` ή το άγκιστρο που κλείνει την εντολή `switch`. Και στις δυο περιπτώσεις, η εκτέλεση της εντολής `switch` τερματίζεται. Η ύπαρξη της εντολής `break` δεν είναι υποχρεωτική, αν όμως λείπει τότε συνεχίζονται να εκτελούνται εντολές που πιθανώς ανήκουν σε άλλες ετικέττες. Αυτό δεν είναι πάντα ανεπιθύμητο, όπως φαίνεται στην περίπτωση της ετικέττας `case 2` που δεν περιέχει εντολή, αλλά χρησιμοποιεί τις εντολές της ετικέττας `case 3`.

Η ειδική ετικέττα `default` χρησιμοποιείται όταν η τιμή της έκφρασης επιλογής δεν προβλέπεται σε κάποια άλλη ετικέττα. Για παράδειγμα, ο έλεγχος θα μεταφερθεί εκεί αν η τιμή της μεταβλητής count είναι ίση με `42`.

2.5.3. Εντολές βρόχων

Η Java υποστηρίζει τις εντολές βρόχων των γλωσσών C και C++. Υποστηρίζονται συγκεκριμένα τρεις εντολές βρόχων. Η πρώτη από αυτές είναι η εντολή `while`, που επαναλαμβάνει την εκτέλεση της εντολής που βρίσκεται στο κύριο σώμα της όσο η τιμή της συνθήκης της είναι `true`. Για παράδειγμα, στο παρακάτω τμήμα κώδικα:

```
{
   int i = 0;

   while (i < 10)                                                    (1)
      System.out.println(++i);
}
```

η εκτέλεση της εντολής `while` προκαλεί την εκτύπωση των αριθμών `1` ως `10`. Σημειώνεται ότι, αν η συνθήκη είναι εξ αρχής ψευδής, το κύριο σώμα της εντολής `while` δεν εκτελείται καμιά φορά.

Δυο ειδικές εντολές μπορούν να χρησιμοποιηθούν στο εσωτερικό κάθε βρόχου. Η εντολή `break` προκαλεί την άμεση έξοδο από το βρόχο, ενώ η εντολή `continue` προκαλεί την άμεση επανάληψη του βρόχου. Για παράδειγμα, στο παρακάτω τμήμα κώδικα:

```
{
   int i = 0;

   while (i < 10) {
      System.out.println(++i);
      if (i != 3)
         continue;
      break;
   }
}
```

μόνο οι αριθμοί `1`, `2` και `3` θα εκτυπωθούν — ο αναγνώστης καλείται να εξετάσει γιατί.

Παρόμοια είναι η εντολή `do … while`, με τη διαφορά ότι η λογική συνθήκη ελέγχεται αφού εκτελεσθεί το κύριο σώμα. Κατ' αυτό τον τρόπο, το κύριο σώμα εκτελείται τουλάχιστον μια φορά. Για παράδειγμα, στο παρακάτω τμήμα κώδικα:

```
{
   int i = 0;

   do {
      System.out.println(i);
      i = 10 - i;
   } while (i > 0);
}
```

η εκτέλεση της εντολής `do … while` προκαλεί την εκτύπωση των αριθμών `0` και `10`.

Η πιο πολύπλοκη εντολή βρόχου είναι η εντολή `for`, που μοιάζει πολύ με την αντίστοιχη της C/C++. Η σύνταξή της είναι η ακόλουθη:

```
for ( αρχικοποίηση ; συνθήκη ; ενημέρωση )
   εντολή
```

Το τμήμα αρχικοποίησης μπορεί να αποτελείται από μια ή περισσότερες εντολές ή δηλώσεις και αρχικοποιήσεις μεταβλητών, χωρισμένες με κόμματα. Εκτελείται μια μόνο φορά, πριν αποτιμηθεί για πρώτη φορά η συνθήκη. Το τμήμα ενημέρωσης αποτελείται από μια ή περισσότερες εντολές, χωρισμένες με κόμματα, και εκτελείται μετά από κάθε επανάληψη του βρόχου και πριν την αποτίμηση της συνθήκης. Για παράδειγμα, η ακόλουθη εντολή:

```
for (int i = 0 ; i < 10 ; i++)
   System.out.println(i);
```

είναι ισοδύναμη με το τμήμα κώδικα (1) που δόθηκε πιο πάνω ως παράδειγμα της εντολής `while`.

Οι εντολές βρόχων δέχονται προαιρετικά ετικέττες, που μπορούν να χρησιμοποιηθούν ως ορίσματα των εντολών `break` και `continue`. Μια τέτοια εντολή χωρίς όρισμα αναφέρεται στον πιο εσωτερικό βρόχο. Αντίθετα, η χρήση ορίσματος την κάνει να αναφέρεται στο βρόχο που φέρει την αντίστοιχη ετικέττα. Η χρήση ετικεττών είναι ιδιαίτερα χρήσιμη στην περίπτωση φωλιασμένων βρόχων. Για παράδειγμα, το παρακάτω τμήμα κώδικα:

```
{
   int [][] a = … ;   /* initialization */

outer:
   for (int i = 0 ; i < 10 ; i++)
      for (int j = 0 ; j < 10 ; j++) {
         if (a[i][j] == 42)
            continue;
         System.out.println(a[i][j]);
         if (a[i][j] == 0)
            break outer;
      }
}
```

εκτυπώνει το διδιάστατο πίνακα `a`, διαστάσεων `10×10`, παραλείποντας τα στοιχεία που έχουν τιμή `42`. Όταν βρεθεί κάποιο στοιχείο με τιμή `0`, η εκτύπωση σταταματά.

2.5.4. Άλλες εντολές

Η Java υποστηρίζει και άλλες εντολές που θα περιγραφούν αναλυτικότερα σε επόμενα κεφάλαια. Πρόκειται για την εντολή `return`, που χρησιμοποιείται για την επιστροφή τιμής από μέθοδο, και τις εντολές `try`, `catch` και `finally` που χρησιμοποιούνται για το χειρισμό εξαιρέσεων.

3. Αντικείμενα και κλάσεις

Στο κεφάλαιο αυτό παρουσιάζονται μέσω παραδειγμάτων τα χαρακτηριστικά αντικειμενοστρεφούς προγραμματισμού (object-oriented programming) της Java, που αποτελούν τον πυρήνα της γλώσσας. Επίσης, παρουσιάζεται το σύστημα οργάνωσης του εκτελέσιμου κώδικα στη Java μέσω "πακέτων".

3.1. Αντικειμενοστρεφής προγραμματισμός

Το *μοντέλο αντικειμενοστρεφούς ανάπτυξης λογισμικού* (object-oriented software development) αποσκοπεί στην οργάνωση του λογισμικού σε ένα σύνολο από αλληλεπιδρώντα *αντικειμένα* (objects). Η επιλογή αυτών των αντικειμένων υπαγορεύεται συνήθως από το φυσικό κόσμο και το πρόβλημα που πρόκειται να επιλυθεί. Κάθε αντικείμενο περιέχει κάποια δεδομένα, που χαρακτηρίζουν την κατάστασή του, και υλοποιεί κάποια συμπεριφορά μέσω κατάλληλων τμημάτων κώδικα. Βασικό χαρακτηριστικό του αντικειμενοστρεφούς προγραμματιστικού μοντέλου, επομένως, είναι η αφαιρετική από κοινού αντιμετώπιση της κατάστασης και της συμπεριφοράς των αντικειμένων, που ονομάζεται *κελυφοποίηση* (encapsulation).

Συχνά στην ανάπτυξη λογισμικού παρουσιάζεται η ανάγκη για την ύπαρξη περισσοτέρων αντικειμένων που είναι της ίδιας ή παρόμοιας μορφής. Για το λόγο αυτό, εξέχουσα θέση στο μοντέλο αντικειμενοστρεφούς ανάπτυξης λογισμικού κατέχει η έννοια της *κλάσης* (class). Οι κλάσεις είναι κατηγορίες αντικειμένων με την ίδια συμπεριφορά: αντικείμενα που προέρχονται από την ίδια κλάση είναι δυνατό να διαφέρουν μόνο στην κατάσταση. Εξέχουσα θέση έχουν επίσης οι έννοιες της *κληρονομικότητας* (inheritance) και του *πολυμορφισμού υπο-τύπων* (subtype polymorphism), που θα περιγραφούν σε επόμενες ενότητες. Με τη δημιουργία κλάσεων και ιεραρχιών από κλάσεις, όπως θα δούμε στη συνέχεια, το μοντέλο αντικειμενοστρεφούς ανάπτυξης λογισμικού συμβάλλει άμεσα προς την κατεύθυνση της *επαναχρησιμοποίησης λογισμικού* (software reusability).

Έχουν προταθεί αρκετές μεθοδολογίες για την αντικειμενοστρεφή ανάπτυξη λογισμικού, με ιδιαίτερη έμφαση στην ανάλυση και τη σχεδίαση. Η περιγραφή μιας τέτοιας μεθοδολογίας είναι έξω από τους στόχους αυτών των σημειώσεων και ο αναγνώστης παραπέμπεται στην εκτενή βιβλιογραφία. Ωστόσο, είναι χρήσιμο να αναφερθούν ορισμένες βασικές αρχές που υιοθετούνται από όλες σχεδόν τις προτεινόμενες μεθοδολογίες.

- Κατά την ανάλυση και τη σχεδίαση, σκεφτείτε και αντιμετωπίστε τα αντικείμενα με βάση τη διαπροσωπεία (interface) τους, και όχι με βάση την υλοποίησή (implementation) τους. Η διαπροσωπεία ενός αντικειμένου είναι μια περιγραφή του τρόπου με το οποίο αυτό

αλληλεπιδρά με άλλα αντικείμενα και οφείλει να είναι εύκολα κατανοητή. Η υλοποίηση ενός αντικειμένου, φυσικά, μπορεί να είναι εξαιρετικά πολύπλοκη.

- Για κάθε αντικείμενο, προσπαθείστε να “κρύψετε” όσο το δυνατόν μεγαλύτερο τμήμα από την υλοποίησή του. Με αυτό τον τρόπο, μπορείτε αργότερα να αλλάξετε την υλοποίηση του αντικειμένου χωρίς κακά συνεπακόλουθα.
- Προσπαθείστε να χρησιμοποιείτε αντικείμενα που έχουν ήδη υλοποιηθεί. Εξειδικεύστε τα και δημιουργείστε νέα αντικείμενα μόνο αν δε γίνεται διαφορετικά.
- Προσπαθείστε να ελαχιστοποιήσετε τη διασύνδεση και την αλληλεπίδραση μεταξύ αντικειμένων. Τα αντικείμενα πρέπει να είναι όσο το δυνατόν αυτοτελή.

3.2. Αντικείμενα και κλάσεις

Προκειμένου να χρησιμοποιηθεί ένα αντικείμενο στη Java είναι απαραίτητο να ορισθεί μια κλάση που να το περιγράφει. Οι κλάσεις, όπως αναφέρθηκε στην προηγούμενη ενότητα, είναι κατηγορίες αντικειμένων με την ίδια συμπεριφορά. Αν και η κατάστασή τους είναι δυνατό να διαφέρει από αντικείμενο σε αντικείμενο, οφείλει γενικά να έχει την ίδια μορφή. Οι κλάσεις ορίζονται στη Java με χρήση της δήλωσης `class`. Κάθε κλάση χαρακτηρίζεται από ένα μοναδικό όνομα.[4] Τα περιεχόμενα μιας κλάσης ονομάζονται *μέλη της κλάσης* (class members) και μπορούν να είναι *μεταβλητές* (variables), που χαρακτηρίζουν την κατάσταση των αντικειμένων, ή *μέθοδοι* (methods), που χαρακτηρίζουν τη συμπεριφορά τους.

Στην επόμενη δήλωση ορίζεται μια κλάση με όνομα `Counter`, που υλοποιεί ένα μετρητή. Περιέχει τη μεταβλητή `value`, στην οποία φυλάσσεται η τρέχουσα τιμή του μετρητή και η οποία αρχικοποιείται στο `0`. Επίσης, περιέχει δυο μεθόδους `inc` και `get`, για την αύξηση και την επιστροφή της τιμής του μετρητή αντίστοιχα. Ας σημειωθεί ότι στη δήλωση των μεθόδων `inc` και `get` συμπεριλαμβάνεται ο κώδικας που τις υλοποιεί. Η κλάση `Counter` είναι μια πολύ απλή κλάση. Ορισμοί πιο πολύπλοκων κλάσεων μπορούν να περιέχουν πολλές σελίδες κώδικα.

```
class Counter
{
   int value = 0;

   void inc () { value++; }
   int  get () { return value; }
}
```

Αφού ορισθεί η κλάση `Counter`, είναι δυνατή η κατασκευή αντικειμένων που προέρχονται από αυτή την κλάση. Τα αντικείμενα αυτά ονομάζονται και *εμφανίσεις* (instances) της κλάσης και, στην προκειμένη περίπτωση, είναι ανεξάρτητοι μετρητές. Η διαδικασία κατασκευής αντικειμένων περιγράφεται στην επόμενη ενότητα. Η χρήση των αντικειμένων μέσα στον κώδικα γίνεται μέσω μεταβλητών τύπου αναφοράς, που περιέχουν αναφορές στα αντικείμενα.

Οι μεταβλητές που δηλώνονται στο εσωτερικό μιας κλάσης χαρακτηρίζουν την κατάσταση των αντικειμένων της κλάσης. Μπορούν να είναι οποιουδήποτε έγκυρου τύπου και μπορούν να αρχικοποιηθούν. Οι μέθοδοι υλοποιούν τη συμπεριφορά του αντικειμένου και είναι τμήματα

[4] Για την ακρίβεια, τα ονόματα των κλάσεων πρέπει να είναι μοναδικά μέσα στα πακέτα όπου ορίζονται.

κώδικα που μπορούν να κληθούν. Δέχονται έναν αριθμό παραμέτρων (στην περίπτωση της κλάσης `Counter` και οι δυο μέθοδοι δε δέχονται παραμέτρους) και επιστρέφουν ένα αποτέλεσμα κάποιου τύπου (η μέθοδος `get` επιστρέφει αποτέλεσμα τύπου `int`), εκτός αν δηλωθούν ως `void`, οπότε δεν επιστρέφουν κανένα αποτέλεσμα (όπως η μέθοδος `inc`). Στο εσωτερικό των μεθόδων, οι μεταβλητές των αντικειμένων είναι ορατές και προσπελαύνονται με τη χρήση μόνο του ονόματός τους.

Αν υποτεθεί ότι η μεταβλητή `c` είναι μια αναφορά σε ένα αντικείμενο της κλάσης `Counter`, τότε η προσπέλαση της μεταβλητής `value` αυτού του αντικειμένου γίνεται με την έκφραση `c.value`, όπως για παράδειγμα στην εντολή:

```
if (c.value > 10)
   c.value = 0;
```

Φυσικά, στην κλάση `Counter` οι μέθοδοι `inc` και `get` παρέχουν τις επιθυμητές λειτουργίες και η μεταβλητή `value` θα ήταν καλό να μην είναι προσπελάσιμη, αλλά στο θέμα αυτό θα επανέλθουμε αργότερα.

Με παρόμοιο τρόπο γίνεται η κλήση μεθόδων. Για παράδειγμα η κλήση της μεθόδου get για το αντικείμενο στο οποίο αναφέρεται η μεταβλητή `c` γίνεται με την έκφραση `c.get()`. Στο εσωτερικό των παρενθέσεων τοποθετούνται οι πραγματικές τιμές των παραμέτρων, αν αυτές υπάρχουν. Η τιμή αυτής της έκφρασης είναι αυτή που επιστρέφεται από το σώμα της μεθόδου, μέσω της εντολής `return`, και είναι του ίδιου τύπου με αυτόν που δηλώνεται στον ορισμό της μεθόδου. Μέθοδοι που δεν επιστρέφουν αποτέλεσμα μπορούν να χρησιμοποιηθούν ως εντολές εκφράσεων. Ένα εκτενέστερο παράδειγμα που χρησιμοποιεί την κλάση `Counter` είναι το ακόλουθο:

```
c.inc();
System.out.println(c.get());
```

Προκειμένου να φανεί η χρήση παραμέτρων, ας ξαναγράψουμε την κλάση Counter, συμπεριλαμβάνοντας επιπλέον μια ακόμα μέθοδο:

```
void set (int newValue) { value = newValue; }
```

Η μέθοδος αυτή τοποθετεί μια νέα τιμή στο μετρητή. Η παράμετρος `newValue` έχει τύπο `int` και καθορίζει ακριβώς ποια θα είναι αυτή η τιμή. Για να κληθεί η μέθοδος `set` πρέπει να δοθεί μια συγκεκριμένη πραγματική τιμή στην παράμετρο `newValue`, όπως φαίνεται στον κώδικα που ακολουθεί:

```
c.set(5);
if (c.get() != 5)
   // die a horrible death, everything was a lie!
```

Ο κώδικας μιας μεθόδου είναι δυνατό να περιέχει τη δήλωση νέων μεταβλητών. Οι μεταβλητές αυτές ονομάζονται *τοπικές* (local) και είναι ορατές μόνο μέσα στον κώδικα της συγκεκριμένης μεθόδου. Πρέπει να αρχικοποιηθούν ρητά μέσα στο πρόγραμμα και δε διατηρούν την τιμή τους μεταξύ διαδοχικών κλήσεων. Με άλλα λογια, η διάρκεια ζωής μιας τοπικής μεταβλητής συμπίπτει με το χρόνο κατά τον οποίο πραγματοποιείται η κλήση της μεθόδου. Για παράδειγμα, ας θεωρήσουμε μια υποθετική παραλλαγή της μεθόδου `set`, η οποία θα αλλάζει την τιμή του μετρητή και θα επιστρέφει την παλιά τιμή. Για το λόγο αυτό απαιτείται η τοπική μεταβλητή

oldValue, στην οποία φυλάσσεται η παλιά τιμή του μετρητή για να μη χαθεί από την εκχώρηση.

```
int set (int newValue)
{
   int oldValue = value;

   value = newValue;
   return oldValue;
}
```

Οι μεταβλητές διαφορετικών αντικειμένων που προέρχονται από μια κλάση είναι εντελώς ανεξάρτητες. Μερικές φορές όμως, είναι χρήσιμο κάποιες μεταβλητές να είναι κοινές για όλα τα αντικείμενα μιας κλάσης. Επίσης, είναι χρήσιμο η κλήση κάποιων μεθόδων να μην αναφέρεται σε συγκεκριμένα αντικείμενα. Αυτές οι ανάγκες υλοποιούνται μέσω των μεταβλητών και των μεθόδων που δηλώνονται ως static. Ας θεωρήσουμε την ακόλουθη έκδοση του μετρητή Counter:

```
class CounterLimited
{
   static int limit = 100;
   int value = 0;

   void inc () { if (++value >= limit) value = 0; }
   int  get () { return value; }

   static void setLimit (int l) { limit = l; }
}
```

Στους μετρητές που προέρχονται από αυτή την κλάση, η τιμή μηδενίζεται όταν υπερβεί κάποιο όριο. Η τιμή αυτού του άνω όριου καθορίζεται από τη μεταβλητή limit και είναι κοινή για όλα τα αντικείμενα της κλάσης. Η μέθοδος setLimit, που πάλι είναι ανεξάρτητη αντικειμένου, μπορεί να χρησιμοποιηθεί για την αλλαγή της τιμής του άνω ορίου. Τόσο η μεταβλητή limit όσο και η μέθοδος setLimit μπορούν να προσπελασθούν είτε μέσω μιας αναφοράς σε ένα αντικείμενο τύπου CounterLimited, όπως και οι κοινές μεταβλητές και μέθοδοι, είτε μέσω του ονόματος της ίδιας της κλάσης, όπως για παράδειγμα:

```
CounterLimited.setLimit(1000);
if (CounterLimited.limit != 1000)
   // go get a beer and forget about Java!
```

Αν η αρχικοποίηση μεταβλητών που δηλώνονται ως static δεν είναι δυνατό να πραγματοποιηθεί κατά τη δήλωσή τους, η Java υποστηρίζει τμήματα κώδικα που δηλώνονται ως static και συμπεριλαμβάνονται στον ορισμό των κλάσεων. Σε κάθε κλάση μπορεί να ορισθεί ένα τέτοιο τμήμα, που μπορεί να προσπελαύνει μόνο μεταβλητές και μεθόδους δηλωμένες ως static. Το τμήμα αυτό θα εκτελεσθεί την πρώτη φορά που θα φορτωθεί η κλάση. Παράδειγμα τέτοιου τμήματος κώδικα δίνεται στην επόμενη παραλλαγή της κλάσης Counter:

```
class NewCounterLimited
{
   static int limit;

   static {
      System.out.println(
         "What do you think a fair limit is?");
```

```
            limit = user.askAboutTheLimit();
        }

        // rest of the class as usual ...
    }
```

Το τμήμα κώδικα που δηλώνεται ως `static` αρχικοποιεί τη μεταβλητή `limit` ρωτώντας το χρήστη για την επιθυμητή τιμή της, μέσω μιας υποθετικής μεθόδου.

3.2.1. Υπερφόρτωση μεθόδων

Με τον όρο *υπερφόρτωση μεθόδων* (method overloading) εννοούμε τη δυνατότητα ορισμού περισσότερων μεθόδων με το ίδιο όνομα μέσα στην ίδια κλάση. Κάτι τέτοιο είναι ιδιαίτερα χρήσιμο στην περίπτωση που οι μέθοδοι υλοποιούν παρόμοιες λειτουργίες για ορίσματα που ανήκουν σε διαφορετικούς τύπους. Στην περίπτωση αυτή, η επιλογή της μεθόδου που καλείται γίνεται από το μεταγλωττιστή, ανάλογα με τον τύπο των ορισμάτων.

Η Java υποστηρίζει υπερφόρτωση μεθόδων, αρκεί οι υπερφορτωμένες μέθοδοι να διαφέρουν στον αριθμό ή/και τον τύπο των ορισμάτων. Αυτός ο τύπος υπερφόρτωσης συχνά ονομάζεται *παραμετρικός πολυμορφισμός* (parametric polymorphism). Παράδειγμα υπερφορτωμένης μεθόδου είναι η `println`, που χρησιμοποιείται για την εκτύπωση δεδομένων. Στην πραγματικότητα, η κλάση `PrintStream` της Java περιέχει πολλές παραλλαγές της μεθόδου, μια για κάθε διαφορετικό τύπο ορίσματος του οποίου η εκτύπωση είναι επιθυμητή:

```
class PrintStream
{
    ...

    void println (int i)    { /* print an int   */ }
    void println (double d) { /* print a double */ }
    void println (String s) { /* print a string */ }

    ...
}
```

Η υπερφόρτωση μεθόδων είναι ένα πολύτιμο εργαλείο της Java. Δεν πρέπει όμως να γίνεται κατάχρηση. Αν κάποιες μέθοδοι δεν υλοποιούν παρόμοιες λειτουργίες, η υπερφόρτωση του ίδιου ονόματος οδηγεί συνήθως σε παρερμηνείες και πρέπει να αποφεύγεται.

3.2.2. Κατασκευή αντικειμένων

Αφού ορισθεί μια κλάση, είναι δυνατό να κατασκευασθούν αντικείμενα που προέρχονται από αυτήν. Για να κατασκευασθεί ένα αντικείμενο, απαιτούνται συνήθως δυο βήματα:

- Δήλωση μιας μεταβλητής με τύπο αναφοράς, το οποίο θα αναφέρεται στο αντικείμενο που θα κατασκευασθεί, π.χ.

  ```
  Counter c;
  ```

Υπενθυμίζεται ότι η δήλωση αυτή δεν κατασκευάζει κανένα αντικείμενο. Η τιμή της μεταβλητής `c`, αν αρχικοποιηθεί αυτόματα, παίρνει την τιμή `null`.

- Κατασκευή του αντικειμένου με χρήση του τελεστή `new` και εκχώρηση της αναφοράς στη μεταβλητή.

```
c = new Counter();
```

Φυσικά, τα δυο βήματα μπορούν να συνενωθούν σε μια δήλωση με αρχικοποίηση:

```
Counter c = new Counter();
```

Στη συνέχεια, το αντικείμενο μπορεί να χρησιμοποιηθεί μέσω της αναφοράς που βρίσκεται στη μεταβλητή `c`. Ας σημειωθεί ότι μπορούν να υπάρχουν περισσότερες αναφορές στο ίδιο αντικείμενο, όπως γίνεται αντιληπτό στο ακόλουθο τμήμα κώδικα:

```
Counter other = c;   // refers to the same counter

c.set(42);
if (other.get() != 42)
   // it's the end of the world as we know it...
```

Η κατασκευή ενός αντικειμένου με τον τελεστή `new` προκαλεί την αρχικοποίηση των μεταβλητών του. Οι τιμές που χρησιμοποιούνται για την αρχικοποίηση μπορούν να καθορισθούν κατά τη δήλωση των μεταβλητών. Αν δε συμβεί αυτό, οι μεταβλητές αρχικοποιούνται αυτόματα. Μερικές φορές όμως, η κατασκευή ενός αντικειμένου απαιτεί κάποια ιδιαίτερη αρχικοποίηση, που δεν είναι δυνατό να πραγματοποιηθεί με τον τρόπο αυτό. Την ανάγκη αυτή καλύπτει ένα ιδιαίτερο είδος μεθόδου, που ονομάζεται *κατασκευαστής* (constructor). Οι κατασκευαστές είναι μέθοδοι που έχουν το όνομα της αντίστοιχης κλάσης. Μπορούν να έχουν παραμέτρους, αλλά δεν επιστρέφουν αποτέλεσμα και αυτό δε χρειάζεται να δηλωθεί. Στο ακόλουθο τμήμα κώδικα, ορίζεται πάλι η κλάση `Counter` συμπεριλαμβάνοντας έναν κατασκευαστή. Αυτός δέχεται ως παράμετρο την αρχική τιμή του μετρητή.

```
class Counter
{
   int value;

   Counter (int v) { value = v; }

   ...
}
```

Στη συνέχεια, για να κατασκευαστεί ένα αντικείμενο απαιτείται ο καθορισμός της παραμέτρου του κατασκευαστή, δηλαδή της αρχικής τιμής του μετρητή, όπως σε κάθε κλήση μεθόδου. Για παράδειγμα:

```
Counter c = new Counter(1);
```

Οι κατασκευαστές είναι δυνατό να υπερφορτωθούν, όπως και οι απλές μέθοδοι. Για παράδειγμα, ας θεωρήσουμε την ακόλουθη παραλλαγή της κλάσσης `Counter`, με δυο κατασκευαστές: ο πρώτος δέχεται ως παράμετρο την αρχική τιμή του μετρητή, ενώ ο δεύτερος δε δέχεται παραμέτρους και θεωρεί ότι η αρχική τιμή είναι `0`. Και στις δυο περιπτώσεις, ο κατασκευαστής ανακοινώνει την κατασκευή ενός νέου αντικειμένου.

```
class HappyCounter
{
   int value;
```

```
    HappyCounter (int v)
    {
       System.out.println("Creating a new counter.")
       value = v;
    }

    HappyCounter ()
    {
       System.out.println("Creating a new counter.")
       value = 0;
    }

    ...
}
```

Διαπιστώνει κανείς ότι, στον παραπάνω ορισμό, ο δεύτερος κατασκευαστής είναι μια υποπερίπτωση του πρώτου, όπου η τιμή της παραμέτρου `v` είναι `0`. Η περίπτωση αυτή, όπου ένας κατασκευαστής μπορεί να χρησιμοποιηθεί για την υλοποίηση ενός άλλου, εμφανίζεται συχνά στην πράξη και υποστηρίζεται από τη Java. Ο δεύτερος κατασκευαστής θα μπορούσε να γραφεί ως:

```
HappyCounter () { this(0); }
```

όπου η εντολή `this(0);` είναι μια κλήση στον πρώτο κατασκευαστή με τιμή παραμέτρου `0`. Τέτοιου είδους κλήσεις είναι υποχρεωτικά οι πρώτες εντολές μέσα στο σώμα ενός κατασκευαστή. Έχοντας τους δυο κατασκευαστές για την κλάση `HappyCounter`, είναι δυνατή η επιλογή του κατασκευαστή που θα χρησιμοποιηθεί:

```
HappyCounter c1 = new HappyCounter();    // value = 0
HappyCounter c2 = new HappyCounter(42);  // value = 42
```

Η ειδική μεταβλητή `this` μπορεί να χρησιμοποιηθεί οπουδήποτε στο κύριο σώμα μιας μεθόδου. Περιέχει πάντα μια αναφορά στο ίδιο το αντικείμενο, για το οποίο καλείται η μέθοδος. Για παράδειγμα, η μέθοδος `get` της κλάσης `Counter` θα μπορούσε να γραφεί ως:

```
int get () { return this.value; }
```

αν και κάτι τέτοιο θα ήταν μάλλον περιττό.

3.2.3. Καταστροφή αντικειμένων

Αντίθετα με την κατασκευή αντικειμένων, που πρέπει να γίνει ρητά μέσα στον κώδικα με χρήση του τελεστή `new`, η καταστροφή των αντικειμένων γίνεται αυτόματα από το περιβάλλον εκτέλεσης της Java όταν τα αντικείμενα πάψουν να χρησιμοποιούνται. Το περιβάλλον εκτέλεσης θεωρεί ότι ένα αντικείμενο παύει να χρησιμοποιείται όταν δεν υπάρχει καμιά μεταβλητή που να περιέχει αναφορά σε αυτό. Αν συμβαίνει κάτι τέτοιο, τότε δεν υπάρχει τρόπος πρόσβασης στο αντικείμενο και αυτό μπορεί ασφαλώς να καταστραφεί.

Το τμήμα του περιβάλλοντος εκτέλεσης της Java που αναλαμβάνει την παρακολούθηση της χρήσης των αντικειμένων και την καταστροφή τους όταν πάψουν να χρησιμοποιούνται ονομάζεται *συλλέκτης σκουπιδιών* (garbage collector). Ο συλλέκτης σκουπιδιών εκτελείται

συνήθως παράλληλα με το πρόγραμμα και απελευθερώνει τη μνήμη που καταλαμβάνεται από άχρηστα αντικείμενα. Μπορεί όμως να ενεργοποιηθεί ρητά με κλήση της μεθόδου `System.gc()`.

Μερικές φορές η καταστροφή των αντικειμένων από το συλλέκτη σκουπιδιών δεν είναι αρκετή. Ορισμένα αντικείμενα είναι δυνατό να χρειάζονται ειδική μεταχείριση κατά την καταστροφή τους, π.χ. για να κλείσουν κάποια αρχεία ή για να απελευθερώσουν πόρους του συστήματος που έχουν στην κατοχή τους. Για το λόγο αυτό η Java υποστηρίζει τη μέθοδο `finalize`, που καλείται όταν ένα αντικείμενο καταστρέφεται. Η μέθοδος αυτή μπορεί να περιέχει κώδικα που αναλαμβάνει την εκκαθάριση των ανοικτών λογαριασμών των αντικειμένων, πριν αυτά καταστραφούν. Όμως, επειδή η διαδικασία της συλλογής σκουπιδιών ελέγχεται απευθείας από το περιβάλλον εκτέλεσης, ο ακριβής χρόνος της κλήσης της μεθόδου `finalize` δεν είναι γνωστός.

3.2.4. Μετατροπείς ορατότητας

Ο κώδικας που βρίσκεται στο εσωτερικό των μεθόδων μιας κλάσης έχει πάντα πρόσβαση στα μέλη της κλάσης. Όμως, η δυνατότητα πρόσβασης στα μέλη μιας κλάσης από κώδικα που δεν ανήκει σε μεθόδους της δεν είναι πάντα χρήσιμη. Μερικές φορές, στο πλαίσιο της κελυφοποίησης των αντικειμένων, είναι σκόπιμη η απόκρυψη της υλοποίησης μιας κλάσης και η παροχή προς τα έξω μόνο ορισμένων μεθόδων, που αποτελούν τη διαπροσωπεία της κλάσης. Για το λόγο αυτό η Java υποστηρίζει ένα σύνολο *μετατροπέων ορατότητας* (visibility modifiers).

Μέχρι τώρα, σε όλες τις κλάσεις που ορίστηκαν δε χρησιμοποιήθηκε κανένας μετατροπέας ορατότητας. Τα μέλη μιας κλάσης που ορίζονται κατ' αυτό τον τρόπο είναι ορατά από κάθε τμήμα κώδικα, είτε ανήκει στην κλάση αυτή είτε όχι.[5] Κατά συνέπεια, η τιμή της μεταβλητής `value` κάποιου αντικειμένου που προέρχεται από την κλάση `Counter` είναι δυνατό να μεταβληθεί με αυθαίρετο τρόπο, χωρίς να κληθούν οι μέθοδοι της κλάσης. Αυτό είναι ανεπιθύμητο στην περίπτωση αυτή και θα διορθωθεί στην ακόλουθη παραλλαγή της κλάσης, που χρησιμοποιεί τον μετατροπέα ορατότητας `private`.

```
class Counter
{
   private int value;

   Counter ()      { value = 0; }
   Counter (int v) { value = v; }

   void inc () { value++; }
   int  get () { return value; }
}
```

Στη νέα έκδοση της κλάσης, η μεταβλητή `value` δηλώνεται ως `private`. Αυτό σημαίνει ότι είναι ορατή μόνο από το εσωτερικό της κλάσης, και κατά συνέπεια, το επόμενο τμήμα κώδικα οδηγεί σε σφάλμα μεταγλώττισης:

```
Counter c = new Counter();
c.value = 42;                // compilation error!
```

[5] Η δήλωση αυτή δεν είναι απόλυτα ακριβής. Για μια πλήρη εικόνα των μετατροπέων ορατότητας, ο αναγνώστης παραπέμπεται στην ενότητα 3.5.2, αφού συζητηθεί ο μηχανισμός οργάνωσης κώδικα σε πακέτα.

Φυσικά είναι δυνατός ο ορισμός μεθόδων με το μετατροπέα `private`. Στην περίπτωση αυτή, οι μέθοδοι μπορούν να καλούνται μόνο μέσα από την κλάση `Counter`.

Εκτός από τον `private`, υπάρχουν και άλλοι μετατροπείς ορατότητας. Η περιγραφή τους όμως αναβάλλεται για λίγο, γιατί πρέπει να προηγηθεί ο ορισμός των ιεραρχιών κλάσεων και του μηχανισμού οργάνωσης κώδικα σε πακέτα. Στο θέμα της ορατότητας θα επανέλθουμε στην ενότητα 3.5.2.

3.3. Ιεραρχίες κλάσεων

Ένα από τα βασικά χαρακτηριστικά του μοντέλου αντικειμενοστρεφούς ανάπτυξης προγραμμάτων είναι η υποστήριξη της *κληρονομικότητας* (inheritance). Η κληρονομικότητα, μεσω της οποίας οι κλάσεις σχηματίζουν ιεραρχίες, καλύπτει πρωτίστως την ανάγκη εξειδίκευσης των κλάσεων, χρησιμοποιώντας κατά τον καλύτερο δυνατό τρόπο ήδη ορισμένες κλάσεις. Ας τα δούμε όλα από την αρχή με ένα παράδειγμα.

Ας υποτεθεί ότι η κλάση `Counter` έχει ήδη ορισθεί και χρησιμοποιηθεί σε έναν αριθμό εφαρμογών με μεγάλη επιτυχία. Κάποια στιγμή όμως, σε μια νέα εφαρμογή παρουσιάζεται η ανάγκη για μετρητές που, εκτός από μια μέθοδο `inc` για την αύξηση της τιμής τους, απαιτούν και μια μέθοδο `dec` για τη μείωση. Η προφανής λύση του προβλήματος είναι να προστεθεί μια νέα μέθοδος στην κλάση `Counter`. Αυτό όμως δεν είναι πάντα επιθυμητό, και, σε μερικές περιπτώσεις δεν είναι καν εφικτό. Μερικά πιθανά επιχειρήματα μιας υποθετικής ομάδας ανάπτυξης λογισμικού κατά της τροποποίησης της κλάσης `Counter` είναι τα ακόλουθα:

- "Η κλάση `Counter` έχει χρησιμοποιηθεί αρκετά στις εφαρμογές μας. Για ποιο λόγο να επέμβουμε σε κώδικα που λειτουργεί με επιτυχία, με κίνδυνο οι αλλαγές που θα γίνουν να καταστρέψουν την ευστάθεια των ήδη υπάρχουσων εφαρμογών;" Σημειώνεται ότι οι εφαρμογές Java είναι ιδιαίτερα ευπαθείς σε τέτοιες μεταβολές, αφού ο κώδικας των κλάσεων φορτώνεται δυναμικά.
- "Η κλάση `Counter` είναι μια αυτοτελής κλάση και κάλυψε πλήρως τις ανάγκες μας μέχρι τώρα. Σε πληθώρα περιπτώσεων, η μέθοδος `dec` είναι άχρηστη. Γιατί λοιπόν να επιβαρύνουμε την κλάση με κάτι που συνήθως είναι περιττό;"
- "Η κλάση `Counter` δεν είναι καν δική μας. Έχει αναπτυχθεί από την εταιρεία Χ έχουμε αγοράσει τα δικαιώματα χρήσης της. Πώς θα τροποποιήσουμε κώδικα που δε μας ανήκει;"

Γίνεται αμέσως αντιληπτό ότι αυτό που πρέπει να γίνει δεν είναι η τροποποίηση της κλάσης `Counter`, αλλά η δημιουργία μιας νέας εξειδικευμένης κλάσης `CounterDec`, που θα παρέχει επιπλέον τη μέθοδο `dec`. Χωρίς το μηχανισμό της κληρονομικότητας, κάτι τέτοιο θα σήμαινε το γράψιμο αρκετού κώδικα. Ας δούμε όμως πώς κάτι τέτοιο είναι άμεσα υλοποιήσιμο στη Java, με ελάχιστο κόπο:

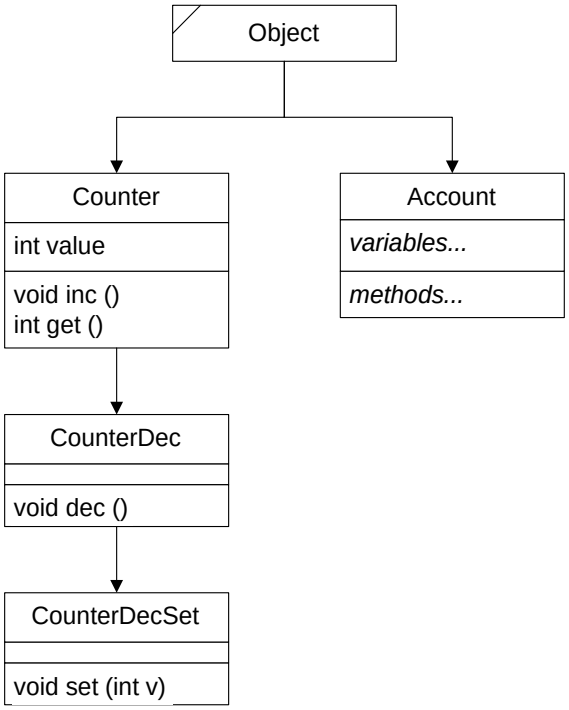
```
class CounterDec extends Counter
{
   void dec () { if (value > 0) value--; }
}
```

Με τη δήλωση `extends Counter`, καθορίζεται ότι η κλάση `CounterDec` ορίζεται ως μια επέκταση της `Counter`. Αυτό σημαίνει ότι κληρονομεί όλα τα μέλη της `Counter` και, στην ουσία, ότι τα αντικείμενα της κλάσης `CounterDec` μπορούν να χρησιμοποιηθούν οπουδήποτε

είναι δυνατή η χρήση αντικειμένων της κλάσης `Counter`. Η κλάση `Counter` ονομάζεται *βάση* (base class) ενώ η κλάση `CounterDec` ονομάζεται *παραγόμενη* (derived class). Επιπλέον, στην `CounterDec` δηλώνεται η μέθοδος `dec`, που χρησιμοποιεί τη μεταβλητή `value` της κλάσης βάσης.

Με το μηχανισμό κληρονομικότητας που μόλις περιγράφηκε είναι δυνατή η κατασκευή ιεραρχιών κλάσεων. Η κλάση βάσης βρίσκονται ένα βήμα υψηλότερα στην ιεραρχία από τις αντίστοιχες παραγόμενες κλάσεις, που μπορούν με τη σειρά τους να χρησιμοποιηθούν ως κλάσεις βάσεις για νέες παραγωγές, κ.ο.κ. Η ιεραρχία κλάσεων της Java έχει τη μορφή δέντρου. Στην κορυφή της βρίσκεται η ειδική κλάση `Object`, που μπορεί να θεωρηθεί ως ο "πατέρας" όλων των κλάσεων, αφού όλες την κληρονομούν.

Ας υποθέσουμε ότι η κλάση `CounterDecSet` επεκτείνει με τη σειρά της την κλάση `CounterDec` με την προσθήκη της μεθόδου `set`. Επιπλέον, ας υποτεθεί ότι ορίζεται και μια κλάση με όνομα `Account`. Η ιεραρχία κλάσεων για μια εφαρμογή που χρησιμοποιεί μόνο αυτές τις δυο κλάσεις απεικονίζεται στο Σχήμα 3.1.



Σχήμα 3.1. Παράδειγμα ιεραρχίας κλάσεων.

3.3.1. Κληρονομικότητα και κατασκευαστές

Ο ορισμός κατασκευαστών για κλάσεις που κληρονομούν παρουσιάζει ορισμένες ιδιαιτερότητες, αφού εκτός από την αρχικοποίηση των μελών της παραγόμενης κλάσης πρέπει να αρχικοποιηθούν και τα μέλη της κλάσης βάσης. Ας δούμε πώς θα μπορούσαν να ορισθούν κατασκευαστές για την κλάση `CounterDec`.

```
class CounterDec extends Counter
{
   CounterDec ()      { super();  }
   CounterDec (int v) { super(v); }
```

```
   void dec () { if (value > 0) value--; }
}
```

Η ειδική μεταβλητή `super` αντιστοιχεί σε μια αναφορά στο αντικείμενο σαν αυτό να ανήκε στην κλάση βάση. Όπως και η ειδική μεταβλητή `this`, μπορεί να χρησιμοποιηθεί στην πρώτη εντολή των κατασκευαστών για την αρχικοποίηση της κλάσης βάσης. Οι δυο κατασκευαστές που ορίσθηκαν πιο πάνω αντιστοιχούν στους δυο κατασκευαστές της κλάσης `Counter`. Η εντολή `super()` είναι μια κλήση στον κατασκευαστή της `Counter` χωρίς παραμέτρους, ενώ η `super(v)` είναι μια κλήση στο δεύτερο κατασκευαστή της `Counter`, με παράμετρο `v`.

Αν τα μέλη της κλάσης βάσης δεν αρχικοποιηθούν σε κάποιο κατασκευαστή με κατάλληλη κλήση κατασκευαστή μέσω της `super`, τότε ο μεταγλωττιστής της Java αυτόματα εισάγει μια κλήση στον κατασκευαστή χωρίς ορίσματα της κλάσης βάσης. Αν στην κλάση βάση ορίζονται κατασκευαστές, αλλά κανείς από αυτούς δεν είναι χωρίς ορίσματα, τότε συμβαίνει σφάλμα μεταγλώττισης.

3.3.2. Υπο-τύποι και μετατροπή τύπων

Όπως ήδη αναφέρθηκε, η κληρονομικότητα στη Java χρησιμοποιείται για την εξειδίκευση κλάσεων. Η παραγόμενη κλάση είναι μια εξειδικευμένη επέκταση της κλάσης βάσης, τα αντικείμενά της όμως δεν παύουν να μπορούν να χρησιμοποιηθούν σαν να ήταν αντικείμενα της κλάσης βάσης. Για παράδειγμα, ένα αντικείμενο της κλάσης `CounterDec` μπορεί να χρησιμοποιηθεί στο παρακάτω τμήμα κώδικα:

```
CounterDec c = new CounterDec();

c.inc();
if (c.get() < 10)
   // do something
```

όπου οι εξειδικευμένη μέθοδος `dec` δε χρησιμοποιείται. Για το λόγο αυτό, αφού τα αντικείμενα της παραγόμενης κλάσης διατηρούν όλες τις ιδιότητες της κλάσης βάσης, η παραγόμενη κλάση μπορεί να θεωρηθεί ως ένας *υπο-τύπος* (subtype) της κλάσης βάσης.

Η σχέση υπο-τύπου είναι πολύ σημαντική για τη γλώσσα Java γιατί επιτρέπει σε αντικείμενα να προσπελαύνονται μέσω αναφορών που ανήκουν σε διαφορετικούς τύπους. Αυτό είναι χρήσιμο π.χ. όταν ο τύπος κάποιου αντικειμένου δεν είναι απόλυτα γνωστός, αλλά είναι μόνο γνωστό ότι ανήκει σε μια κλάση που κληρονομεί κάποια άλλη. Αναφορές σε αντικείμενα που ανήκουν σε υπο-τύπους μιας κλάσης μπορούν να εκχωρηθούν σε μεταβλητές του τύπου αναφοράς της κλάσης βάσης. Τα ίδια τα αντικείμενα δεν υπόκεινται σε καμιά μεταβολή, είναι όμως ορατά σαν να ήταν αντικείμενα της κλάσης βάσης.

Στο παράδειγμα που ακολουθεί, γίνεται αναφορά σε ένα αντικείμενο της κλάσης `CounterDec` μέσω δυο μεταβλητών. Η πρώτη, `cd`, έχει τύπο `CounterDec` ενώ η δεύτερη, `cb`, έχει τύπο `Counter`.

```
CounterDec cd = new CounterDec();
Counter    cb = cd;

cd.inc();   // this is OK, CounterDec has method inc
cb.inc();   // so is this
```

```
cd.dec();   // this is OK, CounterDec has method dec
cb.dec();   // but this is wrong, compilation error!
```

Επίσης, με τον ίδιο τρόπο, αντικείμενα της κλάσης `CounterDec` μπορούν να περάσουν ως πραγματικές παράμετροι σε μεθόδους, των οποίων τα ορίσματα έχουν δηλωθεί ως τύπου `Counter`.

Μια αναφορά σε ένα αντικείμενο της παραγόμενης κλάσης, μπορεί λοιπόν να εκληφθεί ως αναφορά σε αντικείμενο της κλάσης βάσης. Η μετατροπή γίνεται με συνοπτική διαδικασία κατά τη μεταγλώττιση. Η αντίστροφη πορεία, δηλαδή από μια αναφορά σε αντικείμενο της κλάσης βάσης προς μια αναφορά σε αντικείμενο της παραγόμενης κλάσης, είναι δυνατό να γίνει μόνο υπό όρους. Για την ακρίβεια, μια τέτοια μετατροπή έχει νόημα μόνο αν η το αντικείμενο της αναφοράς είναι στην πραγματικότητα αντικείμενο της παραγόμενης κλάσης. Ο έλεγχος για να διαπιστωθεί κάτι τέτοιο δεν μπορεί να γίνει κατά τη μεταγλώττιση του προγράμματος, γίνεται όμως στο χρόνο εκτέλεσης. Για το λόγο αυτό, η μετατροπή γίνεται με ρητή δήλωση του τύπου της παραγόμενης κλάσης, μέσα στον κώδικα.

Στο παράδειγμα που ακολουθεί κατασκευάζονται δυο αντικείμενα. Το πρώτο προέρχεται από την κλάση `CounterDec` ενώ το δεύτερο από την κλάση `Counter`. Και στα δυο γίνεται αρχικά αναφορά από δυο μεταβλητές τύπου `Counter`.

```
Counter cb1 = new Counter();
Counter cb2 = new CounterDec();
CounterDec cd;

// this will be OK, cb1 refers to a CounterDec

cd = (CounterDec) cb1;

// this is a run-time error, cb2 refers to a Counter!

cd = (CounterDec) cb2;
```

Στη συνέχεια γίνεται μετατροπή των δυο αναφορών σε αντικείμενα της κλάσης `CounterDec`. Ο μεταγλωττιστής δε θα διαμαρτυρηθεί, γιατί αφού η κλάση `CounterDec` είναι επέκταση της `Counter`, είναι δυνατό να πρόκειται στην πραγματικότητα για τέτοια αντικείμενα. Στο χρόνο εκτέλεσης, η μετατροπή της πρώτης αναφοράς θα γίνει χωρίς πρόβλημα, γιατί στην πραγματικότητα επρόκειτο για αντικείμενο της κλάσης `CounterDec` που ήταν ορατό μέσω αναφοράς σε `Counter`. Η δεύτερη μετατροπή, όμως, θα οδηγήσει σε σφάλμα εκτέλεσης: το αντικείμενο δεν μπορεί να εκληφθεί ως `CounterDec` γιατί, πολύ απλά, δεν είναι τέτοιο.

3.3.3. Υπερκάλυψη μεταβλητών και υπερίσχυση μεθόδων

Η επέκταση κλάσεων με την προσθήκη νέων μελών είναι δυνατό να προκαλέσει προβλήματα. Τί θα συμβεί αν η παραγόμενη κλάση δηλώνει ένα μέλος που έχει το ίδιο όνομα με ένα μέλος της κλάσης βάσης; Ας υποτεθεί ότι η κλάση `UselessCounter` επεκτείνει την κλάση `Counter` ως εξής:

```
class UselessCounter extends Counter
{
   String value = "hello";
}
```

Μια νέα μεταβλητή προστίθεται, το όνομα της οποίας συμπίπτει με το όνομα ενός μέλους της κλάσης `Counter`, που κληρονομείται στην `UselessCounter`.

Όταν μια παραγόμενη κλάση ορίζει μια νέα μεταβλητή, το όνομα της οποίας συμπίπτει με το όνομα μιας κληρονομούμενης μεταβλητής, τότε η νέα μεταβλητή υπερκαλύπτει την παλιά. Και οι δυο μεταβλητές συνυπάρχουν στα αντικείμενα της παραγόμενης κλάσης, αλλά οι νέες μεταβλητές επισκιάζουν τις παλιές:[6]

```
UselessCounter u = new UselessCounter();
Counter        c = new Counter();

// this is OK, refering to the new variable

u.value = "look";

// this is also OK, refering to the old variable

c.value = 5;

// But this is wrong, compilation error!
// value refers to the old variable now!

c = u;
c.value = "hi";
```

Επίσης, η λειτουργικότητα της κλάσης `Counter` δε θα μεταβληθεί, γιατί οι μέθοδοι που ορίζονται σε αυτή δε βλέπουν τη νέα μεταβλητή. Γενικά η επισκίαση μεταβλητών στις παραγόμενες κλάσεις δεν είναι ιδιαίτερα χρήσιμη και είναι καλό να αποφεύγεται, αφού μπορεί εύκολα να οδηγήσει σε παρεξηγήσεις.

Ας θεωρήσουμε τώρα μια διαφορετική επέκταση της κλάσης `Counter`, λίγο πιο χρήσιμη αυτή τη φορά. Στην κλάση `DoubleCounter` η μέθοδος `inc` αυξάνει την τιμή του μετρητή κατά `2`, αντί κατά `1`.

```
class DoubleCounter extends Counter
{
   void inc () { value += 2; }
}
```

Όταν μια παραγόμενη κλάση ορίζει μια νέα μέθοδο, το όνομα της οποίας συμπίπτει με το όνομα μιας μεθόδου που κληρονομείται από την κλάση βάση, τότε υπάρχουν δυο περιπτώσεις. Αν η νέα μέθοδος διαφέρει από την παλιά στον αριθμό ή/και στον τύπο των παραμέτρων, τότε πρόκειται για απλή υπερφόρτωση μεθόδου, όπως περιγράφηκε σε προηγούμενη ενότητα. Αν όμως αυτό δε συμβαίνει, τότε η νέα μέθοδος υπερισχύει της παλιάς. Αυτό ονομάζεται *υπερίσχυση μεθόδου* (method overriding). Ας θεωρήσουμε το ακόλουθο τμήμα κώδικα:

```
Counter       c = new Counter();
DoubleCounter d = new DoubleCounter();

c.inc();   // value of c becomes 1, old inc
d.inc();   // value of d becomes 2, new inc
```

[6] Προσωρινά παραβλέπουμε το γεγονός ότι η μεταβλητή `value` της κλάσης `Counter` δεν είναι ορατή στο εξωτερικό της κλάσης.

Στην περίπτωση του αντικειμένου `c`, καλείται η μέθοδος inc της κλάσης `Counter` και η τιμή του μετρητή αυξάνει κατά `1`. Στην περίπτωση του αντικειμένου `d` όμως, η μέθοδος `inc` της κλάσης `DoubleCounter` υπερισχύει και η τιμή αυξάνει κατά `2`.

Είναι χαρακτηριστικό ότι αντίθετα με την υπερφόρτωση μεθόδων, η επίλυση της οποίας γίνεται κατά τη μεταγλώττιση του προγράμματος, η επίλυση της υπερίσχυσης μεθόδων γίνεται δυναμικά στο χρόνο εκτέλεσης. Αυτό γίνεται αντιληπτό με το ακόλουθο παράδειγμα:

```
Counter c = new Counter();
Counter d = new DoubleCounter();

c.inc();   // value of c becomes 1, old inc
d.inc();   // value of d becomes 2, new inc
```

Το παράδειγμα μοιάζει με το προηγούμενο, με τη μόνη διαφορά ότι και τα δυο αντικείμενα προσπελαύνονται μέσω αναφορών τύπου `Counter`. Παρόλα αυτά, στην περίπτωση του αντικειμένου `DoubleCounter`, η νέα μέθοδος `inc` υπερισχύει και πάλι της παλιάς και η τιμή του μετρητή αυξάνει κατά `2`.

Η υπερίσχυση μεθόδων είναι ένα πολύτιμο εφόδιο στη γλώσσα Java, καθώς επιτρέπει στις εξειδικευμένες κλάσεις να παρακάμψουν, μέχρι ενός σημείου, τη συμπεριφορά που τους επιβάλλεται από τις κλάσεις βάσεις.

Στην περίπτωση που χρειάζεται να κληθεί η μέθοδος της κλάσης βάσης, μέσα από τον κώδικα μεθόδου της παραγόμενης κλάσης, είναι δυνατό να χρησιμοποιηθεί η ειδική μεταβλητή `super`. Για παράδειγμα, ο ορισμός της μεθόδου `inc` της κλάσης `DoubleCounter` θα μπορούσε να είναι ισοδύναμα:

```
void inc () { super.inc(); super.inc(); }
```

3.3.4. Αφηρημένες και τελικές κλάσεις

Οποιαδήποτε μέθοδος στον ορισμό μιας κλάσης μπορεί να δηλωθεί χρησιμοποιώντας το μετατροπέα `abstract`. Στην περίπτωση αυτή, η μέθοδος ονομάζεται *αφηρημένη* (abstract). Οι αφηρημένες μέθοδοι δεν έχουν σώμα παρά μόνο επικεφαλίδα. Ουσιαστικά καθορίζεται μόνο ο τρόπος χρήσης και όχι η υλοποίησή τους. Κλάσεις που περιέχουν ή κληρονομούν αφηρημένες μεθόδους ονομάζονται επίσης αφηρημένες και πρέπει να ορίζονται με το μετατροπέα `abstract`.

Ας θεωρήσουμε το ακόλουθο παράδειγμα:

```
abstract class DisplayableCounter extends Counter
{
   abstract void display ();
}
```

Η κλάση `DisplayableCounter` περιγράφει ένα μετρητή που περιέχει μια μέθοδο για την εμφάνιση της τιμής του στο χρήστη. Η μέθοδος αυτή είναι αφηρημένη και κατά συνέπεια ολόκληρη η κλάση είναι αφηρημένη. Η κατασκευή αντικειμένων που προέρχονται από μια αφηρημένη κλάση είναι φυσικά αδύνατη, καθώς εκρεμεί η υλοποίηση κάποιων μεθόδων. Κατά συνέπεια, το ακόλουθο τμήμα κώδικα οδηγεί σε σφάλμα μεταγλώττισης:

```
c = new DisplayableCounter();   // compilation error!
```

Ένας πιθανός λόγος που η εμφάνιση της τιμής δεν υλοποιείται αμέσως είναι ότι υπάρχουν πολλοί διαφορετικοί τρόποι για την υλοποίησή της. Για παράδειγμα, αν το λειτουργικό σύστημα και η εφαρμογή που αναπτύσσεται επιτρέπει τη χρήση παραθύρων με γραφικά, η τιμή θα μπορούσε να εμφανίζεται μέσα σε ένα νέο παράθυρο, με κατάλληλο μήνυμα. Ας προχωρήσουμε όμως σε μια απλούστερη υλοποίηση, με την οποία η τιμή του μετρητή απλώς εκτυπώνεται στην οθόνη:

```
class SimpleDCounter extends DisplayableCounter
{
   void display () { System.out.println(get()); }
}
```

Στην κλάση `SimpleDCounter` καθορίζεται μια υλοποίηση για τη μέθοδο `display` και η κλάση παύει να είναι αφηρημένη. Στη συνέχεια μπορεί να χρησιμοποιηθεί:

```
SimpleDCounter c = new SimpleDCounter();

if (c.get() < 100)
   c.display();
```

Η χρήση μπορεί να γίνει επίσης μέσω αναφοράς σε τύπο `DisplayableCounter`:

```
DisplayableCounter c = new SimpleDCounter();

c.display();
```

Μερικές φορές είναι επιθυμητό να εξασφαλισθεί ότι μια κλάση δε θα εξειδικευθεί περισσότερο. Η Java επιτρέπει τη δήλωση μιας κλάσης ή επιμέρους μελών της με τον μετατροπέα `final`, απαγορεύοντας έτσι την περαιτέρω εξειδίκευσή της. Μια κλάση που έχει δηλωθεί ως `final` είναι αδύνατο να επεκταθεί. Μια μέθοδος που έχει δηλωθεί ως final είναι αδύνατο να επανορισθεί σε παραγόμενη κλάση. Επίσης, το ίδιο μπορεί να συμβεί και με μεταβλητές μιας κλάσης. Αν μια μεταβλητή δηλωθεί ως `final`, τότε η τιμή της δεν μπορεί να μεταβληθεί, πρόκειται επομένως για σταθερά. Ας θεωρήσουμε το ακόλουθο παράδειγμα:

```
final class LimitedCounter extends Counter
{
   static final int LIMIT = 100;

   void inc () { if (value < LIMIT) value++; }
}
```

Η κλάση `LimitedCounter` δηλώνεται ως `final` και κατά συνέπεια η παρακάτω δήλωση οδηγεί σε σφάλμα μεταγλώττισης:

```
class DoubleLCounter extends LimitedCounter   // wrong!
{
   void inc () { super.inc(); super.inc(); }
}
```

Επίσης, η μεταβλητή `LIMIT` ορίζεται ως `final`, κατά συνέπεια η τιμή της δεν μπορεί να μεταβληθεί και το παρακάτω τμήμα κώδικα οδηγεί πάλι σε σφάλμα μεταγλώττισης:

```
LimitedCounter.LIMIT = 1000;   // wrong!
```

3.4. Διαπροσωπείες

Μια *διαπροσωπεία* (interface) στη Java μπορεί να θεωρηθεί ως μια αφηρημένη κλάση, η οποία δε δηλώνει μεταβλητές και όλες οι μέθοδοί της είναι αφηρημένες. Οι διαπροσωπείες, όπως και οι κλάσεις, ορίζουν ιεραρχίες κληρονομικότητας και είναι δυνατό να δηλωθούν μεταβλητές με τύπο αναφοράς σε αυτές. Με την έννοια αυτή, είναι ισοδύναμες με πλήρως αφηρημένες κλάσεις.

Ας θεωρήσουμε την ακόλουθη διαπροσωπεία, η οποία περιγράφει αντικείμενα που διαθέτουν μια μέθοδο εμφάνισης, με το όνομα `display`.

```
interface Displayable
{
   void display ();
}
```

Στη συνέχεια, ας θεωρήσουμε την κλάση `SimpleDCounter`, που ορίζεται ως:

```
class SimpleDCounter extends Counter
                     implements Displayable
{
   void display () { System.out.println(get()); }
}
```

Η κλάση αυτή υλοποιεί μια μέθοδο display, με την ίδια επικεφαλίδα που ορίζεται στη διαπροσωπεία `Displayable`. Είναι επομένως δυνατό να θεωρηθεί ως μια υλοποίηση της διαπροσωπείας και αυτό φαίνεται στον ορισμό της με τη δήλωση `implements Displayable`. Στη συνέχεια, τα αντικείμενα της κλάσης μπορούν να χρησιμοποιηθούν μέσω αναφορών τύπου `Displayable`, όπως φαίνεται στο ακόλουθο τμήμα κώδικα:

```
SimpleDCounter c = new SimpleDCounter();
Displayable    d = c;

d.display();
```

Οι ιεραρχίες των διαπροσωπειών μπορούν να θεωρηθούν ως ορθογώνιες προς την ιεραρχία των κλάσεων. Οι διαπροσωπείες καλύπτουν την ανάγκη για πολλαπλή κληρονομικότητα, δηλαδή την επέκταση περισσότερων της μιας κλάσης, που δεν υποστηρίζεται από τη Java. Μια κλάση μπορεί να επεκτείνει μόνο μια κλάση βάση, μπορεί όμως να υλοποιεί οποιονδήποτε αριθμό από διαπροσωπείες. Για παράδειγμα:

```
interface Incrementable
{
   void inc ();
}

SimpleDCounter extends Counter
               implements Displayable, Incrementable
{
   void display () { System.out.println(get()); }
}
```

Η κλάση `SimpleDCounter` τώρα υλοποιεί δυο διαπροσωπείες. Η μέθοδος `inc` που απαιτείται από τη διαπροσωπεία `Incrementable` κληρονομείται στην κλάση `SimpleDCounter` από την `Counter`.

Όπως αναφέρθηκε στην αρχή, οι διαπροσωπείες δεν είναι δυνατό να περιέχουν μεταβλητές. Μοναδική εξαίρεση είναι οι σταθερές, δηλαδή μεταβλητές που δηλώνονται ως `static final`.

3.5. Πακέτα

Τα *πακέτα* (packages) στη Java παρέχουν ένα μηχανισμό οργάνωσης των κλάσεων. Κάθε πακέτο περιέχει έναν αριθμό κλάσεων και χαρακτηρίζεται το όνομά του. Οι ορισμοί αυτών των κλάσεων μπορούν να περιέχονται σε ένα ή περισσότερα αρχεία, που ονομάζονται *μονάδες μεταγλώττισης* (compilation units). Τα ονόματα των πακέτων απαρτίζονται από μια ή περισσότερες λέξεις, χωρισμένες με τελείες. Για παράδειγμα, ονόματα πακέτων που χρησιμοποιούνται συχνά στην πράξη είναι τα `java.lang` και `java.io`.

Προκειμένου να χρησιμοποιηθούν τα περιεχόμενα ενός πακέτου, πρέπει να προηγείται το όνομα του πακέτου. Το ακόλουθο τμήμα κώδικα χρησιμοποιεί την κλάση `Stack` που βρίσκεται στο πακέτο `java.util`:

```
java.util.Stack s = new java.util.Stack();
MyObject m = new MyObject();

s.push(m);
```

Φυσικά, τα μεγάλα ονόματα γίνονται κουραστικά μετά από λίγο. Για το λόγο αυτό τα περιεχόμενα ενός πακέτου μπορούν να “εισαχθούν” στον κώδικα, ώστε να μην απαιτείται η χρήση του ονόματος του πακέτου. Αυτό γίνεται με μια δήλωση `import`, που πρέπει να βρίσκεται στην αρχή του αρχείου:

```
import java.util.Stack;

...

Stack s = new Stack();
...
```

Η παραπάνω δήλωση import εισάγει την κλάση `Stack` από το πακέτο `java.util`. Η δήλωση έχει πληροφοριακό χαρακτήρα, δηλαδή ο κώδικας της κλάσης `Stack` δεν εισάγεται στην πραγματικότητα και το μέγεθος του μεταγλωττισμένου αρχείου δε μεταβάλλεται. Με παρόμοιο τρόπο, μπορούν να εισαχθούν όλα τα περιεχόμενα του πακέτου `java.util`, χρησιμοποιώντας τον ειδικό χαρακτήρα `*` ως εξής:

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

Αν απαιτείται η εισαγωγή περισσότερων πακέτων, είναι δυνατό να χρησιμοποιηθούν περισσότερες δηλώσεις `import`, πάντα στην αρχή του αρχείου. Στην περίπτωση που από διαφορετικά πακέτα εισάγονται κλάσεις με το ίδιο όνομα, οι κλάσεις αυτές δεν είναι δυνατό να προσπελασθούν απλώς με το όνομά τους, αλλά απαιτείται χρήση του πλήρους ονόματος.

3.5.1. Δημιουργία πακέτων

Εκτός από το να χρησιμοποιούν υπάρχοντα πακέτα, η Java επιτρέπει στους προγραμματιστές να δημιουργούν τα δικά τους πακέτα, στα οποία να οργανώνουν τις δικές τους κλάσεις. Για να γίνει κάτι τέτοιο, πρέπει στη γραμμή της μονάδας μεταγλώττισης να υπάρχει μια δήλωση `package`, που καθορίζει το όνομα του πακέτου, στο οποίο θα συμπεριληφθούν οι κλάσεις και τα λοιπά στοιχεία που δηλώνονται στη συνέχεια.

Για παράδειγμα, ας θεωρήσουμε το ακόλουθο πακέτο με όνομα `mine.counters`, που συμπεριλαμβάνει τις κλάσεις των μετρητών. Προκειμένου να είναι σε θέση να χρησιμοποιηθεί, το πακέτο αυτό πρέπει πρώτα να μεταγλωττισθεί. Στη συνέχεια, τα αρχεία `.class` που θα προκύψουν πρέπει να τοποθετηθούν σε ένα directory με όνομα `counters`, που θα βρίσκεται κάτω από ένα directory με όνομα `mine`, κάπου στο μονοπάτι των πακέτων της Java.

```
package mine.counters;

class Counter { ... }

class CounterDec extends Counter { ... }

...

interface Displayable { ... }

class SimpleDCounter extends Counter
                     implements Displayable { ... }
```

Όλες οι δηλώσεις κάθε αρχείου Java αποτελούν τμήμα κάποιου πακέτου. Αν η μονάδα μεταγλώττισης δεν αρχίζει με μια δήλωση `package`, τότε οι δηλώσεις τοποθετούνται σε ένα γενικό ανώνυμο πακέτο, το οποίο είναι πάντα διαθέσιμο.

3.5.2. Πρόσβαση σε μέλη κλάσεων

Αφού περιγράφηκε ο μηχανισμός οργάνωσης κλάσεων σε πακέτα, είναι πάλι σκόπιμο να αναφερθούμε στην πρόσβαση στα μέλη των κλάσεων και στους μετατροπείς ορατότητας, καθώς η εικόνα που δόθηκε σε προηγούμενη ενότητα δεν ήταν απόλυτα ακριβής. Στη Java, οι κλάσεις που ανήκουν στο ίδιο πακέτο θεωρείται ότι αποτελούν μια ενότητα. Για το λόγο αυτό, η έννοια του πακέτου έχει μεγάλη σημασία όσον αφορά στην ορατότητα των μελών των κλάσεων. Μεγάλη σημασία έχει επίσης η έννοια της κληρονομικότητας, καθώς οι παραγόμενες κλάσεις ορισμένες φορές χρειάζεται να έχουν πρόσβαση στην υλοποίηση των κλάσεων βάσεων.

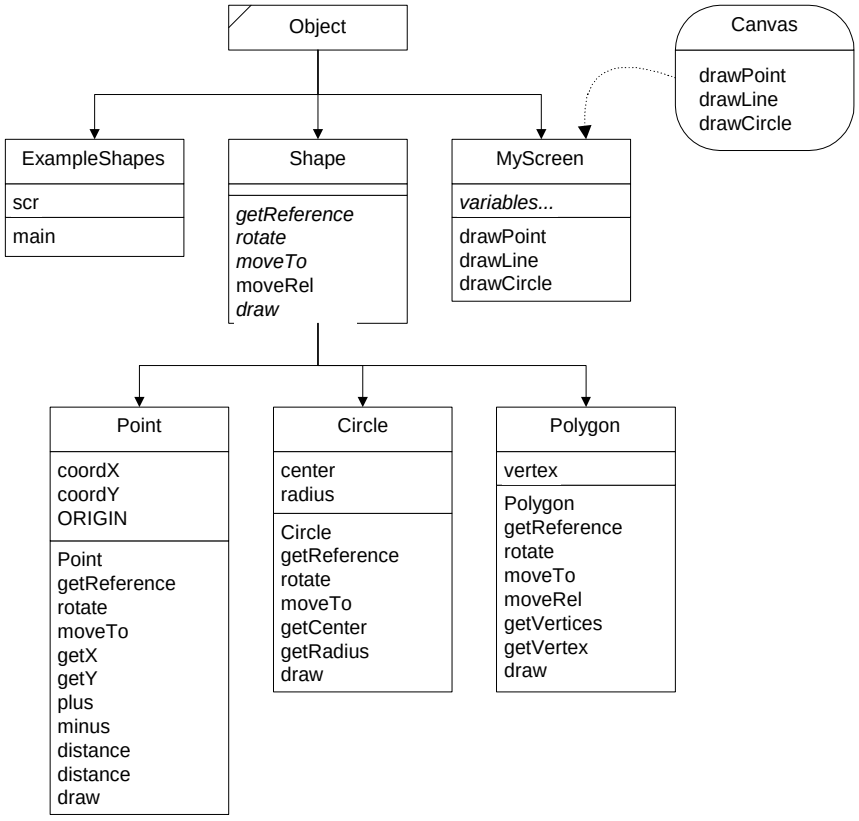
Η Java υποστηρίζει τέσσερεις μετατροπείς ορατότητας για τα μέλη των κλάσεων που, μαζί με την απουσία μετατροπέα, αποτελούν πέντε διαφορετικούς τρόπους για να δηλωθεί ένα μέλος. Η λειτουργία των μετατροπέων περιγράφεται στη συνέχεια, από τον περισσότερο προς τον λιγότερο περιοριστικό. Τα μέλη μιας κλάσης είναι *πάντοτε* ορατά μέσα από τις μεθόδους της ίδιας κλάσης και το γεγονός αυτό δεν μπορεί να μεταβληθεί από κανένα μετατροπέα ορατότητας. Για το λόγο αυτό, δεν αναφέρεται ρητά στην παρακάτω περιγραφή.

- **private**: τα μέλη μιας κλάσης που δηλώνονται με το μετατροπέα private δεν είναι πουθενά ορατά έξω από την κλάση.

- **private protected**: τα μέλη που δηλώνονται με χρήση των δυο μετατροπέων private και protected είναι ορατά μόνο από παραγόμενες κλάσεις, είτε αυτές βρίσκονται στο ίδιο είτε σε άλλα πακέτα. Αυτός είναι ο μοναδικός επιτρεπτός συνδυασμός μετατροπέων ορατότητας.
- ***απουσία μετατροπέα***: τα μέλη που δηλώνονται απουσία μετατροπέα είναι ορατά μόνο από τις κλάσεις που ανήκουν στο ίδιο πακέτο.
- **protected**: τα μέλη που δηλώνονται με το μετατροπέα protected είναι ορατά από τις κλάσεις που ανήκουν στο ίδιο πακέτο και επιπλέον από παραγόμενες κλάσεις εκτός πακέτου.
- **public**: τα μέλη που δηλώνονται με το μετατροπέα public είναι ορατά από όλες τις κλάσεις, ανεξαρτήτως πακέτου. Ο μετατροπέας αυτός είναι ιδιαίτερα χρήσιμος για τη διαπροσωπεία μιας κλάσης.

Στο σημείο αυτό αξίζει να γίνουν δυο παρατηρήσεις, σχετικές με την ορατότητα μελών και την κληρονομικότητα. Η πρώτη παρατήρηση αφορά στην υπερίσχυση μεθόδων: δεν είναι δυνατό να περιορισθεί η ορατότητα μιας μεθόδου της παραγόμενης κλάσης που υπερισχύει μιας μεθόδου της κλάσης βάσης. Το αντίστροφο όμως είναι δυνατό. Η δεύτερη παρατήρηση αφορά στη χρήση του μετατροπέα `protected`: τα μέλη της κλάσης βάσης είναι ορατά από τις παραγόμενες κλάσεις μόνο όταν πρόκειται για αντικείμενα που προέρχονται από τις τελευταίες.

3.6. Παράδειγμα: Βιβλιοθήκη σχημάτων



Σχήμα 3.2. Ιεραρχία κλάσεων σχημάτων.

Στο παράδειγμα που ακολουθεί υλοποιείται μια μικρή βιβλιοθήκη σχημάτων στις δυο διαστάσεις. Η ιεραρχία των κλάσεων που ορίζονται απεικονίζεται στο Σχήμα 3.2, καθώς και οι μέθοδοι κάθε κλάσης. Οι κύριες κλάσεις που υλοποιούν τη βιβλιοθήκη είναι οι `Shape`, `Point`, `Circle` και `Polygon`. Η διαπροσωπεία `Canvas` περιγράφει αντικείμενα που διαθέτουν μεθόδους σχεδίασης, και η κλάση `MyScreen` είναι μια υποθετική υλοποίηση αυτής της διαπροσωπείας. Οι ορισμοί όλων αυτών των στοιχείων γίνονται σε ένα νέο πακέτο με όνομα `shapes`. Επίσης, για τις ανάγκες του παραδείγματος υλοποιείται η κλάση `ExampleShapes`, που χρησιμοποιεί τη βιβλιοθήκη. Ο κώδικας για όλα αυτά τα στοιχεία δίνεται στις επόμενες παραγράφους.

Κλάση Shape

Η αφηρημένη αυτή κλάση υλοποιεί ένα σχήμα στο χώρο δυο διαστάσεων. Τα σχήματα, όπως συνηθίζεται με ορισμένες κατηγορίες αντικειμένων στη Java, δεν είναι δυνατό να μεταβληθούν μετά την κατασκευή τους, αλλά οι μέθοδοι επεξεργασίας επιστρέφουν ως αποτέλεσμα νέα σχήματα. Οι μέθοδοι που ορίζονται στην κλάση `Shape` είναι οι ακόλουθες. Από αυτές, μόνο η μέθοδος `moveRel` υλοποιείται, μέσω της `moveTo` και των μεθόδων της κλάσης `Point`.

- `Point` **`getReference`** `()`

Επιστρέφει το σημείο αναφοράς του σχήματος, που συνήθως είναι κάποιο από τα χαρακτηριστικά του σημεία (π.χ. στον κύκλο είναι το κέντρο του).

- `Shape` **`rotate`** `(Point origin, double angle)`

Περιστρέφει το σχήμα κατά γωνία `angle` (σε ακτίνια) γύρω από το σημείο `origin`.

- `Shape` **`moveTo`** `(Point target)`

Μετακινεί το σχήμα ώστε το σημείο αναφοράς του να βρεθεί στο `target`.

- `Shape` **`moveRel`** `(Point displacement)`

Μετακινεί το σχήμα κατά σχετική απόσταση `displacement` από την τρέχουσα θέση του.

- `void` **`draw`** `(Canvas c)`

Σχεδιάζει το σχήμα στο αντικείμενο με δυνατότητες σχεδίασης `c`.

Ο κώδικας της κλάσης ακολουθεί.

```
package shapes;

abstract public class Shape
{
   abstract public Point getReference ();
   abstract public Shape rotate (Point origin, double angle);
   abstract public Shape moveTo (Point target);

   public Shape moveRel (Point displacement)
   {
      return moveTo(getReference().plus(displacement));
   }

   abstract static public Shape draw (Canvas c);
}
```

Κλάση Point

Η κλάση αυτή κληρονομεί την `Shape` και υλοποιεί ένα σημείο. Τα επιπλέον μέλη που ορίζονται είναι τα εξής:

- `double` **`coordX`**, `double` **`coordY`**

Οι συντεταγμένες του σημείου.

- **`Point`** `(double x, double y)`

Κατασκευαστής σημείου με συντεταγμένες (*x*, *y*).

- `double` **`getX`** `()`, `double` **`getY`** `()`

Συναρτήσεις που επιστρέφουν τις συντεταγμένες του σημείου.

- `Point` **`plus`** `(Point p), Point` **`minus`** `(Point p)`

Το διανυσματικό άθροισμα και διαφορά αντίστοιχα του τρέχοντος σημείου και του σημείου `p`.

- `double` **`distance`** `(), double` **`distance`** `(Point p)`

Η απόσταση του τρέχοντος σημείου από την αρχή των αξόνων ή από το σημείο `p`, αντίστοιχα.

- `Point` **`ORIGIN`**

Ένα σταθερό σημείο που περιγράφει την αρχή των αξόνων.

Οι αφηρημένες μέθοδοι της κλάσης `Shape` υλοποιούνται με κατάλληλο τρόπο. Το σημείο αναφοράς που επιστρέφεται από την `getReference` για ένα αντικείμενο τύπου `Point` είναι προφανώς το ίδιο το σημείο. Ο κώδικας της κλάσης ακολουθεί. Γίνεται χρήση των μαθηματικών συναρτήσεων της κλάσης `Math`, που θα περιγραφούν σε επόμενο κεφάλαιο.

```
package shapes;

final public class Point extends Shape
{
   protected double coordX, coordY;

   final static public Point ORIGIN = new Point(0.0, 0.0);

   public Point (double x, double y) { coordX = x; coordY = y; }

   public Point getReference () { return this; }

   public Shape rotate (Point origin, double angle)
   {
      Point  delta  = minus(origin);
      double radius = delta.distance();
      double arg    = Math.atan2(delta.coordY,
                                 delta.coordX);

      return new Point(
       origin.coordX + radius * Math.cos(arg + angle),
       origin.coordY + radius * Math.sin(arg + angle));
   }

   public Shape moveTo (Point target) { return target; }

   public double getX () { return coordX; }
   public double getY () { return coordY; }

   public Point plus (Point p)
   {
      return new Point(coordX + p.coordX, coordY + p.coordY);
   }

   public Point minus (Point p)
   {
      return new Point(coordX - p.coordX, coordY - p.coordY);
   }

   public double distance ()
   {
      return Math.sqrt(coordX * coordX + coordY * coordY);
   }

   public double distance (Point p)
```

```
    {
       return minus(p).distance();
    }

    static public void draw (Canvas c)
    {
       c.drawPoint(coordX, coordY);
    }
}
```

Κλάση Circle

Η κλάση `Circle` είναι η υποκλάση της `Shape` που υλοποιεί έναν κύκλο. Τα επιπλέον μέλη που ορίζονται στην `Circle` είναι τα εξής:

- `Point` **`center`**, `double` **`radius`**

Το κέντρο και η ακτίνα του κύκλου αντίστοιχα.

- **`Circle`** `(Point c, double r)`

Κατασκευαστής κύκλου με κέντρο `c` και ακτίνα `r`.

- `Point` **`getCenter`** `(), double` **`getRadius`** `()`

Συναρτήσεις που επιστρέφουν το κέντρο και την ακτίνα του κύκλου.

Το σημείο αναφοράς του κύκλου είναι προφανώς το κέντρο του. Οι υπόλοιπες αφηρημένες μέθοδοι της κλάσης `Shape` υλοποιούνται με κατάλληλο τρόπο. Ο κώδικας της κλάσης `Circle` ακολουθεί.

```
package shapes;

final public class Circle extends Shape
{
   protected Point center;
   protected double radius;

   public Circle (Point c, double r) { center = c; radius = r; }

   public Point getReference () { return center; }

   public Shape rotate (Point origin, double angle)
   {
      Point newCenter = (Point) center.rotate(origin, angle);

      return new Circle(newCenter, radius);
   }

   public Shape moveTo (Point target)
   {
      return new Circle(target, radius);
   }

   public Point  getCenter () { return center; }
   public double getRadius () { return radius; }

   static public void draw (Canvas c)
   {
```

```
            c.drawCircle(center.getX(), center.getY(), radius);
        }
    }
```

Κλάση Polygon

Η κλάση αυτή υλοποιεί πολύγωνα και φυσικά κληρονομεί την κλάση `Shape`. Οι κορυφές των πολυγώνων ορίζονται ως ένας πίνακας σημείων. Τα επιπλέον μέλη της κλάσης `Polygon` είναι τα ακόλουθα:

- `Point []` **`vertex`**

Ο πίνακας των σημείων που αποτελούν τις κορυφές του πολυγώνου.

- **`Polygon`** `(Point [] v)`

Κατασκευαστής πολυγώνου με κορυφές `v`.

- `int` **`getVertices`** `()`

Επιστρέφει τον αριθμό των κορυφών του πολυγώνου.

- `Point` **`getVertex`** `(int i)`

Επιστρέφει την `i`-οστή κορυφή του πολυγώνου.

Το σημείο αναφοράς ενός πολυγώνου είναι το κέντρο βάρους του. Οι υπόλοιπες αφηρημένες μέθοδοι της κλάσης `Shape` υλοποιούνται με κατάλληλο τρόπο. Ο κώδικας της κλάσης `Polygon` ακολουθεί:

```
package shapes;

public class Polygon extends Shape
{
   protected Point [] vertex;

   public Polygon (Point [] v) { vertex = v; }

   public Point getReference ()
   {
      double sumX = 0.0;
      double sumY = 0.0;

      for (int i = 0 ; i < vertex.length ; i++) {
         sumX += vertex[i].coordX;
         sumY += vertex[i].coordY;
      }
      return new Point(sumX/vertex.length, sumY/vertex.length);
   }

   public Shape rotate (Point origin, double angle)
   {
      Point [] newVertex = new Point [vertex.length];

      for (int i = 0 ; i < vertex.length ; i++)
         newVertex[i] = (Point) vertex[i].rotate(origin, angle);
      return new Polygon(newVertex);
```

```
    }

    public Shape moveTo (Point target)
    {
       return moveRel(target.minus(getReference()));
    }

    public Shape moveRel (Point displacement)
    {
       Point [] newVertex = new Point [vertex.length];

       for (int i = 0 ; i < vertex.length ; i++)
          newVertex[i] = vertex[i].plus(displacement);
       return new Polygon(newVertex);
    }

    public int getVertices () { return vertex.length; }
    public Point getVertex (int i) { return vertex[i]; }

    static public void draw (Canvas c)
    {
       double x1 = vertex[0].getX();
       double y1 = vertex[0].getY();

       for (int i = 1 ; i < vertex.length ; i++) {
          double x2 = vertex[i].getX(), y2 = vertex[i].getY();

          c.drawLine(x1, y1, x2, y2);
          x1 = x2; y1 = y2;
       }
       c.drawLine(x1, y1, vertex[0].getX(), vertex[0].getY());
    }
}
```

Διαπροσωπεία Canvas

Η διαπροσωπεία αυτή περιγράφει αντικείμενα που έχουν τη δυνατότητα σχεδίασης. Ορίζει τρεις μεθόδους, για τη σχεδίαση σημείων, γραμμών και κύκλων:

- `void` **`drawPoint`** `(double x, double y)`

Σχεδιάζει ένα σημείο με συντεταγμένες (x, y).

- `void` **`drawLine`** `(double x1, double y1,`
  `double x2, double y2)`

Σχεδιάζει το ευθύγραμμο τμήμα που ενώνει τα σημεία (x_1, y_1) και (x_2, y_2).

- `void` **`drawCircle`** `(double x, double y, double r)`

Σχεδιάζει έναν κύκλο με κέντρο (x, y) και ακτίνα r.

Ο κώδικας της διαπροσωπείας ακολουθεί:

```
package shapes;

interface Canvas
{
   public void drawPoint  (double x, double y);
```

```
   public void drawLine   (double x1, double y1,
                           double x2, double y2);
   public void drawCircle (double x, double y, double r);
}
```

Κλάση MyScreen

Η κλάση αυτή είναι μια υποθετική υλοποίηση της διαπροσωπείας `Canvas`. Η περιγραφή της υλοποίησης των μεθόδων της παραλείπεται στον κώδικα που ακολουθεί.

```
package shapes;

class MyScreen implements Canvas
{
   /* Various other things... */

   public void drawPoint (double x, double y)
   { /* Forget about the implementation... */ }

   public void drawLine (double x1, double y1,
                         double x2, double y2)
   { /* Forget about the implementation... */ }


   public void drawCircle (double x, double y, double r)
   { /* Forget about the implementation... */ }
}
```

Κλάση ExampleShapes

Η κλάση αυτή παρέχει τη μέθοδο `main` που απαιτείται για την εκτέλεση του παραδείγματος. Στη μέθοδο αυτή ορίζονται δυο σχήματα, ένα τρίγωνο και ένας κύκλος, και στη συνέχεια σχεδιάζονται, αφού περιστραφούν. Ο κώδικας της κλάσης ακολουθεί.

```
import shapes.*;

class ExampleShapes
{
   MyScreen scr = new MyScreen();

   static public void main (String [] args)
   {
      Point [] v1 = { new Point(1.0, 1.0),
                      new Point(3.0, 1.0),
                      new Point(2.0, 2.0) }
      Shape s1 = new Polygon(v1);
      Shape s2 = new Circle(new Point(2.0, 2.0), 1.0);

      s1.rotate(Point.ORIGIN, Math.PI / 2).draw(scr);
      s2 = s2.moveRel(-1.0, -2.0).draw(scr);
   }
}
```

4. Εξαιρέσεις και νήματα

Στο κεφάλαιο αυτό περιγράφονται δυο από τα πιο προηγμένα χαρακτηριστικά της γλώσσας Java: ο χειρισμός εξαιρέσεων και η υποστήριξη πολλαπλών νημάτων εκτέλεσης. Τα χαρακτηριστικά αυτά υποστηρίζονται τόσο από τη Java ως γλώσσα προγραμματισμού, όσο και από το API, μέσω κλάσεων που ορίζονται στο java.lang και σε άλλα πακέτα.

4.1. Χειρισμός εξαιρέσεων

Η γλώσσα Java προορίζεται πρωτίστως για εφαρμογές με μεγάλες απαιτήσεις σωστής συμπεριφοράς, ασφάλειας και μεταφερσιμότητας. Για το λόγο αυτό είναι πολύ σημαντικό να γίνεται σωστός χειρισμός των σφαλμάτων, που μπορούν να προκύψουν κατά την εκτέλεση των προγραμμάτων. Πολλά είδη σφαλμάτων εκτέλεσης είναι δυνατό να προληφθούν, με κατάλληλους ελέγχους που πραγματοποιούνται κατά τη μεταγλώττιση του προγράμματος. Ο μεταγλωττιστής, όμως, δεν είναι δυνατό να εξασφαλίσει την απουσία σφαλμάτων εκτέλεσης.

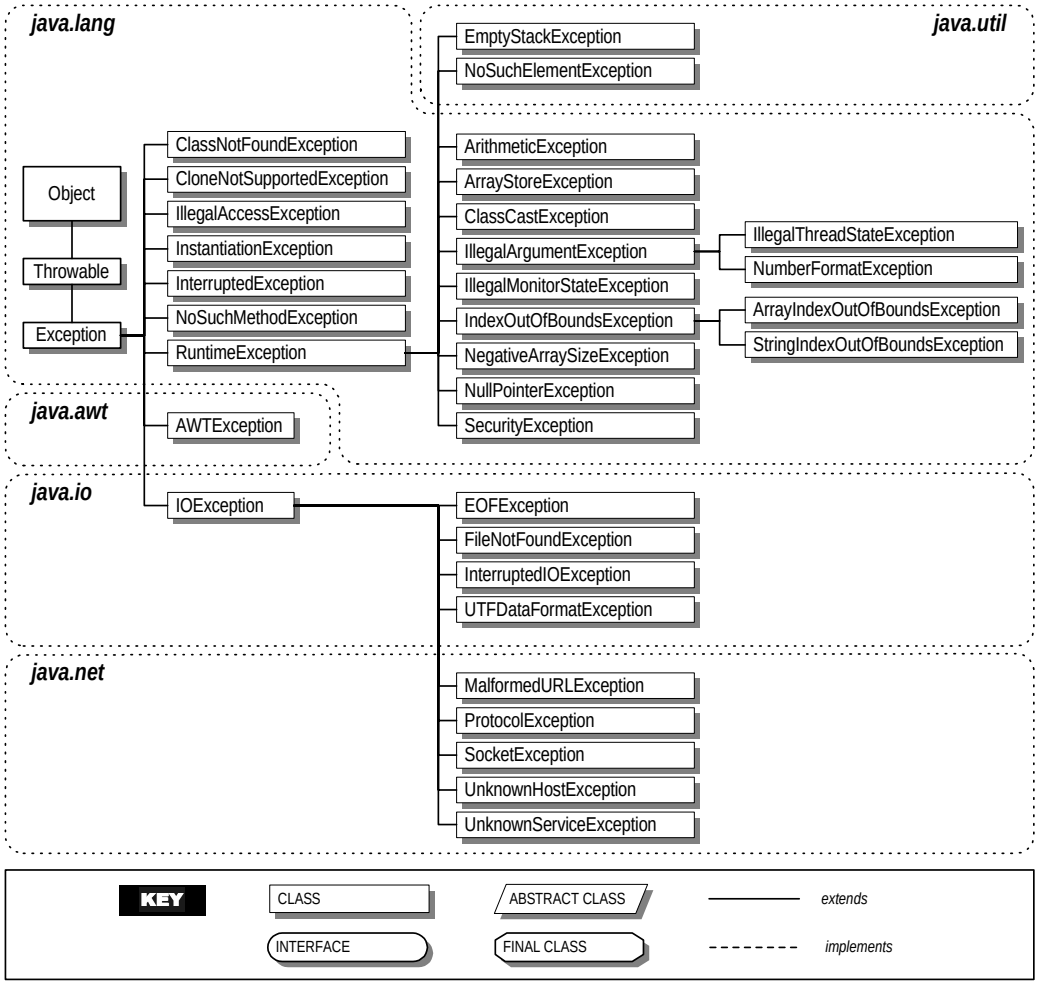
Σε μια γλώσσα προγραμματισμού όπως η C, η διαχείριση των σφαλμάτων εκτέλεσης είναι αποκλειστική αρμοδιότητα του προγραμματιστή, ο οποίος υποχρεώνεται να αναζητήσει τα σφάλματα, ενσωματώνοντας ελέγχους στο πρόγραμμα, και να προσδιορίσει τί θα συμβεί στην περίπτωση που αυτά εντοπισθούν. Αντίθετα, η Java παρέχει ενσωματωμένη διαχείριση των σφαλμάτων εκτέλεσης, μέσω του μηχανισμού των *εξαιρέσεων* (exceptions). Εκτός από την περίπτωση των σφαλμάτων εκτέλεσης, ο μηχανισμός των εξαιρέσεων χρησιμεύει για τη διαχείριση ειδικών καταστάσεων, που ο προγραμματιστής αυθαίρετα θεωρεί ότι παρεκλίνουν από τη σωστή συμπεριφορά του προγράμματος.

4.1.1. Κλάσεις εξαιρέσεων και σφαλμάτων

Η Java διακρίνει δυο κύριες κατηγορίες περιπτώσεων, στις οποίες η εκτέλεση του προγράμματος δεν μπορεί να συνεχισθεί φυσιολογικά: τις *εξαιρέσεις* (exceptions) και τα *σφάλματα* (errors). Η διαχείριση και των δυο αυτών κατηγοριών γίνεται με τον ίδιο τρόπο, όπως θα περιγραφεί στις επόμενες ενότητες.

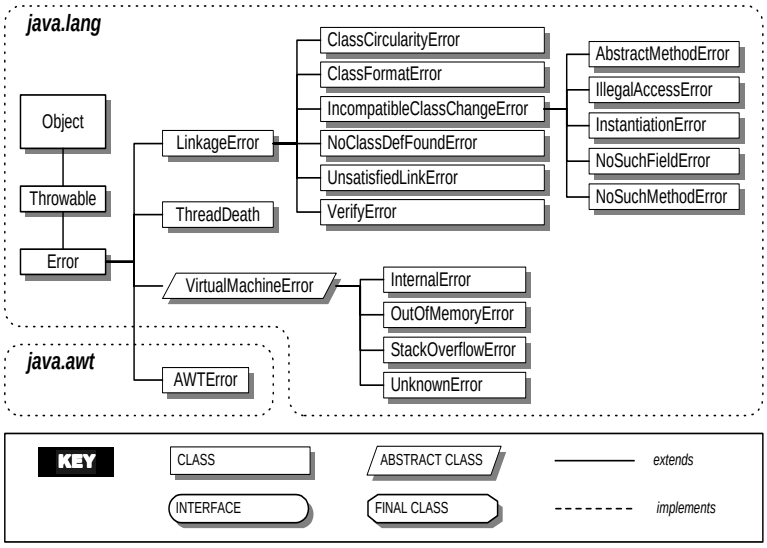
Η πρώτη κατηγορία αφορά εξαιρέσεις, δηλαδή ειδικές περιπτώσεις σφαλμάτων εκτέλεσης που μπορούν να ξεπερασθούν αν υπάρχει κατάλληλη πρόβλεψη και χειρισμός του προγραμματιστή. Οι εξαιρέσεις είναι αντικείμενα που προέρχονται από την κλάση `Exception`, ή υποκλάσεις

αυτής. Το Σχήμα 4.1 απεικονίζει την ιεραρχία των εξαιρέσεων που ορίζονται στο API της Java, και την κατανομή τους στα διάφορα πακέτα.



Σχήμα 4.1. Ιεραρχία κλάσεων εξαιρέσεων.

Η δεύτερη κατηγορία αφορά σφάλματα εκτέλεσης από τα οποία ο διερμηνέας της Java δεν μπορεί να ανανήψει. Τα σφάλματα αυτά είναι αντικείμενα που προέρχονται από την κλάση `Error`, ή υποκλάσεις αυτής. Ο προγραμματιστής συνήθως δε χρειάζεται να ασχοληθεί με την πρόβλεψη των σφαλμάτων αυτού του είδους. Σε περίπτωση που συμβούν, ο διερμηνέας διακόπτει την εκτέλεση του προγράμματος, εκτυπώνοντας κάποιο κατάλληλο διαγνωστικό μήνυμα. Η ιεραρχία των κλάσεων σφαλμάτων απεικονίζεται στο Σχήμα 4.2.



Σχήμα 4.2. Ιεραρχία κλάσεων σφαλμάτων.

4.1.2. Εντολές χειρισμού εξαιρέσεων

Οι εντολές `try` και `catch` της Java χρησιμεύουν για το χειρισμό των εξαιρέσεων. Όταν η ειδική περίπτωση που περιγράφεται από κάποια κλάση εξαίρεσης συμβαίνει, κατά την εκτέλεση του προγράμματος, τότε λέμε ότι *προκαλείται* η εξαίρεση (throw an exception). Αν ο προγραμματιστής έχει προβλέψει αυτή την εξαίρεση και τη χειρίζεται με κάποιο τρόπο, λέμε ότι η εξαίρεση *συλλαμβάνεται* (catch an exception).

Αν ο προγραμματιστής προβλέπει ότι ένα τμήμα κώδικα είναι πιθανό να προκαλέσει κάποια εξαίρεση και θέλει να τη συλλάβει, αρκεί να ορίσει το τμήμα κώδικα στο τμήμα `try` μιας δομής `try` / `catch`, όπως φαίνεται στο παράδειγμα που ακολουθεί:

```
try {
   ...
   readFromFile(file);
   ...
}
catch (Exception e) {
   // Do something about it!

   System.err.println("Sorry, cannot read!");
}
```

Στο παραπάνω παράδειγμα, ο προγραμματιστής προβλέπει ότι το τμήμα κώδικα που περιέχει την κλήση της μεθόδου `readFromFile` είναι πιθανό να προκαλέσει κάποια εξαίρεση. Στο τμήμα `catch` ορίζεται τί θα συμβεί αν όντως προκληθεί εξαίρεση: στην προκειμένη περίπτωση τυπώνεται κάποιο διαγνωστικό μήνυμα.

Στην περίπτωση που κάποιο τμήμα κώδικα, το οποίο περιέχεται σε κάποιο τμήμα `try`, προκαλέσει μια εξαίρεση, τότε η εκτέλεση του τμήματος αυτού διακόπτεται. Ο έλεγχος μεταφέρεται στο τμήμα `catch` και ελέγχεται αν η εξαίρεση που προκλήθηκε μπορεί να εκχωρηθεί σε μια μεταβλητή του τύπου εξαίρεσης που καθορίζεται εκεί. Στην περίπτωση αυτή, εκτελείται το τμήμα `catch`. Σε ένα τμήμα `try` μπορούν να αντιστοιχούν περισσότερα τμήματα

catch, που το καθένα να χειρίζεται διαφορετικό τύπο εξαίρεσης. Σε περίπτωση εξαίρεσης, ο έλεγχος μεταφέρεται στο πρώτο τμήμα catch που ορίζει τύπο συμβατό με την εξαίρεση. Αυτό φαίνεται στο επόμενο παράδειγμα:

```
try {
   ...
   readFromFile(file);
   ...
}
catch (FileNotFoundException e) {
   System.err.println("Cannot find the file!");
}
catch (IOException e) {
   System.err.println("Error while reading!");
}
```

Στην περίπτωση που η κλήση της readFromFile προκαλέσει εξαίρεση τύπου FileNotFoundException, ο έλεγχος μεταφέρεται στο πρώτο τμήμα catch, ενώ αν η εξαίρεση είναι κάθε άλλου τύπου, συμβατού με IOException, ο έλεγχος μεταφέρεται στο δεύτερο τμήμα catch.

Αν η εξαίρεση που προκαλείται μέσα σε ένα τμήμα try δεν είναι συμβατή με κανέναν από τους τύπους εξαιρέσεων που ορίζονται στα αντίστοιχα τμήματα catch, τότε η εξαίρεση δε συλλαμβάνεται. Στην περίπτωση αυτή, η εξαίρεση μεταδίδεται προς τον υπόλοιπο κώδικα, σαν να είχε προκληθεί από το τμήμα κώδικα που περιέχει τη δομή try / catch. Μια τέτοια εξαίρεση μπορεί να συλληφθεί από κάποια εξωτερική δομή try / catch. Αν δεν υπάρχει τέτοια δομή, τότε η εξαίρεση συλλαμβάνεται από το διερμηνέα της Java, ο οποίος συνήθως τερματίζει την εκτέλεση του προγράμματος με κάποιο διαγνωστικό μήνυμα.

4.1.3. Πρόκληση εξαιρέσεων

Οι εξαιρέσεις μπορούν να προκληθούν με την εντολή throw, που συνοδεύεται από μια αναφορά σε ένα αντικείμενο. Το αντικείμενο αυτό πρέπει να προέρχεται από μια υποκλάση της Throwable, που ορίζεται στο πακέτο java.lang. Τόσο η κλάση Exception όσο και η Error είναι υποκλάσεις της Throwable. Στο τμήμα κώδικα που φαίνεται στο παρακάτω παράδειγμα, προκαλείται μια εξαίρεση του τύπου SecurityException.

```
if (!canWriteToFile(f))
   throw new SecurityException("Access denied!");
```

Η πρόκληση εξαιρέσεων με την εντολή throw είναι ιδιαίτερα χρήσιμη στην περίπτωση που ο προγραμματιστής ορίζει δικούς του τύπους εξαιρέσεων. Έστω ο παρακάτω ορισμός της υποθετικής κλάσης εξαίρεσης MyException:

```
public class MyException extends Exception
{
   private String message;

   public Exception (String msg) { message = msg; }

   public String getMessage () { return message; }
}
```

Στο παρακάτω τμήμα κώδικα προκαλείται αυτή η εξαίρεση με χρήση της εντολής `throw`. Στη συνέχεια συλλαμβάνεται από ένα τμήμα `catch`:

```
try {
   ...
   if (someKindOfSituation())
      throw new MyException("What on earth?");
   ...
}
catch (MyException e) {
   System.out.println(e.getMessage());
}
```

4.1.4. Εξαιρέσεις και κλήση μεθόδων

Όπως ήδη αναφέρθηκε, αν μια εξαίρεση δε συλληφθεί από το τμήμα κώδικα που την προκαλεί, τότε συνεχίζει να μεταδίδεται προς τα έξω, ώσπου τελικά να συλληφθεί από κάποιο εξωτερικό τμήμα κώδικα ή από το διερμηνέα της Java. Αυτό σημαίνει ότι η κλήση μιας μεθόδου μπορεί να προκαλέσει εξαίρεση, αν αυτή προκληθεί από την εκτέλεση του κώδικά της και δε συλληφθεί στο εσωτερικό της μεθόδου. Στην περίπτωση αυτή, η Java απαιτεί τη ρητή δήλωση του ότι η μέθοδος μπορεί να προκαλέσει εξαίρεση, στην επικεφαλίδα της μεθόδου. Αυτό γίνεται με τη λέξη `throws`, που συνοδεύεται από τον τύπο των εξαιρέσεων που η μέθοδος μπορεί να προκαλέσει.

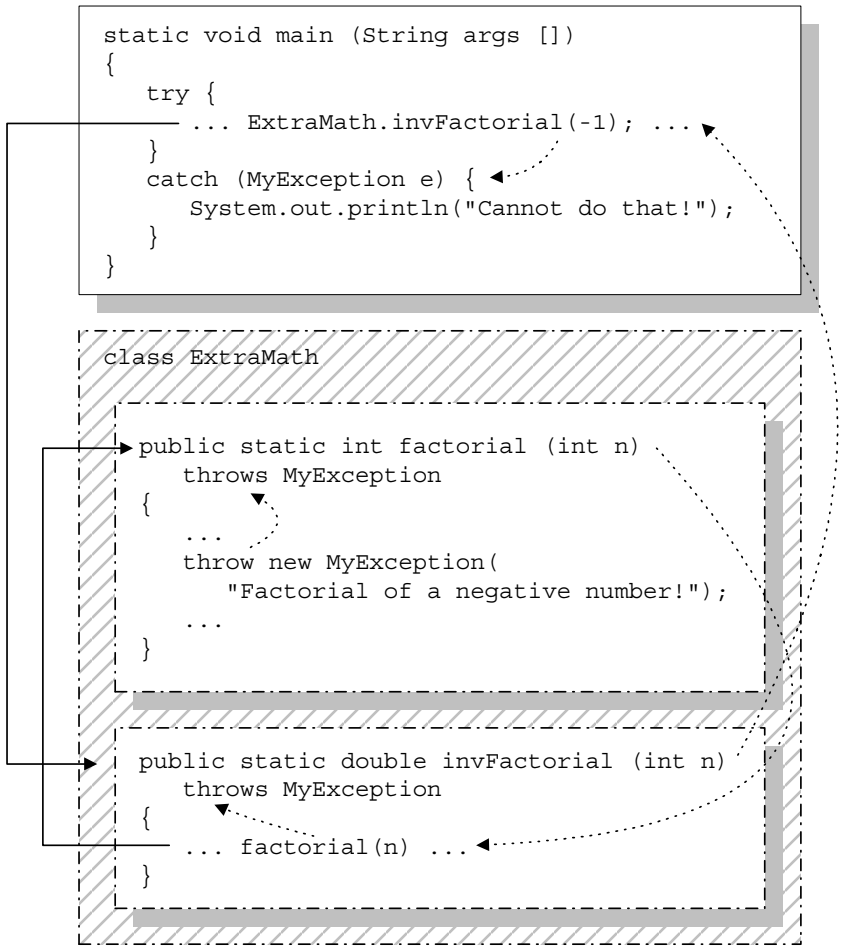
Ο παρακάτω κώδικας ορίζει μια κλάση με όνομα `ExtraMath`, που υλοποιεί μεταξύ άλλων τη συνάρτηση παραγοντικού στη στατική μέθοδο `factorial`. Η μέθοδος αυτή υπολογίζει το παραγοντικό ενός μη αρνητικού ακέραιου αριθμού. Προκαλεί μια εξαίρεση `MyException` στην περίπτωση που το όρισμά της είναι αρνητικό. Επειδή η εξαίρεση αυτή δε συλλαμβάνεται στο εσωτερικό της μεθόδου, στην επικεφαλίδα της δηλώνεται ότι μπορεί να προκληθεί μια εξαίρεση αυτού του τύπου.

```
public class ExtraMath
{
   public static int factorial (int n)
      throws MyException
   {
      if (n < 0)
         throw MyException(
            "Factorial of a negative number!");

      int result = 1;

      for (int i = 2 ; i <= n ; i++)
         result *= i;
      return result;
   }

   public static double invFactorial (int n)
      throws MyException
   {
      return 1.0 / (double) factorial(n);
   }
}
```



Σχήμα 4.3. Πρόκληση εξαίρεσης κατά την κλήση μεθόδου.

Σημειώνεται επίσης ότι η δήλωση πρόκλησης εξαίρεσης στην επικεφαλίδα μιας μεθόδου είναι απαραίτητη, ακόμα κι αν η εξαίρεση προκαλείται έμμεσα, όπως στην περίπτωση της μεθόδου `invFactorial` στο προηγούμενο παράδειγμα. Η μέθοδος αυτή καλεί τη `factorial`, που μπορεί να προκαλέσει εξαίρεση τύπου `MyException`, χωρίς να τη συλλαμβάνει. Κατά συνέπεια, από την κλήση της `invFactorial` μπορεί να προκληθεί εξαίρεση αυτού του τύπου. Μια κλήση της μεθόδου `invFactorial` απεικονίζεται στο Σχήμα 4.3. Τα βέλη με συνεχή γραμμή παριστάνουν την κανονική ροή εκτέλεσης, ενώ τα βέλη με διακεκομμένη γραμμή παριστάνουν τη ροή της εκτέλεσης μετά την πρόκληση της εξαίρεσης, μέχρι τη σύλληψή της.

4.1.5. Ανάνηψη από εξαιρέσεις

Πολλές φορές απαιτείται η εκτέλεση κάποιου τμήματος κώδικα, άσχετα από το αν έχει προκληθεί εξαίρεση ή όχι. Ο κώδικας αυτός συνήθως χρησιμοποιείται προκειμένου να γίνουν απαραίτητες ενέργειες, όπως π.χ. το κλείσιμο κάποιων αρχείων. Για το λόγο αυτό, η Java παρέχει την εντολή `finally`, που συνδυάζεται με τις `try` και `catch`. Τα περιεχόμενα του τμήματος `finally` μιας δομής που ξεκινά με την εντολή `try` εκτελούνται οπωσδήποτε πριν η ροή εκτέλεσης εγκαταλείψει τη δομή. Αυτό γίνεται είτε δεν προκληθεί εξαίρεση και η εκτέλεση των περιεχομένων του τμήματος `try` ολοκληρωθεί κανονικά, είτε αν προκληθεί εξαίρεση και ανεξάρτητα από το αν αυτή συλληφθεί.

Το παρακάτω τμήμα κώδικα περιέχει δυο δομές `try`. Η πρώτη αποσκοπεί στο άνοιγμα του αρχείου, ενώ η δεύτερη στο διάβασμα και το κλείσιμο. Η εντολή `finally` χρησιμοποιείται στη δεύτερη δομή, προκειμένου να κλείσει το αρχείο. Όπως φαίνεται, το κλείσιμο είναι ανεξάρτητο από το αν έχει προκληθεί εξαίρεση κατά το διάβασμα του αρχείου. Όλες οι μέθοδοι διαχείρισης αρχείων του παραδείγματος είναι υποθετικές.

```
File f;

try {
   f = openTheFile();
}
catch (IOException e) {
   System.out.println("Sorry, cannot open file!");
   return;
}

try {
   byte [] contents = readFromFile(f);

   // do something with the contents...
}
catch (IOException e) {
   System.out.println("Sorry, cannot read!");
}
finally {
   closeTheFile(f);
}
```

Σε αυτό το παράδειγμα, αξίζει να γίνουν δυο παρατηρήσεις, σχετικές με την ορατότητα των μεταβλητών που ορίζονται σε μια δομή `try`. Η μεταβλητή `f` δε θα μπορούσε να ορισθεί στο εσωτερικό της πρώτης δομής, όπως π.χ. γίνεται με τη μεταβλητή `contents` στη δεύτερη, γιατί τότε δε θα ήταν δυνατό να χρησιμοποιηθεί έξω από το τμήμα `try` αυτής της δομής, κάτι που είναι απαραίτητο για το διάβασμα και το κλείσιμο του αρχείου. Επίσης, αν το τμήμα `catch` της πρώτης δομής `try` δεν τελείωνε με την εντολή `return`, ο μεταγλωττιστής θα διαμαρτυρόταν ότι η μεταβλητή `f` είναι πιθανό να χρησιμοποιηθεί στη συνέχεια, χωρίς να έχει αρχικοποιηθεί. Αυτό θα συνέβαινε αν προκαλείτο εξαίρεση στην πρώτη δομή `try` και ο έλεγχος εκτέλεσης κατέληγε μετέπειτα στο διάβασμα του αρχείου.

4.2. Πολλαπλά νήματα εκτέλεσης

Πολλές σύγχρονες γλώσσες προγραμματισμού επιτρέπουν τη χρήση πολλαπλών *νημάτων εκτέλεσης* (multiple execution threads). Αυτή η προγραμματιστική τεχνική είναι ιδιαίτερα δημοφιλής για τον προγραμματισμό διαπροσωπειών ανθρώπου-υπολογιστή, καθώς και για την επίλυση άλλων προβλημάτων, για τα οποία είναι απλούστερο να φαντασθεί κανείς τη λύση ως συνδυασμό πολλών παράλληλων εργασιών, παρά ως ένα μοναδικό αλγόριθμο. Σε πολλές γλώσσες προγραμματισμού, η δυνατότητα ύπαρξης πολλαπλών νημάτων έχει προστεθεί με τη βοήθεια βιβλιοθηκών χαμηλού επιπέδου, εφόσον φυσικά αυτό υποστηρίζεται από το λειτουργικό σύστημα. Η Java παρέχει ένα μηχανισμό πολλαπλών νημάτων ως μέρος της ίδιας της γλώσσας.

Κάθε νήμα εκτέλεσης είναι ουσιαστικά μια ανεξάρτητη ροή του ελέγχου εκτέλεσης, μέσα στο πρόγραμμα. Η έννοια του νήματος (thread) είναι παρόμοια με αυτήν της διεργασίας (process), με τη διαφορά ότι τα διάφορα νήματα μοιράζονται τον ίδιο χώρο διευθύνσεων, κάτι που δε συμβαίνει συνήθως με τις διεργασίες. Με άλλα λόγια, δυο διαφορετικά νήματα μέσα σε ένα

πρόγραμμα έχουν τη δυνατότητα πρόσβασης σε κάποιο αντικείμενο του προγράμματος, ενώ δυο διαφορετικές διεργασίες συνήθως βλέπουν διαφορετικά αντίγραφα του ίδιου αντικειμένου. Φυσικά, η ταυτόχρονη πρόσβαση σε αντικείμενα στη Java μπορεί να προκαλέσει ανεπιθύμητες παρενέργειες, χωρίς τον κατάλληλο συγχρονισμό των νημάτων.

Στις επόμενες παραγράφους περιγράφεται ο μηχανισμός πολλαπλών νημάτων της Java, καθώς και οι μέθοδοι χειρισμού και συγχρονισμού των νημάτων. Στο τέλος της ενότητας, ως παράδειγμα χρήσης πολλαπλών νημάτων δίνεται ένα πρόγραμμα Java που προσομοιώνει το κλασσικό πρόβλημα των φιλοσόφων που γευματίζουν.

4.2.1. Δημιουργία νημάτων

Όταν αρχίζει η εκτέλεση ενός προγράμματος, μπορεί κανείς να φαντασθεί ότι υπάρχει μόνο ένα νήμα εκτέλεσης.[7] Για να δημιουργηθεί ένα νέο νήμα, πρέπει να δημιουργηθεί ένα νέο αντικείμενο, που να προέρχεται από την κλάση `Thread`, που ορίζεται στο πακέτο `java.lang`, ή από κάποια υποκλάση της. Το νέο αυτό αντικείμενο παριστάνει μια νέα εργασία που πρέπει να αρχίσει να εκτελείται παράλληλα με την παλιά. Υπάρχουν δυο τρόποι για την υλοποίηση νέων νημάτων εκτέλεσης. Σε κάθε περίπτωση, ο κώδικας που υλοποιεί τη νέα εργασία τοποθετείται σε μια μέθοδο με όνομα `run`.

Ο πρώτος τρόπος, που θα χρησιμοποιείται στη συνέχεια του κεφαλαίου, απαιτεί τη δημιουργία μιας νέας κλάσης για τη νέα εργασία, που να υλοποιεί τη διαπροσωπεία `Runnable`. Η διαπροσωπεία αυτή δηλώνει μόνο μια μέθοδο, με όνομα `run`. Για παράδειγμα, η παρακάτω κλάση υλοποιεί μια νέα εργασία, που εκτυπώνει στην οθόνη επ' άπειρον την τιμή ενός μετρητή:

```
class CountingJob implements Runnable
{
   private int counter;

   public CountingJob (int c) { counter = c; }

   public void run ()
   {
      while (true)
         System.out.println(counter++);
   }
}
```

Στη συνέχεια, για να αρχίσει η εκτέλεση της εργασίας, απαιτείται η δημιουργία ενός αντικειμένου τύπου `Thread`, περνώντας στον κατασκευαστή του ένα αντικείμενο της νέας εργασίας ως παράμετρο. Η έναρξη της εκτέλεσης πραγματοποιείται καλώντας τη μέθοδο start του αντικειμένου `Thread`:

```
CountingJob job = new CountingJob(1);
Thread t = new Thread (job);

t.start();
```

[7] Η παρατήρηση αυτή δεν είναι απόλυτα ακριβής. Στην περίπτωση των ανεξάρτητων εφαρμογών Java, εκτός από το νήμα εκτέλεσης του προγράμματος, τρέχει παράλληλα τουλάχιστον ο συλλέκτης σκουπιδιών. Στην περίπτωση των Java applets, ο αριθμός των νημάτων εκτέλεσης είναι δυνατό να αυξηθεί κατά πολύ, καθώς το σύστημα δημιουργεί νέα νήματα όποτε το θεωρεί σκόπιμο.

Η εκτέλεση της νέας εργασίας συνεχίζεται ώσπου είτε να τελειώσει η εκτέλεση της μεθόδου `run`, είτε να κληθεί η μέθοδος `stop` του αντικειμένου `t`.

Το αντικείμενο τύπου `Thread` θα μπορούσε να αποτελεί τμήμα της κλάσης `CountingJob` και η εκτέλεση της νέας εργασίας να αρχίζει στον κατασκευαστή αυτής της κλάσης:

```
class CountingJob implements Runnable
{
   private int counter;
   private Thread thread;

   public CountingJob (int c)
   {
      counter = c;
      thread = new Thread(this);
      thread.start();
   }

   public void run ()
   {
      while (true)
         System.out.println(counter++);
   }
}
```

Ο δεύτερος τρόπος απαιτεί τη δημιουργία μιας νέας υποκλάσης της `Thread`, που να ορίζει τη μέθοδο `run`. Το προηγούμενο παράδειγμα μπορεί να γραφεί σύμφωνα με αυτό τον τρόπο ως εξής:

```
class CountingJob extends Thread
{
   private int counter;

   public CountingJob (int c)
   {
      counter = c;
      start();
   }

   public void run ()
   {
      while (true)
         System.out.println(counter++);
   }
}
```

4.2.2. Χειρισμός νημάτων

Ο Πίνακας 4.1 περιγράφει τις βασικότερες μεθόδους της κλάσης `Thread`. Οι μέθοδοι `start` και `stop` μπορούν να κληθούν μόνο μια φορά, κατά τη διάρκεια ζωής ενός νήματος. Αντίθετα, οι μέθοδοι `suspend` και `resume` μπορούν να χρησιμοποιηθούν περισσότερες φορές. Η μέθοδος `sleep` χρησιμοποιείται για την καθυστέρηση ενός νήματος εκτέλεσης για χρόνο που καθορίζεται από την παράμετρο σε milliseconds. Σε αυτό το χρονικό διάστημα, το νήμα δεν εκτελείται. Η `sleep` προκαλεί την εξαίρεση `InterruptedException` σε περίπτωση που το νήμα διακοπεί για κάποιο λόγο. Η στατική μέθοδος `currentThread` επιστρέφει το τρέχον νήμα εκτέλεσης και είναι χρήσιμη για πρόσβαση στο αρχικό νήμα.

Η μέθοδος `setDaemon` δέχεται ως παράμετρο μια τιμή `boolean`. Αν αυτή είναι `true`, τότε το νήμα χαρακτηρίζεται ως "δαίμων". Οι δαίμονες είναι νήματα που καταστρέφονται αυτόματα μετά τον τερματισμό όλων των άλλων νημάτων.

Όταν ένα πρόγραμμα που περιέχει περισσότερα νήματα εκτέλεσης τρέχει σε έναν υπολογιστή με μια μόνο μονάδα επεξεργασίας, όπως είναι το σύνηθες στις μέρες μας, είναι προφανές ότι τα νήματα δεν είναι δυνατό να εκτελούνται παράλληλα. Αντίθετα, τα νήματα "μάχονται" μεταξύ τους, προκειμένου να τους παραχωρηθεί από τη μοναδική μονάδα επεξεργασίας λίγος χρόνος για να συνεχίσουν την εκτέλεσή τους.

Πίνακας 4.1. Χρήσιμες μέθοδοι της κλάσης **`Thread`.**

Μέθοδος	***Περιγραφή***
`start()`	Έναρξη της εκτέλεσης.
`stop()`	Τερματισμός της εκτέλεσης.
`suspend()`	Προσωρινή παύση της εκτέλεσης.
`resume()`	Επανέναρξη της εκτέλεσης, μετά από παύση.
`sleep(n)`	Καθυστέρηση της εκτέλεσης κατά χρόνο n milliseconds.
`currentThread()`	Επιστροφή τρέχοντος νήματος εκτέλεσης.
`setDaemon(b)`	Χαρακτηρισμός του νήματος ως "δαίμονα".
`setPriority(n)`	Καθορισμός της προτεραιότητας του νήματος.
`yield()`	Παραχώρηση προτεραιότητας.

Η μέθοδος `setPriority` χρησιμοποιείται για τον καθορισμό της προτεραιότητας εκτέλεσης του νήματος. Η προτεραιότητα είναι μια αριθμητική τιμή: όσο μεγαλύτερη είναι αυτή, τόσο περισσότερος χρόνος αφιερώνεται από τον υπολογιστή για την εκτέλεση του νήματος. Με άλλα λόγια, νήματα με μεγαλύτερη προτεραιότητα πριμοδοτούνται από το διερμηνέα της Java έναντι νημάτων με μικρότερη προτεραιότητα. Ένα νήμα μπορεί να παραχωρήσει το χρόνο υπολογισμού που του έχει δοθεί καλώντας τη μέθοδο `yield`.

4.2.3. Συγχρονισμός νημάτων

Στις περισσότερες περιπτώσεις, όταν πολλά νήματα εκτέλεσης διαχειρίζονται το ίδιο σύνολο αντικειμένων, είναι επιθυμητό να υπάρχει συγχρονισμός στις εργασίες που επιτελούνται. Αυτό είναι απαραίτητο, γιατί διαφορετικά τα αποτελέσματα ενδέχεται να είναι καταστροφικά. Αν, για παράδειγμα, ένα νήμα εκτέλεσης διαβάζει μια μεταβλητή ενός αντικειμένου την ίδια στιγμή που ένα άλλο νήμα εκχωρεί μια τιμή σε αυτή, το αποτέλεσμα και των δυο ενεργειών είναι εντελώς απρόβλεπτο. Το πρόβλημα του συγχρονισμού παράλληλων διεργασιών είναι εξαιρετικά δύσκολο, αφού εκτός από το να παρέχει τη δυνατότητα αποκλειστικής χρήσης στα αντικείμενα, ένας μηχανισμός συγχρονισμού πρέπει να εξασφαλίζει την αποφυγή αδιεξόδων.

Η Java παρέχει έναν αρκετά απλό και εύχρηστο μηχανισμό για το συγχρονισμό των νημάτων εκτέλεσης. Ο μηχανισμός αυτός βασίζεται στην έννοια του *κλειδώματος* (lock).[8] Κάθε

[8] Ο μηχανισμός αυτός συγχρονισμού προτάθηκε από τον C.A.R. Hoare και αναφέρεται στη βιβλιογραφία ως *monitors*.

αντικείμενο της Java μπορεί να θεωρηθεί ότι διαθέτει ένα κλειδί. Το ίδιο συμβαίνει και για κάθε κλάση. Προκειμένου να επιτευχθεί αποκλειστική πρόσβαση σε ένα σύνολο μεθόδων ενός αντικειμένου, αυτές δηλώνονται ως `synchronized` (συγχρονισμένες). Στη συνέχεια, για να κληθεί μια συγχρονισμένη μέθοδος ενός αντικειμένου, πρέπει να αποκτηθεί το κλειδί του αντικειμένου από το νήμα εκτέλεσης που πραγματοποιεί την κλήση. Το κλειδί επιστρέφεται μετά την εκτέλεση της μεθόδου.

Ας θεωρήσουμε για παράδειγμα την ακόλουθη κλάση `SafeVariable`. Η κλάση αυτή ορίζει δυο συγχρονισμένες μεθόδους `get` και `put`, που χρησιμοποιούνται για την ανάκτηση και την αποθήκευση μιας ακέραιας τιμής από ένα αντικείμενο:

```
public class SafeVariable
{
   private int value;

   public synchronized int get () { return value; }
   public synchronized void put (int v) { value = v; }
}
```

Με τη χρήση των μεθόδων αυτών εξασφαλίζεται ότι για κάθε αντικείμενο αυτής της κλάσης, το πολύ μια μέθοδος θα εκτελείται κάθε χρονική στιγμή. Τα κλειδιά των κλάσεων χρησιμοποιούνται για στατικές συγχρονισμένες μεθόδους, κατά τρόπο παρόμοιο με τα κλειδιά των αντικειμένων.

Εκτός από τη δήλωση συγχρονισμένων μεθόδων, είναι δυνατός ο συγχρονισμός αυθαίρετων τμημάτων κώδικα, με χρήση και πάλι της λέξης `synchronized`. Για παράδειγμα, στο παρακάτω τμήμα κώδικα, προκειμένου να εκτελεσθεί το εσωτερικό της εντολής πρέπει να αποκτηθεί το κλειδί του αντικειμένου `obj`. Αυτό το τμήμα κώδικα μπορεί όμως να ανήκει σε οποιαδήποτε μέθοδο, όχι απαραίτητα του αντικειμένου αυτού.

```
synchronized (obj) {
   // manipulate obj in some way...
}
```

Εκτός από το συγχρονισμό μεθόδων με χρήση της λέξης `synchronized`, η Java παρέχει τη δυνατότητα συγχρονισμού με τις μεθόδους `wait` και `notify` της κλάσης `Object`. Οι μέθοδοι αυτές ορίζονται για κάθε αντικείμενο, καθώς όλες οι κλάσεις κληρονομούν την `Object`. Οι κλήσεις σε αυτές τις μεθόδους πραγματοποιούνται σχεδόν πάντα μέσα από συγχρονισμένες μεθόδους ή τμήματα κώδικα.

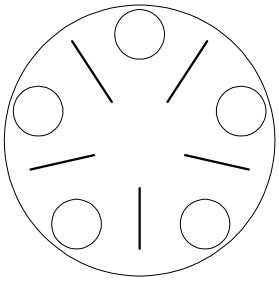
Με την κλήση της μεθόδου `wait` για κάποιο αντικείμενο, το τρέχον νήμα εκτέλεσης επιστρέφει το κλειδί του αντικειμένου. Αν το νήμα βρίσκεται σε μια συγχρονισμένη μέθοδο αυτού του αντικειμένου, η επιστροφή του κλειδιού έχει ως αποτέλεσμα να σταματήσει προσωρινά η εκτέλεση του νήματος. Αυτό είναι επιθυμητό στην περίπτωση που η εκτέλεση της εργασίας δεν είναι δυνατό να ολοκληρωθεί λόγω έλλειψης πληροφοριών, για τις οποίες πρέπει το τρέχον νήμα να περιμένει.

Για να συνεχίσει η εκτέλεση θα πρέπει να συμβούν δυο πράγματα: αφενός να ενημερωθεί το νήμα εκτέλεσης ότι μπορεί να συνεχίσει, μέσω της κλήσης στη μέθοδο `notify` του αντικειμένου από ένα άλλο νήμα, και αφετέρου να αποκτηθεί και πάλι το κλειδί. Για κάθε κλήση της μεθόδου `notify` ενός αντικειμένου, ο διερμηνέας της Java ενημερώνει μόνο μια μέθοδο που έχει εκτελέσει τη μέθοδο `wait` για αυτό το αντικείμενο. Αν πρέπει να ενημερωθούν όλες οι μέθοδοι που βρίσκονται σε αναμονή, μπορεί να χρησιμοποιηθεί η μέθοδος `notifyAll`.

Ένα εκτενές παράδειγμα της χρήσης των μεθόδων `wait` και `notify` για το συγχρονισμό νημάτων εκτέλεσης δίνεται στην επόμενη παράγραφο.

4.2.4. Παράδειγμα: Οι φιλόσοφοι που γευματίζουν

Το *πρόβλημα των φιλοσόφων που γευματίζουν* (dining philosophers' problem) είναι ένα από τα κλασσικότερα προβλήματα στη βιβλιογραφία που ασχολείται με το συγχρονισμό παράλληλων διεργασιών. Η εικόνα που περιγράφεται σε αυτό το πρόβλημα είναι η εξής:



Σχήμα 4.4. Φιλόσοφοι που γευματίζουν.

Πέντε Κινέζοι φιλόσοφοι περνούν τη ζωή τους κάνοντας εναλλάξ δυο μόνο πράγματα: σκέπτονται και γευματίζουν. Όταν πρόκειται να γευματίσει, κάθε φιλόσοφος εισέρχεται στην τραπεζαρία και κάθεται σε ένα κυκλικό τραπέζι με πέντε θέσεις. Στο τραπέζι υπάρχουν πέντε πιάτα και πέντε ξυλάκια, στη διάταξη που περιγράφεται στο Σχήμα 4.4. Προκειμένου να γευματίσει, κάθε φιλόσοφος χρειάζεται να πιάσει τα δυο ξυλάκια που βρίσκονται δίπλα στο πιάτο του και να τα χρησιμοποιήσει για να φάει με αυτά το ρύζι του. Κάθε ξυλάκι διεκδικείται φυσικά από δυο φιλοσόφους.

Το πρόγραμμα Java που ακολουθεί είναι μια απλοϊκή προσομοίωση του προβλήματος των φιλοσόφων που γευματίζουν, χρησιμοποιώντας πέντε νήματα εκτέλεσης, ένα για κάθε φιλόσοφο. Το πρόγραμμα περιέχει τρεις κλάσεις. Η πρώτη κλάση υλοποιεί τους φιλοσόφους, η δεύτερη τα ξυλάκια και η τρίτη παρέχει τη μέθοδο `main` που απαιτείται για να τρέξει η εφαρμογή.

Κλάση Philosopher

Η κλάση αυτή υλοποιεί τη διαπροσωπεία `Runnable` και ξεκινά στον κατασκευαστή της ένα νέο νήμα εκτέλεσης. Η μέθοδος `run` περιγράφει τη ζωή ενός φιλοσόφου: μια αέναη εναλλαγή σκέψης και γευμάτων. Οι δυο μέθοδοι `think` και `eat` υλοποιούν αυτές τις δυο διαδικασίες, εκτυπώνοντας κατάλληλα μηνύματα και καθυστερώντας την εκτέλεση για ένα τυχαίο χρονικό διάστημα. Οι σταθερές `MAX_THINK_DELAY` και `MAX_EAT_DELAY` προσδιορίζουν τις μέγιστες τιμές της καθυστέρησης. Η μέθοδος `kill` χρησιμοποιείται για τον τερματισμό του νήματος ενός φιλοσόφου.

Πριν αρχίσει να τρώει, κάθε φιλόσοφος είναι υποχρεωμένος να πιάσει τα ξυλάκια που βρίσκονται εκατέρωθεν του πιάτου του, στα οποία αναφέρεται μέσω των μεταβλητών `leftStick` και `rightStick`. Αυτό απαιτεί προφανώς το συγχρονισμό των φιλοσόφων, καθώς δεν είναι απαραίτητο ότι τα ξυλάκια θα είναι διαθέσιμα. Ο συγχρονισμός επιτυγχάνεται στο εσωτερικό των μεθόδων `grab` και `release` της κλάσης `Stick`, που θα περιγραφεί στην επόμενη ενότητα. Ο κώδικας της κλάσης `Philosopher` ακολουθεί.

```
public class Philosopher implements Runnable
{
   private static final int MAX_THINK_DELAY = 1000;
   private static final int MAX_EAT_DELAY = 500;

   private String name;
   private Stick leftStick, rightStick;
   private Thread thread;

   public Philosopher (String n, Stick left, Stick right)
   {
      name = n;
      leftStick = left;
      rightStick = right;
      thread = new Thread(this);
      thread.start();
   }

   public void run ()
   {
      while (true) {
         try {
            think();
            rightStick.grab(this);
            leftStick.grab(this);
            eat();
            rightStick.release(this);
            leftStick.release(this);
         }
         catch (InterruptedException e) { }
      }
   }

   private void think () throws InterruptedException
   {
      int ms = (int) (Math.random() * MAX_THINK_DELAY);

      System.out.println(name + " is thinking.");
      thread.sleep(ms);
   }

   private void eat () throws InterruptedException
   {
      int ms = (int) (Math.random() * MAX_EAT_DELAY);

      System.out.println(name + " is eating.");
      thread.sleep(ms);
   }

   public void kill ()
   {
      thread.stop();
   }
}
```

Κλάση Stick

Η κλάση αυτή αναλαμβάνει το συγχρονισμό της διαδικασίας των γευμάτων. Κάθε ξυλάκι μπορεί να βρίσκεται σε μια από δυο δυνατές καταστάσεις: ελεύθερο ή κατειλημμένο από κάποιο φιλόσοφο. Η μεταβλητή `owner` έχει τιμή `null` στην πρώτη κατάσταση, ενώ στη δεύτερη κατάσταση περιέχει μια αναφορά στο φιλόσοφο που κρατά το ξυλάκι.

Η κλάση `Stick` παρέχει δυο συγχρονισμένες μεθόδους, τις `grab` και `release`. Και οι δυο δέχονται ως παράμετρο μια αναφορά στο φιλόσοφο που ζητάει ή απελευθερώνει το ξυλάκι. Η `grab` καθυστερεί την εκτέλεση του νήματος, καλώντας τη μέθοδο `wait` του αντικειμένου, μέχρις ότου το ξυλάκι ελευθερωθεί από τον κάτοχό του. Στη συνέχεια, παραχωρεί το ξυλάκι στο φιλόσοφο που το ζήτησε. Η μέθοδος `release` χρησιμοποιείται για να απελευθερωθεί το ξυλάκι μετά τη χρήση του. Αφού ελέγξει αν όντως ο φιλόσοφος που το απελευθερώνει είναι ο νόμιμος κάτοχός του, η μέθοδος αυτή ενημερώνει ένα από τα νήματα που περιμένουν το ξυλάκι, χρησιμοποιώντας τη μέθοδο `notify`. Στην περίπτωση αυτή δεν έχει νόημα η χρήση της `notifyAll`, γιατί μόνο ένας φιλόσοφος θα καταφέρει τελικά να πιάσει το ξυλάκι. Ο διερμηνέας της Java αναλαμβάνει να ενημερώσει μόνο τον “τυχερό” φιλόσοφο.

Ο κώδικας της κλάσης `Stick` ακολουθεί:

```
public class Stick
{
   private Philosopher owner = null;

   public synchronized void grab (Philosopher p)
      throws InterruptedException
   {
      while (owner != null)
         wait();
      owner = p;
   }

   public synchronized void release (Philosopher p)
   {
      if (p == owner)
         owner = null;
      notify();
   }
}
```

Κλάση Dining

Η κλάση αυτή χρησιμοποιείται μόνο για τον ορισμό μιας μεθόδου `main`. Η μέθοδος αυτή κατασκευάζει πέντε φιλοσόφους και πέντε ξυλάκια. Στη συνέχεια, δίνει στους φιλοσόφους 30 δευτερόλεπτα ζωής, στα οποία καταφέρνουν να σκεφθούν και να φάνε έναν ικανοποιητικό αριθμό φορών, ώστε να παρατηρήσει κανείς τη λειτουργία του προγράμματος.

```
class Dining
{
   private static final int NUMBER = 5;

   public static void main (String args [])
   {
      Philosopher [] philosopher = new Philosopher [NUMBER];
      Stick [] stick = new Stick [NUMBER];

      // Create sticks

      for (int i = 0 ; i < n ; i++)
         stick[i] = new Stick();

      // Create philosophers

      for (int i = 0 ; i < n ; i++) {
         String name = "Philosopher " + (i+1);
```

```
            Stick left = stick[i], right = stick[(i+1)%n];

            philosopher[i] = new Philosopher(name, left, right);
         }

         // Run the simulation for 30 seconds

         try {
            Thread.currentThread().sleep(30000);
         }
         catch (InterruptedException e) { }

         // Kill the philosophers

         for (int i = 0 ; i < n ; i++)
            philosopher[i].kill();
      }
   }
```

Παρατηρήσεις

Η λύση που δίνεται στο πρόβλημα των φιλοσόφων που γευματίζουν στις προηγούμενες παραγράφους δεν ασχολείται καθόλου με τα πολύ σοβαρά προβλήματα της αποφυγής αδιεξόδων και της δίκαιης μεταχείρισης των φιλοσόφων. Οι σημειώσεις αυτές δεν αποσκοπούν στην ανάλυση των δυο αυτών περιπτώσεων, που περιγράφονται απλώς στη συνέχεια, προκειμένου να πάρει ο αναγνώστης μια ιδέα των προβλημάτων που προκύπτουν κατά το συγχρονισμό νημάτων εκτέλεσης.

- Το πρόγραμμα μπορεί να οδηγηθεί σε αδιέξοδο αν όλοι οι φιλόσοφοι συγχρόνως εκτελέσουν την μέθοδο grab για το δεξιό ξυλάκι τους και επιτύχουν να το πιάσουν. Στη συνέχεια, κάθε φιλόσοφος θα περιμένει να απελευθερωθεί το αριστερό του ξυλάκι, πράγμα που δε θα συμβεί ποτέ και οι φιλόσοφοι θα πεθάνουν από την πείνα.
- Προκειμένου να ξεπερασθεί το αδιέξοδο μπορούν να προταθούν αρκετές λύσεις. Μια πιθανή λύση επιβάλλει στους φιλοσόφους να επιστρέφουν το δεξί τους ξυλάκι, αν δεν μπορούν να πιάσουν και το αριστερό. Μια δεύτερη πιθανή λύση, μάλλον απλούστερη, καταστρέφει τη συμμετρία του προβλήματος ορίζοντας ότι ένας συγκεκριμένος φιλόσοφος είναι αριστερόχειρας και σηκώνει πρώτα το αριστερό του ξυλάκι.
- Στο πρόγραμμα που περιγράφηκε, δε διασφαλίζεται ότι όλοι οι φιλόσοφοι θα έχουν κάποια στιγμή την ευκαιρία να γευματίσουν. Το πρόβλημα της δίκαιης μεταχείρισης στην εκτέλεση των νημάτων βρίσκεται στη δικαιοδοσία του διερμηνέα της Java, η διάθεση του οποίου για απονομή δικαιοσύνης εξαρτάται πολύ από την υλοποίησή του.

5. Βοηθητικές κλάσεις

Στο κεφάλαιο αυτό ξεκινά η εξερεύνηση του API της Java. Περιγράφονται ορισμένες βοηθητικές κλάσεις, που είναι απαραίτητες στην ανάπτυξη λογισμικού. Ανάμεσα σε αυτές, οι πιο χρήσιμες είναι η κλάση String, για την υλοποίηση συμβολοσειρών, η κλάση Math, για τη χρήση μαθηματικών συναρτήσεων, και η κλάση Date για την υλοποίηση ημερομηνιών. Οι κλάσεις αυτές ορίζονται στα πακέτα java.lang και java.util.

5.1. Συμβολοσειρές

Η κλάση `String` του API της Java, που ορίζεται στο πακέτο `java.lang`, υλοποιεί πολλές λειτουργίες για τη διαχείριση συμβολοσειρών. Βασική αρχή είναι ότι τα αντικείμενα που προέρχονται από την κλάση `String` δεν μπορούν να μεταβληθούν μετά τη δημιουργία τους. Οι μέθοδοι της κλάσης `String` που αποσκοπούν στη μεταβολή του περιεχομένου των συμβολοσειρών, στη Java επιστρέφουν απλώς νέες συμβολοσειρές. Επίσης σημαντικό είναι ότι η κλάση `String` είναι τελική, πράγμα που σημαίνει ότι δεν μπορούν να ορισθούν υποκλάσεις.

5.1.1. Κατασκευή και απλές λειτουργίες

Η κατασκευή μιας συμβολοσειράς γίνεται πολύ απλά, όπως φαίνεται στο ακόλουθο παράδειγμα:

```
String a = "This is a string";
```

Υπενθυμίζεται ότι η τιμή της έκφρασης `"This is a string"` είναι μια αναφορά σε ένα αντικείμενο τύπου `String` και, κατά συνέπεια, η ίδια η έκφραση μπορεί να χρησιμοποιηθεί χωρίς την χρήση κάποιας μεταβλητής.

Η μέθοδος `length` επιστρέφει το μέγεθος της συμβολοσειράς, δηλαδή το πλήθος των χαρακτήρων που την αποτελούν:

```
int len = a.length();
```

Η μόνη πράξη που επιτρέπεται μεταξύ συμβολοσειρών είναι η παράθεση, με χρήση του τελεστή `+`:

```
String b = "Antonis " + "Kavarnos";
```

Για τον ίδιο σκοπό μπορεί να χρησιμοποιηθεί και η μέθοδος `concat`:

```
String c = "John ".concat("Smith");
```

Μεγάλου μήκους συμβολοσειρές δεν μπορούν να γραφούν σε περισσότερες γραμμές κώδικα, χωρίς τη χρήση της παράθεσης. Έτσι, το παρακάτω τμήμα κώδικα δεν είναι σωστό:

```
String d = "This is line 1
            This is line 2";
```

Η σωστή γραφή απαιτεί χρήση της παράθεσης και της συμβολοσειράς διαφυγής `\n` για την αλλαγή της γραμμής:

```
String d = "This is line 1\n" +
           "This is line 2";
```

5.1.2. Μετατροπές από και προς συμβολοσειρές

Είναι δυνατό να κατασκευασθεί μια συμβολοσειρά από έναν πίνακα, που περιέχει τους χαρακτήρες που την αποτελούν:

```
char [] data = {'T', 'h', 'i', 'n', 'g'};
String thing = new String(data);
```

ή από έναν πίνακα από bytes, που περιέχει τους κωδικούς των χαρακτήρων που την αποτελούν. Στην περίπτωση αυτή, απαιτείται προσοχή στη χρήση του Unicode. Η δεύτερη παράμετρος του κατασκευαστή είναι το δεύτερο (high) byte των χαρακτήρων Unicode, που συνήθως δίνεται ως `0` και αντιστοιχεί στο αγγλικό αλφάβητο:

```
byte [] data = {65, 66, 67, 68};
String abcd = new String(data, 0);   // "ABCD"
```

Η αντίστροφη μετατροπή, δηλαδή από συμβολοσειρα προς χαρακτήρες, είναι επίσης δυνατή με τη μέθοδο `charAt`, που επιστρέφει το χαρακτήρα που βρίσκεται σε μια συγκεκριμένη θέση της συμβολοσειράς, ή τη μέθοδο `toCharArray`, που μετατρέπει τη συμβολοσειρά σε πίνακα χαρακτήρων. Το ακόλουθο παράδειγμα χρησιμοποιεί τις δυο αυτές μεθόδους:

```
String s = "Antonis";
char [] c = s.toCharArray();

for (int i = 0 ; i < s.length() ; i++)
   if (s.charAt(i) != c[i])
      // Et tu String?!
```

Οι περισσότεροι τύποι δεδομένων είναι δυνατό να μετατραπούν σε συμβολοσειρές, μέσω της στατικής μεθόδου `valueOf`:

```
String one   = String.valueOf(1);
String wrong = String.valueOf(false);
```

Η μέθοδος αυτή είναι υπερφορτωμένη και αναλαμβάνει τη μεταφορά δεδομένων των βασικών τύπων κατά το συνήθη τρόπο. Αν το όρισμά της ανήκει σε τύπο αναφοράς, τότε η μέθοδος αυτή

καλεί τη μέθοδο `toString` του αντικειμένου, η οποία κληρονομείται από την κλάση `Object`. Η μέθοδος αυτή αναλαμβάνει τη μετατροπή αντικειμένων σε συμβολοσειρές. Στην περίπτωση που το όρισμα της `valueOf` είναι `null`, το αποτέλεσμα είναι η συμβολοσειρά `"null"`:

```
String date = String.valueOf(new Date());
System.out.println(date); // Tue Jan 1 0:06:34 EET 1997
```

Η αντίστροφη μετατροπή, δηλαδή από μια συμβολοσειρά σε κάποιον άλλο τύπο, δε θεωρείται λειτουργία που πρέπει να παρέχεται από την κλάση των συμβολοσειρών. Για τη μετατροπή συμβολοσειράς σε ακέραιο ή πραγματικό αριθμό χρησιμοποιούμε τις κλάσεις που αντιστοιχούν στους πρωταρχικούς τύπους και θα περιγραφούν στην επόμενη ενότητα. Οι κλάσεις αυτές περιέχουν μεθόδους που υλοποιούν αυτή τη μετατροπή:

```
String five = "5";
int i = Integer.valueOf(five).intValue();
```

5.1.3. Σύγκριση συμβολοσειρών

Ακολουθώντας τη φιλοσοφία της γλώσσας C, η αλφαβητική σύγκριση δύο συμβολοσειρών δε γίνεται απευθείας με τη χρήση κάποιων τελεστών. Οι παρακάτω συγκρίσεις δεν έχουν το επιθυμητό αποτέλεσμα:

```
"maria" == "maria"
"antonis" < "maria"
```

Η πρώτη συγκρίνει στην πραγματικότητα αναφορές σε αντικείμενα και ενδέχεται να δώσει αποτέλεσμα `false`, ενώ η δεύτερη προκαλεί σφάλμα μεταγλώττισης γιατί δεν επιτρέπεται η χρήση του τελεστή `<` για τη σύγκριση αναφορών.

Η αλφαβητική ισότητα δύο συμβολοσειρών μπορεί να ελεγχθεί με τις μεθόδους `equals` και `equalsIgnoreCase`. Η δεύτερη αγνοεί αν τα γράμματα είναι μικρά η κεφαλαία. Η σύγκριση των συμβολοσειρών γίνεται με απλή σύγκριση των αριθμών των χαρακτήρων Unicode:

```
String s1 = "one";
char [] c = {'o', 'n', 'e'};
String s2 = new String(c);

if (s1.equals(s2))
   // Of course they are equal!

String s3 = "oNe";

if (s1.equals(s3))
   // These are not equal!
if (s1.equalsIgnoreCase(s3))
   // But they are, if case is ignored!
```

Πληρέστερη σύγκριση μεταξύ συμβολοσειρών γίνεται με τη μέθοδο `compareTo`. Η μέθοδος αυτή λειτουργεί όπως η συνάρτηση `strcmp` στη C, δηλαδή επιστρέφει έναν ακέραιο μικρότερο από το `0`, αν η πρώτη συμβολοσειρά προηγείται της δεύτερης αλφαβητικά, έναν ακέραιο μεγαλύτερο από το `0`, αν η δεύτερη προηγείται της πρώτης, και `0` αν οι συμβολοσειρές είναι ίσες.

5.1.4. Επεξεργασία συμβολοσειρών

Η ανεύρεση μιας συμβολοσειράς μέσα σε μια άλλη είναι απλή υπόθεση. Οι μέθοδοι `startsWith` και `endsWith` ελέγχουν αν μια συμβολοσειρά αρχίζει ή τελειώνει με μια άλλη:

```
String alphabet = "abcdefghijklmnopqrstuvwxyz";

if ( alphabet.startsWith("abc") )
   // yes!
if ( alphabet.endsWith("xyz") )
   // yes!
```

Με τις μεθόδους `indexOf` και `lastIndexOf` είναι δυνατή η ανεύρεση της θέσης της πρώτης ή της τελευταίας εμφάνισης μιας συμβολοσειράς μέσα σε μια άλλη:

```
String s = "tumustukulurumu";

int i = s.indexOf("mu");     // this will be 2
int j = s.lastIndexOf("mu"); // this will be 13
```

Η μεταβολή μιας συμβολοσειράς γίνεται έμμεσα, με μεθόδους που επιστρέφουν συμβολοσειρές. Για παράδειγμα, είναι δυνατό να απαλειφθούν οι λευκοί χαρακτήρες, δηλαδή το κενό (space), το tab και την αλλαγή γραμμής, από την αρχή και το τέλος μιας συμβολοσειράς χρησιμοποιώντας τη μέθοδο `trim`:

```
String s = " abc \n "
String t = s.trim();    // this will be "abc"
```

Παρόμοια, η μετατροπή σε μικρά η κεφαλαία γράμματα γίνεται με τις μεθόδους `toLowerCase` και `toUpperCase`:

```
String s1 = "bLAh".toUpperCase(); // "BLAH"
String s2 = "bLAh".toLowerCase(); // "blah"
```

Επίσης, είναι δυνατό να πάρει κανείς μόνο ένα μέρος μιας συμβολοσειράς, χρησιμοποιώντας τη μέθοδο `substring`. Η πρώτη παράμετρος είναι η θέση του πρώτου χαρακτήρα, ενώ η δεύτερη παράμετρος είναι η θέση του χαρακτήρα μετά τον τελευταίο:

```
String alphabet = "abcdefghijklmnopqrstuvwxyz";
String abc      = alphabet.substring(0, 3);
```

5.1.5. Αποθήκες συμβολοσειρών

Για να αποφευχθεί η σπατάλη χώρου και χρόνου επεξεργασίας από τη δημιουργία νέων συμβολοσειρών με την κλήση κάθε μεθόδου, είναι δυνατή η χρήση της κλάσης `StringBuffer`. Η κλάση αυτή υλοποιεί μια συμβολοσειρά που μπορεί να επεκτείνεται επ' αόριστον με γρήγορο και αποδοτικό τρόπο. Έτσι το παρακάτω τμήμα κώδικα:

```
String phrase = "Hello";

phrase = phrase + " beautiful";
phrase = phrase + " world!";
```

μπορεί να αντικατασταθεί με το πιο αποδοτικό:

```
StringBuffer phrase = new StringBuffer("Hello");

phrase.append(" beautiful");
phrase.append(" world!");
```

Όπως θα περίμενε κανείς, τα περιεχόμενα ενός `StringBuffer` με τη μορφή `String` επιστρέφονται από τη μέθοδο `toString`:

```
StringBuffer phrase = new StringBuffer("Hello");
String hello = phrase.toString();
```

5.1.6. Διαχωρισμός συμβολοσειρών

Πολύ συχνά ειναι απαραίτητος ο διαχωρισμός μιας συμβολοσειράς σε κομμάτια (tokens), που χωρίζονται από κάποια διαχωριστικά σύμβολα (delimiters). Συνηθέστερα θέλει κανείς να αναγνωρίσει τις λέξεις μιας φράσης, που χωρίζονται μεταξύ τους με κενά. Για το σκοπό αυτό είναι ιδιαίτερα χρήσιμη η κλάση `StringTokenizer`. Το παρακάτω τμήμα κώδικα χωρίζει μια φράση στις λέξεις από τις οποίες αποτελείται:

```
String text = "This is my first Java program";
StringTokenizer st = new StringTokenizer(text);

for (int i = 1 ; st.hasMoreTokens() ; i++) {
   String word = st.nextToken();

   System.out.println("Word " + i + ": " + word);
}
```

Η αρχικοποίηση ενός αντικειμένου τύπου `StringTokenizer` γίνεται με μια συμβολοσειρά. Η μέθοδος `nextToken` επιστρέφει το επόμενο κάθε φορά κομμάτι, ενώ με τη μέθοδο `hasMoreTokens` ελέγχουμε την ύπαρξη άλλων κομματιών. Είναι δυνατό να ξέρουμε από την αρχή πόσα κομμάτια βρέθηκαν με τη μέθοδο `countTokens`. Αν δεν ορίσουμε τα διαχωριστικά σύμβολα, όπως έγινε στο παραπάνω παράδειγμα, υπονοείται το κενό. Μπορούμε όμως να ορίσουμε και δικά μας διαχωριστικά στον κατασκευαστή της κλασης `StringTokenizer`:

```
String path = "/usr/local/X11/lib/fonts/misc";
StringTokenizer st = new String Tokenizer(path, "/");
int i = st.countTokens(); // this will be 6
```

5.2. Μαθηματικές συναρτήσεις

Οι απλές αριθμητικές πράξεις μεταξύ των βασικών αριθμητικών τύπων γίνονται απευθείας στη γλώσσα. Πράξεις που δεν είναι δυνατό να γίνουν, προκαλούν μια αριθμητική εξαίρεση του τύπου `ArithmeticException`:

```
try {
   int i = 72 / 0;
}
catch (ArithmeticException e) {
   System.out.println("Division by zero");
}
```

Στις πράξεις μεταξύ αριθμών κινητής υποδιαστολής (`float` και `double`) δεν προκαλούνται εξαιρέσεις, αλλά προκύπτουν ως αποτελέσματα οι παρακάτω ειδικές τιμές:

`POSITIVE_INFINITY`	Θετικό άπειρο (+∞), π.χ. `1.0/0.0`
`NEGATIVE_INFINITY`	Αρνητικό άπειρο (−∞), π.χ. `(-1.0)/0.0`
`NaN`	Αόριστη τιμή (Not a Number), π.χ. `0.0/0.0`

Οι τιμές αυτές ορίζονται ως σταθερές στις κλάσεις `Float` και `Double`, που θα περιγραφούν στη συνέχεια. Η τιμή `NaN` έχει την ειδική ιδιότητα ότι δεν είναι ίση με τον εαυτό της, δηλαδή `NaN != Nan`. Ο έλεγχος αν μια τιμή είναι `NaN` γίνεται με ειδική μέθοδο `isNan`.

Η κλάση `Math`, που ορίζεται στο πακέτο `java.lang`, χρησιμεύει ως μαθηματική βιβλιοθήκη. Όλες οι μέθοδοί της είναι στατικές και η κατασκευή μεταβλητών τύπου `Math` δεν είναι δυνατή. Στον Πίνακα 5.1 δίνεται ένας κατάλογος των μαθηματικών συναρτήσεων, που υλοποιούνται ως μέθοδοι της `Math`. Στην ίδια κλάση επίσης ορίζονται και οι βασικές μαθηματικές σταθερές `PI` και `E`.

***Πίνακας 5.1. Μέθοδοι της κλάσης* `Math`.**

Μέθοδος	***Τύπος ορίσματος***	***Περιγραφή***

`abs(a)`
`max(a, b)`
`min(a, b)`
	`int, long, float, double`	Απόλυτη τιμή.
		Μέγιστος και ελάχιστος δυο αριθμών

`sqrt(a)`	`double`	Τετραγωνική ρίζα: $\sqrt{a}$
`pow(a, b)`	`double`	Ύψωση σε δύναμη: a^b
`exp(a)`	`double`	Εκθετική συνάρτηση: e^a
`log(a)`	`double`	Φυσικός λογάριθμος: $\ln a$

`sin(a)`
`cos(a)`
`tan(a)`
	`double`	Τριγωνομετρικές συναρτήσεις:
		$\sin a$, $\cos a$ και $\tan a$

`acos(a)`
`asin(a)`
`atan(a)`
	`double`	Αντίστροφες τριγωνομετρικές συναρτήσεις:
		$\sin^{-1} a$, $\cos^{-1} a$ και $\tan^{-1} a$

`atan2(y, x)`	`double`	Μετατροπή καρτεσιανών σε πολικές συντεταγμένες: $\tan^{-1}\left(y/x\right)$
`ceil(a)` `floor(a)`	`double`	Ο μεγαλύτερος ακέραιος που δεν υπερβαίνει το a και ο μικρότερος ακέραιος που δεν είναι μικρότερος του a, αντίστοιχα.
`round(a)`	`float, double`	Στρογγυλοποίηση σε ακέραιο, με ακέραιο αποτέλεσμα.
`rint(a)`	`double`	Στρογγυλοποίηση σε ακέραιο, με αποτέλεσμα κινητής υποδιαστολής.
`random()`		Γεννήτρια τυχαίων αριθμών.

5.2.1. Περιτυλίγματα βασικών τύπων

Σε μια αγνή γλώσσα αντικειμενοστρεφούς προγραμματισμού, τα αριθμητικά δεδομένα είναι αντικείμενα που προέρχονται από ειδικές αριθμητικές κλάσεις. Στη Java, όπως έχει ήδη αναφερθεί, τα αριθμητικά δεδομένα ανήκουν στους βασικούς αριθμητικούς τύπους, για λόγους ταχύτητας εκτέλεσης των προγραμμάτων. Μερικές φορές, όμως, είναι χρήσιμο να μεταχειρίζεται κανείς τα δεδομένα των βασικών τύπων ως αντικείμενα. Για το λόγο αυτό, στο πακέτο `java.lang` ορίζονται κλάσεις αντικειμένων που μπορούν να χρησιμοποιηθούν αντί των βασικών τύπων. Οι κλάσεις αυτές ονομάζονται *περιτυλίγματα* (wrappers) και παρέχουν μεθόδους μετατροπής από και προς τους βασικούς τύπους.

Τα ονόματα των κλάσεων περιτυλιγμάτων προκύπτουν από τα ονόματα των βασικών τύπων, με τη μετατροπή του πρώτου γράμματος από μικρό σε κεφαλαίο. Για παράδειγμα η κλάση περιτυλίγματος του βασικού τύπου `boolean` ονομάζεται `Boolean`. Εξαίρεση στην ονοματολογία αποτελεί η κλάση `Integer`, που είναι το περιτύλιγμα του τύπου `int`.

Τα αντικείμενα που προέρχονται από τις κλάσεις περιτυλίγματα, όπως και τα αντικείμενα τύπου `String`, δεν είναι δυνατό να μεταβληθούν μετά τη δημιουργία τους. Δημιουργούνται με προφανή τρόπο, είτε από δεδομένα κατάλληλων βασικών τύπων, είτε από συμβολοσειρές:

```
Float pi = new Float(3.14);
Integer n = new Integer("-153");
```

Σε περίπτωση αδυναμίας της μετατροπής από συμβολοσειρά σε αριθμητικό τύπο, προκαλείται εξαίρεση τύπου `NumberFormatException`:

```
try {
   Double bogus = new Double("tralala");
}
catch (NumberFormatException e) {
   System.out.println("Wrong format!");
}
```

Μια άμεση χρήση των κλάσεων περιτυλιγμάτων είναι, όπως ήδη αναφέρθηκε, η μετατροπή μιας συμβολοσειράς σε αριθμό:

```
String five = "5";
int n = new Integer(five).intValue();
```

Ειδικά για μετατροπές από συμβολοσειρές σε ακεραίους, οι κλάσεις `Integer` και `Long`, περιέχουν τις στατικές μεθόδους `parseInt` και `parseLong`. Έτσι το παραπάνω παράδειγμα μπορεί ισοδύναμα να γραφεί ως:

```
String file = "5";
int n = Integer.parseInt(five);
```

Άλλες μετατροπές μεταξύ των κλάσεων περιτυλιγμάτων και βασικών τύπων μπορούν με τον ίδιο τρόπο να πραγματοποιηθούν, για παράδειγμα:

```
Double number = new Double(12.34);

double d = number.doubleValue();
float  f = number.floatValue();
```

```
long   l = number.longValue();
int    i = number.intValue();
```

Μια άλλη χρήση των κλάσεων περιτυλιγμάτων, όπως θα περιγραφεί στη συνέχεια του κεφαλαίου, είναι για τη συλλογή δεδομένων σε αντικείμενα. Οι κλάσεις συλλογής απαιτούν τα στοιχεία που συλλέγουν να είναι αντικείμενα και όχι δεδομένα των βασικών τύπων.

5.2.2. Γεννήτρια τυχαίων αριθμών

Για την παραγωγή τυχαίων αριθμών, το API της Java παρέχει την κλάση `Random`, που ορίζεται στο πακέτο `java.util`. Πρόκειται για μια απλή γεννήτρια τυχαίων αριθμών που αρχικοποιείται από έναν αριθμό 48 bit. Αν δεν ορίσθεί αρχική τιμή, χρησιμοποιείται για το σκοπό αυτό η τρέχουσα ώρα του συστήματος:

```
Random r1 = new Random();

long seed = mySeed;
Random r2 = new Random(seed);
```

Με τις ακόλουθες μεθόδους η γεννήτρια παράγει ψευδοτυχαίους αριθμούς με ομοιόμορφη κατανομή:

```
r1.nextInt()      // -(2^31) ως (2^31-1)
r1.nextLong()     // -(2^63) ως (2^63-1)
r1.nextFloat()    // -1.0 ως 1.0
r1.nextDouble()   // -1.0 ως 1.0
```

Η μέθοδος `nextGaussian` παράγει ψευδοτυχαίους αριθμούς, που ακολουθούν κατανομή Gauss με μέση τιμή `0.0` και απόκλιση `1.0`. Τέλος, στην κλάση `Math` ορίζεται μια αρχικοποιημένη γεννήτρια τυχαίων αριθμών, που επιστρέφει ψευδοτυχαίους αριθμούς τύπου `double` μέσω της μεθόδου `random`.

5.3. Ημερομηνίες

Το πακέτο `java.util` παρέχει την κλάση `Date` για τη διευκόλυνση της διαχείρισης ημερομηνιών και ωρών. Ο κατασκευαστής χωρίς παραμέτρους αυτής της κλάσης κατασκευάζει ένα αντικείμενο που παριστάνει την τρέχουσα ημερομηνία και ώρα. Αντικείμενα με διαφορετική ημερομηνία και ώρα κατασκευάζονται με χρήση άλλων κατασκευαστών:

```
// Wed Jan 15 16:00:01 EET 1997 (τώρα)

Date now = new Date();

// Fri Jan 17 0:00:00 EET 1997

Date stAntonios = new Date(97, 0, 17);

// Fri Jun 14 15:00:00 EET 1997

Date birthday = new Date(97, 5, 14, 15, 00, 00);
```

Για την κατασκευή αντικειμένων τύπου `Date`, αξιοσημείωτα είναι τα ακόλουθα:

- Αν δεν ορίσουμε συγκεκριμένη ώρα, εννοούνται τα μεσάνυχτα (0:00:00).
- Τα έτη χαρακτηρίζονται ως χρόνια μετά το 1900. Έτσι το 1997 δίνεται ως `97` και το 2001 ως `101`.
- Όλες οι τιμές ξεκινούν από `0`, εκτός από την ημέρα του μήνα, που ξεκινά από `1`. Έτσι ο Ιανουάριος είναι `0`, ο Φεβρουάριος `1` και ο Ιούνιος `5`.
- Ανύπαρκτες ή εσφαλμένες ημερομηνίες μετατρέπονται αριθμητικά σε σωστές.

Λόγω του ότι οι κατασκευαστές της κλάσης `Date` που αναφέρθηκαν πιο πάνω ίσως να φαίνονται λίγο δύσχρηστοι, λόγω των πολλών ιδιαιτεροτήτων τους, η κλάση `Date` υποστηρίζει μετατροπή από συμβολοσειρά σε ημερομηνία, μέσω ενός ακόμα κατασκευαστή. Η συμβολοσειρά αυτή περιγράφει μια ημερομηνία και ώρα σε αρκετά ελεύθερη μορφή, όπως φαίνεται πιο κάτω:

```
Date a = new Date("Jan 15, 1997");
Date b = new Date("19 Dec 1997");
Date c = new Date("Fri, 19 Dec 1997 13:30:00");
Date d = new Date("Fri, 19 Dec 1997 13:30:00 EET");
Date e = new Date("19 Dec 1997 13:30:00 GMT+0600");
```

Η Java μετρά τον χρόνο με ακρίβεια χιλιοστών του δευτερολέπτου, από τα μεσάνυχτα της 1ης Ιανουαρίου 1970 GMT, όπως περίπου το Unix. Υπάρχει λοιπόν η δυνατότητα να καθορισθεί μια ημερομηνία με αυτή τη μορφή:

```
Date unixStart = new Date(0L);
```

Η τρέχουσα ώρα στην παραπάνω μορφή δίνεται από την στατική μέθοδο `currentTimeMillis` της κλάσης `System`. Ο Πίνακας 5.2 περιγράφει τις σημαντικότερες μεθόδους της κλάσης `Date`.

***Πίνακας 5.2. Μέθοδοι της κλάσης* `Date`.**

Μέθοδος	***Περιγραφή***

`getYear` `setYear`
`getMonth` `setMonth`
`getDate` `setDate`
	Διάβασμα και αλλαγή του έτους, μήνα και ημέρας του μήνα.

`getDay` `setDay`	Διάβασμα και αλλαγή της ημέρας της εβδομάδας (0 = Κυριακή)

`getHours` `setHours`
`getMinutes` `setMinutes`
`getSeconds` `setSeconds`
	Διάβασμα και αλλαγή της ώρας, των λεπτών και των δευτερολέπτων.

`getTime` `setTime`	Διάβασμα και αλλαγή του χρόνου σε απόλυτη μορφή (όπως στο Unix).
`before` `after` `equals`	Σύγκριση ημερομηνιών.

5.4. Συλλογές δεδομένων

Οι *κλάσεις συλλογής δεδομένων* (collection classes) ορίζονται στο πακέτο java.util και χρησιμοποιούνται για την κατασκευή αποθηκών δεδομένων. Κάθε τέτοια κλάση υλοποιεί αντικείμενα που συνήθως διαφέρουν στον τρόπο ανάκτησης της πληροφορίας. Σημαντικότερες

από τις κλάσεις συλλογής δεδομένων είναι οι δυναμικοί πίνακες και οι πίνακες κατακερματισμού, που περιγράφονται στις επόμενες ενότητες..

Σύμφωνα με τα πρότυπα του αντικειμενοστρεφούς προγραμματισμού, όλες οι κλάσεις συλλογής δεδομένων αποθηκεύουν αντικείμενα και, σύμφωνα με την αρχή της αφαίρεσης (data abstraction), αυτά δεν είναι απαραίτητο να προέρχονται όλα από την ίδια κλάση. Έτσι, μια αποθήκη δεδομένων μπορεί να περιέχει σε τυχαία σειρά αντικείμενα αριθμών, συμβολοσειρών ή άλλων πιο πολύπλοκων κλάσεων. Ωστόσο, είναι ευθύνη του προγραμματιστή να ξέρει τον τύπο κάθε αντικειμένου.

5.4.1. Δυναμικοί πίνακες

Η κλάση `Vector` υλοποιεί δυναμικούς πίνακες, δηλαδή πίνακες των οποίων το μέγεθος μπορεί να μεταβάλλεται δυναμικά. Η χρήση της κλάσης `Vector` είναι απλή. Ο κατασκευαστής χωρίς παραμέτρους κατασκευάζει κενούς δυναμικούς πίνακες, στους οποίους μπορούν να προστεθούν στοιχεία με τις μεθόδους `addElement` και `insertElementAt`, όπως φαίνεται στο ακόλουθο παράδειγμα:

```
Vector things = new Vector();

String  one   = "one";
String  two   = "two";
Integer three = new Integer(3);

things.addElement(one);
things.addElement(three);
things.insertElementAt(two,1);

// things now contains: "one", "two", 3
```

Οι μέθοδοι `elementAt`, `firstElement` και `lastElement` επιστρέφουν αντικείμενα που περιέχονται σε ένα δυναμικό πίνακα, χωρίς να τα εξάγουν από αυτόν. Προκειμένου αυτά τα αντικείμενα να χρησιμοποιηθούν, απαιτείται συνήθως η μετατροπή των τύπων τους, αφού ο ίδιος ο πίνακας δε γνωρίζει τον τύπο των αντικειμένων που περιέχει:

```
String  s1 = (String) things.firstElement();   // "one"
String  s2 = (String) things.elementAt(1);     // "two"
Integer n3 = (Integer) things.lastElement();   // 3
```

Επίσης, πρέπει να σημειωθεί ότι δεν είναι δυνατή η εισαγωγή δεδομένων που ανήκουν στους βασικούς τύπους. Αντί αυτών πρέπει να χρησιμοποιούνται οι κλάσεις περιτυλίγματος.

Η κλάση `Vector` παρέχει επίσης τη δυνατότητα αναζήτησης και διαγραφής στοιχείων, όπως φαίνεται στο ακόλουθο παράδειγμα:

```
int i = things.indexOf(three);    // 2
int j = things.indexOf("blah");   // -1: not there!

boolean f = things.contains(two);   // true

things.removeElement(one);

int num = things.size();   // 2

// things now contains: "two", 3
```

5.4.2. Απαριθμήσεις

Προκειμένου να διατρέξει κανείς όλα τα στοιχεία μιας κλάσης συλλογής, το πακέτο java.util παρέχει την κλάση `Enumeration`. Η κλάση αυτή υλοποιεί μια απαρίθμηση των στοιχείων μιας συλλογής, χωρίς να εγγυάται τη σειρά με την οποία αυτά θα εμφανίζονται. Παρέχει τις μεθόδους `nextElement` και `hasMoreElements`, με τη χρήση των οποίων είναι δυνατό να διατρέξει κανείς όλα τα στοιχεία, ακριβώς μια φορά. Δεν υπάρχει τρόπος επιστροφής στην αρχή.

Το ακόλουθο παράδειγμα χρησιμοποιεί ένα αντικείμενο της κλάσης Enumeration για την εκτύπωση όλων των στοιχείων ενός δυναμικού πίνακα. Η μέθοδος `elements` της κλάσης `Vector` επιστρέφει την απαρίθμηση όλων των στοιχείων της συλλογής:

```
Enumeration e = things.elements();

while (e.hasMoreElements()) {
   String s = (String) e.nextElement();

   System.out.println(s);
}
```

5.4.3. Πίνακες κατακερματισμού

Οι *πίνακες κατακερματισμού* (hashtables) μοιάζουν με λεξικά. Υλοποιούνται από την κλάση `HashTable` και χρησιμοποιούνται για την αντιστοίχιση αντικειμένων, που ονομάζονται *κλειδιά* (keys), σε άλλα αντικείμενα, που ονομάζονται *τιμές* (values). Συνηθέστερα χρησιμοποιούνται για την αντιστοίχιση συμβολοσειρών σε συμβολοσειρές. Το ακόλουθο παράδειγμα υλοποιεί ένα πολυ απλό λεξικό, από τα αγγλικά στα γαλλικά:

```
Hashtable dict = new Hashtable();

dict.put("Yes", "Oui");
dict.put("No", "Non");
dict.put("Love", "Amour");
```

Κάθε τύπου αντικείμενα μπορούν να χρησιμοποιηθούν στους πίνακες κατακερματισμού:

```
Hashtable dates = new Hashtable();

dates.put("Christmas", new Date("25 Dec 97"));
dates.put("Easter", new Date("27 Apr 97"));
dates.put("St. Antonios", new Date("17 Jan 97"));
```

Κάθε κλειδί μπορεί να αντιστοιχεί σε μια το πολύ τιμή. Αν προσπαθήσει κανείς να εισαγάγει ένα κλειδί δύο φορές, τότε η πρώτη αντιστοίχιση χάνεται.

Με τη μέθοδο `get` επιστρέφεται η τιμή που αντιστοιχεί σε ένα κλειδί, όπως φαίνεται στο ακόλουθο παράδειγμα. Η μέθοδος `get` επιστρέφει `null` αν το κλειδί δεν αντιστοιχεί σε κάποια τιμή.

```
Date d = (Date) dates.get("Easter");
```

Με τις μεθόδους `remove` και `containsKey` μπορεί κανείς να αφαιρέσει μια αντιστοίχιση από έναν πίνακα κατακερματισμού και να ελέγξει την ύπαρξη αντιστοίχισης για ένα κλειδί:

```
dates.remove("Christmas");
if (dict.containsKey("Love"))
   System.out.println(dict.get("Love"));
```

Τέλος, η κλάση `Hashtable` παρέχει δύο μεθόδους `keys` και `elements`, που επιστρέφουν απαριθμήσεις των κλειδιών και των τιμών ενός πίνακα κατακερματισμού. Το παρακάτω τμήμα κώδικα εκτυπώνει το μικρό λεξικό:

```
Enumeration e = dict.keys();

while (e.hasMoreElements()) {
   String key   = (String) e.nextElement();
   String value = (String) dict.get(key);

   System.out.println(key + ": " + value);
}
```

5.4.4. Πίνακας ιδιοτήτων

Η κλάση `Properties`, που ορίζεται στο πακέτο `java.util`, είναι μια ειδική περίπτωση πίνακα κατακερματισμού που περιέχει συμβολοσειρές. Χρησιμεύει για την αποθήκευση πληροφοριών για τις παραμέτρους της εφαρμογής, κάτι που σε άλλες γλώσσες γίνεται με τη χρήση *μεταβλητών περιβάλλοντος* (environment variables). Τα στοιχεία που περιέχει ένα αντικείμενο τύπου `Properties` ονομάζονται *ιδιότητες*. Η χρήση της κλάσης είναι εντελώς όμοια με αυτή των πινάκων κατακερματισμού, με τη διαφορά ότι η ανάκληση των στοιχείων γίνεται με τη μέθοδο `getProperty`:

```
Properties p = new Properties();

p.put("myApp.xsize", "100");
p.put("myApp.ysize", "200");

String xsize    = p.getProperty("myApp.xsize");
String notfound = p.getProperty("dummy");       // null
```

Πολύ χρήσιμο στοιχείο των `Properties` είναι η χρησιμοποίηση εξ ορισμού τιμών (default values). Είναι δυνατή η κατασκευή ενός αντικειμένου τύπου `Properties`, βασισμένου πάνω σε ένα άλλο αντικείμενο, που περιέχει τις εξ' ορισμού τιμές. Αν αναζητηθεί μια τιμή που δεν υπάρχει στο νέο αντικείμενο, θα αναζητηθεί αυτόματα στο αντικείμενο βάσης:

```
Properties defaults = new Properties();

defaults.put("length", "50");
defaults.put("width", "100");
defaults.put("height", "100");

Properties p = new Properties (defaults);

p.put("weight", "20");
p.put("length", "20");

System.out.println(p.getProperty("weight")); // 20
System.out.println(p.getProperty("length")); // 20
System.out.println(p.getProperty("width"));  // 50
```

Τα αντικείμενα τύπου `Properties` είναι δυνατό να αποθηκευθούν σε αρχεία, και στη συνέχεια να ανακτηθούν από αυτά. Αυτό επιτυγχάνεται με τις μεθόδους `load` και `save`, που αποθηκεύουν τις ιδιότητες σε μορφή κειμένου. Περισσότερα για την είσοδο και έξοδο δεδομένων καθώς και για τις λειτουργίες αρχείων θα αναφερθούν στο επόμενο κεφάλαιο.

```
FileInputStream fin;
...
Properties props = new Properties()
props.load(fin);
...
props.save(System.out, "Application Parameters");
```

Η κλάση `System`, που ορίζεται στο πακέτο `java.lang`, δίνει πρόσβαση στις βασικές πληροφορίες για τις ιδιότητες του συστήματος μέσω της στατικής μεθόδου `getProperty`, που επιστρέφει ένα αντικείμενο τύπου `Properties`. Ο Πίνακας 5.3 περιγράφει κλειδιά που υπάρχουν οπωσδήποτε σε αυτό το αντικείμενο.

Πίνακας 5.3. Ιδιότητες του συστήματος.

Όνομα ιδιότητας	***Περιγραφή***
`java.vendor`	Όνομα του κατασκευαστή της υλοποίησης της Java.
`java.vendor.url`	URL του κατασκευαστή.
`java.version`	Έκδοση της υλοποίησης της Java.
`java.home`	Directory εγκατάστασης της Java.
`java.class.version`	Έκδοση της κλάσης της Java.
`java.class.path`	Μονοπάτι εύρεσης κλάσεων.
`os.name`	Όνομα του λειτουργικού συστήματος.
`os.arch`	Τύπος της αρχιτεκτονικής του υπολογιστή.
`os.version`	Έκδοση του λειτουργικού συστήματος.
`file.separator`	Χαρακτήρας διαχωρισμού αρχείων (π.χ. "/" ή "\").
`path.separator`	Χαρακτήρας διαχωρισμού directories στο μονοπάτι.
`line.separator`	Χαρακτήρας τέλους γραμμής (π.χ. "\n" ή "\r\n").
`user.name`	Όνομα χρήστη.
`user.home`	Προσωπικό directory του χρήστη.
`user.dir`	Τρέχον directory του χρήστη.

6. Είσοδος και έξοδος

Στο κεφάλαιο αυτό συνεχίζεται η εξερεύνηση του API της Java με τις κλάσεις που υλοποιούν τις λειτουργίες εισόδου και εξόδου. Οι κλάσεις αυτές ορίζονται στο πακέτο java.io.

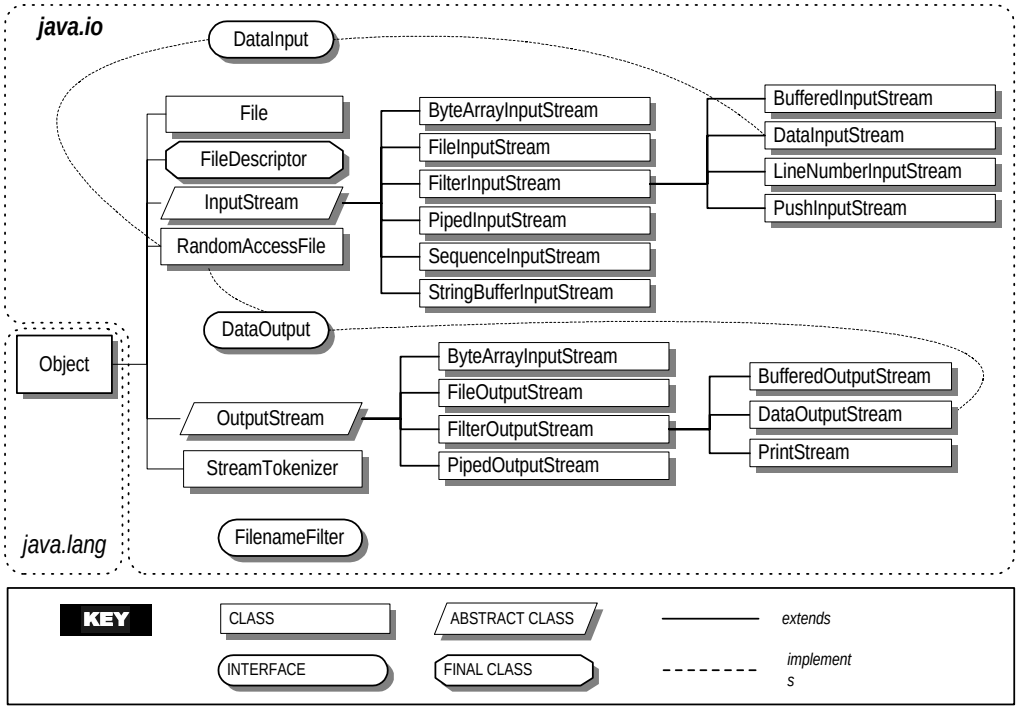
6.1. Εισαγωγή

Με τον όρο *είσοδος* (input) εννοούμε την εισαγωγή δεδομένων κατά την εκτέλεση ενός προγράμματος από τον εξωτερικό κόσμο. Με τον όρο *έξοδος* (output) εννοούμε την αντίστροφη διαδικασία, με την οποία τα αποτελέσματα του προγράμματος γίνονται γνωστά στον εξωτερικό κόσμο. Οι λειτουργίες της εισόδου και εξόδου χρησιμοποιούνται πολύ συχνά στα προγράμματα και συχνά αναφέρονται από κοινού με τη συντομογραφία Ε/Ε.

Το API της Java ορίζει ένα πλήθος από κλάσεις για την υλοποίηση των λειτουργιών Ε/Ε. Οι κλάσεις αυτές υποστηρίζουν ένα σημαντικό αριθμό μορφών εισόδου και εξόδου. Όπως φαίνεται στο Σχήμα 6.1, σχηματίζουν μια αρκετά δομημένη ιεραρχία, στην οποία οι περισσότερες κλάσεις αποτελούν υποκλάσεις των βασικών κλάσεων `InputStream` και `OutputStream`. Κάθε μια από αυτές τις κλάσεις έχει ένα πολύ συγκεκριμένο σκοπό και, παρόλο το μέγεθός του, το πακέτο `java.io` είναι αρκετά απλό στην κατανόηση και τη χρήση.

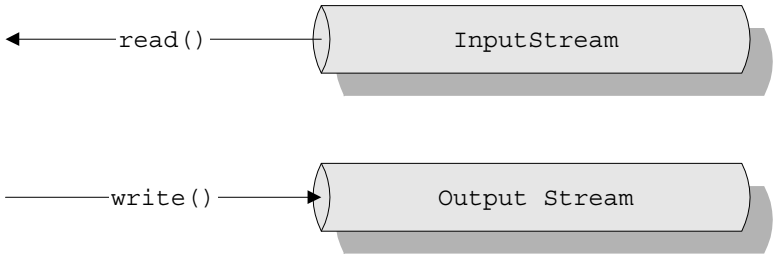
6.2. Ρεύματα

Στη Java, όλες οι θεμελιώδεις λειτουργίες Ε/Ε βασίζονται στα *ρεύματα* (streams). Ένα ρεύμα αντιπροσωπεύει μια ροή δεδομένων ή ένα κανάλι επικοινωνίας που, ιδεατά, έχει έναν συγγραφέα στο ένα του άκρο και έναν αναγνώστη στο άλλο. Οι λειτουργίες Ε/Ε ενός τερματικού, η ανάγνωση και εγγραφή αρχείων και η επικοινωνία μέσω θυρίδων απαιτούν φυσικά τη χρήση διαφορετικών τύπων ρευμάτων. Οι διαφορετικοί τύποι ρευμάτων φαίνονται περιληπτικά στον Πίνακα 6.1 και περιγράφονται στις αμέσως επόμενες παραγράφους.



Σχήμα 6.1. Κλάσεις εισόδου-εξόδου στη Java.

Στη Java, τα ρεύματα μπορούν να παρομοιαστούν με δρόμους μιας κατεύθυνσης. Οι κλάσεις `InputStream` και `OutputStream` αντιπροσωπεύουν τα άκρα ενός απλού ρεύματος, όπως φαίνεται στο Σχήμα 6.2. Για αμφίδρομες επικοινωνίες χρησιμοποιούμε και τους δυο παραπάνω τύπους ρευμάτων.



***Σχήμα 6.2. Βασικές λειτουργίες των κλάσεων* `InputStream` *και* `OutputStream`.**

Οι κλάσεις `InputStream` και `OutputStream` είναι αφηρημένες και ορίζουν το κατώτατο επίπεδο διαπροσωπείας για όλα τα ρεύματα. Περιέχουν μεθόδους για την ανάγνωση και εγγραφή μιας ροής μη δομημένων δεδομένων, σε επίπεδο bytes. Το API της Java υλοποιεί υποκλάσεις αυτών των κλάσεων, για τη διεκπεραίωση εργασιών όπως η ανάγνωση και η εγγραφή αρχείων και η επικοινωνία μέσω θυρίδων. Λόγω του ότι όλα τα ρεύματα κληρονομούν τη δομή των `InputStream` και `OutputStream`, οι διάφοροι τύποι ρευμάτων μπορούν να χρησιμοποιούνται εναλλακτικά. Για παράδειγμα, μια μέθοδος δέχεται συχνά σαν όρισμα ένα `InputStream`. Αυτό σημαίνει ότι η μέθοδος δέχεται οποιαδήποτε υποκλάση του `InputStream`. Οι εξειδικευμένοι τύποι ρευμάτων μπορούν επίσης να διαστρωματωθούν ώστε

να παρέχουν υψηλότερου επιπέδου λειτουργικότητα, όπως δυνατότητα απομόνωσης και χειρισμό μεγαλύτερων τύπων δεδομένων.

Πίνακας 6.1. Συνοπτική περιγραφή των κλάσεων εισόδου-εξόδου.

Κλάση	***Περιγραφή***
`InputStream` `OutputStream`	Αφηρημένες κλάσεις που ορίζουν μεθόδους για την ανάγνωση και την εγγραφή δεδομένων με τη μορφή μη δομημένων ακολουθιών από bytes. Όλοι οι υπόλοιποι τύποι ρευμάτων της Java προέρχονται από τις δυο αυτές βασικές κλάσεις.
`DataInputStream` `DataOutputStream`	Εξειδικευμένα φίλτρα ρευμάτων, τα οποία δίνουν τη δυνατότητα ανάγνωσης και εγγραφής απλών τύπων δεδομένων, όπως στοιχειώδη αριθμητικά δεδομένα και αντικείμενα τύπου String.
`BufferedInputStream` `BufferedOutputStream`	Εξειδικευμένα φίλτρα ρευμάτων, τα οποία ενσωματώνουν τη λειτουργία της προσωρινής αποθήκευσης δεδομένων για επίτευξη μεγαλύτερης αποδοτικότητας.
`PrintStream`	Εξειδικευμένο φίλτρο που βοηθά στην εκτύπωση κειμένου.
`PipedInputStream` `PipedOutputStream`	Ρεύματα “διπλού-άκρου” πάντα σε ζεύγη. Τα δεδομένα που γράφονται σε ένα ρεύμα τύπου PipedOutputStream διαβάζονται από το αντίστοιχο PipedInputStream.
`FileInputStream` `FileOutputStream`	Υλοποιήσεις των ρευμάτων InputStream και OutputStream, με τις οποίες γίνεται η ανάγνωση και εγγραφή τοπικών αρχείων.

6.2.1. Απλή Ε/Ε

Η κοινή είσοδος (standard input) μιας εφαρμογής Java υλοποιείται μέσω του αντικειμένου `System.in`. Το αντικείμενο αυτό προέρχεται από υποκλάση της `InputStream`. Όπως το `stdin` στη C και το `cin` στη C++, το αντικείμενο αυτό διαβάζει δεδομένα από το περιβάλλον εκτέλεσης του προγράμματος, συνήθως από κάποιο τερματικό ή παράθυρο.

Με τη μέθοδο `read` της `InputStream` είναι δυνατή η ανάγνωση ενός byte κάθε φορά, από τη κοινή είσοδο. Η μέθοδος `read` παρέχει ένα byte πληροφορίας και επιστρέφει έναν ακέραιο αριθμό. Μια τιμή επιστροφής ίση με `-1` δηλώνει ότι επετεύχθη κανονικός τερματισμός του ρεύματος. Εάν προκύψει κάποιο λάθος κατά τη διάρκεια της ανάγνωσης, δημιουργείται μια εξαίρεση τύπου `IOException`. Όλες οι βασικές εντολές εισόδου και εξόδου των ρευμάτων μπορούν να προκαλέσουν `IOException` και για το λόγο αυτό ο προγραμματιστής πρέπει να φροντίζει για την ανίχνευση και τον κατάλληλο χειρισμό τους σε σχέση με τη συγκεκριμένη εφαρμογή. Για παράδειγμα, το παρακάτω τμήμα κώδικα διαβάζει μια ακολουθία των 256 bytes από την κοινή είσοδο:

```
try {
   byte [] data = new byte [256];
   int bytesRead = System.in.read(data);

   if (bytesRead < 256)
      System.out.println("Not enough bytes to read!");
   else
      // process the data now!
}
catch (IOException e) {
```

```
        System.out.println("Something went wrong!");
    }
```

Η `InputStream` παρέχει επίσης τη μέθοδο `available`, η οποία μας πληροφορεί για τον αριθμό των bytes που είναι διαθέσιμα προς ανάγνωση σε ένα `InputStream`, καθώς και τη μέθοδο `skip` σαν ένα τρόπο για την παράκαμψη ενός ορισμένου αριθμού bytes κατά τη διάρκεια της λειτουργίας ανάγνωσης. Η μέθοδος `close` κλείνει το ρεύμα και ελευθερώνει τους πόρους του συστήματος που σχετίζονται με αυτό. Για παράδειγμα, το επόμενο τμήμα κώδικα χρησιμοποιεί τις δυο πρώτες μεθόδους για να διαβάσει από την κοινή είσοδο, αγνοώντας τα πρώτα 200 bytes:

```
System.in.skip(200);

int numMoreBytes = System.in.available();

if (numMoreBytes > 0) {
   byte [] data = new byte [numMoreBytes];

   System.in.read(data);
}
```

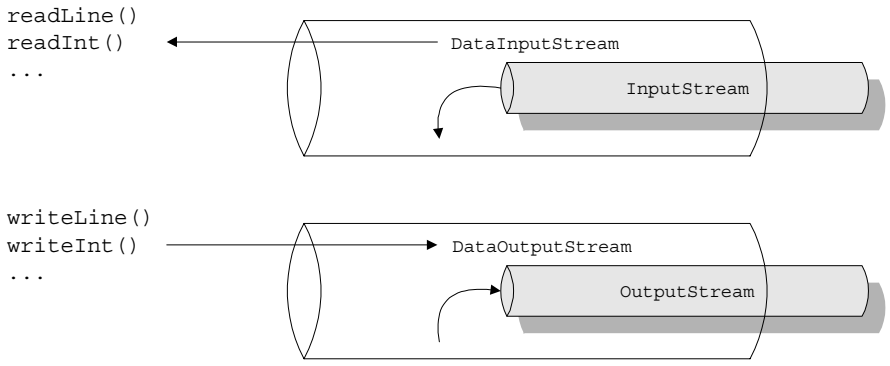
Τα αντικείμενα `System.out` και `System.err` υλοποιούν την κοινή έξοδο (standard output) του προγράμματος και την έξοδο σφαλμάτων (standard error output) αντίστοιχα. Τα αντικείμενα αυτά προέρχονται από κάποια υποκλάση της `PrintStream`, που κληρονομεί την `OutputStream`, και συνήθως εκτυπώνουν σε κάποιο τερματικό ή παράθυρο. Η λειτουργία των αντικειμένων τύπου `PrintStream` περιγράφεται αναλυτικά σε επόμενη παράγραφο.

6.2.2. Φιλτραρισμένα ρεύματα

Στην περίπτωση που είναι απαραίτητη η εγγραφή και ανάγνωση δομημένων δεδομένων, χρησιμοποιούνται οι κλάσεις `FilterInputStream` και `FilterOutputStream`. Οι κλάσεις αυτές παρέχουν τη βάση για τη δημιουργία ρευμάτων που περιβάλλουν άλλα ρεύματα και προσθέτουν νέα χαρακτηριστικά. Τα ρεύματα αυτά ονομάζονται *φιλτραρισμένα* ή *περιβάλλοντα ρεύματα* (stream wrappers). Ο κατασκευαστής ενός φιλτραρισμένου ρεύματος δέχεται ως παράμετρο ένα `InputStream` ή ένα `OutputStream`, μεταβιβάζει τις κλήσεις στο ρεύμα του εσωτερικού επιπέδου, ενώ ταυτόχρονα εκτελεί το ίδιο κάποια επιπρόσθετη επεξεργασία.

Οι κλάσεις `FilterInputStream` και `FilterOutputStream` δεν είναι από μόνες τους ιδιαίτερα χρήσιμες. Είναι συνήθως απαραίτητο να παραχθούν εξειδικευμένες υποκλάσεις των κλάσεων αυτών, με σκοπό τη δημιουργία μιας νέας λειτουργίας φιλτραρίσματος. Για παράδειγμα, εξειδικευμένα φιλτραρισμένα ρεύματα, όπως το `DataInputStream` και το `DataOutputStream`, παρέχουν επιπρόσθετες μεθόδους για την ανάγνωση και εγγραφή διάφορων τύπων δεδομένων.

Όπως προαναφέρθηκε, για τη δημιουργία ενός στιγμιότυπου κάποιου φιλτραρισμένου ρεύματος ορίζεται ένα `InputStream` ή ένα `OutputStream` στον κατασκευαστή. Το εξειδικευμένο ρεύμα περιβάλει κάποιο άλλο ρεύμα με ένα επιπρόσθετο επίπεδο λειτουργικότητας, όπως απεικονίζεται στο Σχήμα 6.3. Λόγω του ότι τα φιλτραρισμένα ρεύματα είναι με τη σειρά τους υποκλάσεις των θεμελιωδών τύπων `InputStream` και `OutputStream`, είναι δυνατή η χρήση πολλών φίλτρων σε διάφορα επίπεδα διαστρωμάτωσης ώστε να επιτευχθούν ποικίλοι συνδυασμοί χαρακτηριστικών.



Σχήμα 6.3. Διαστρωμάτωση ρευμάτων.

6.2.3. Ρεύματα Δεδομένων

Οι κλάσεις `DataInputStream` και `DataOutputStream` υλοποιούν φιλτραρισμένα ρεύματα που επιτρέπουν την ανάγνωση ή εγγραφή συμβολοσειρών και στοιχειωδών τύπων δεδομένων, που αποτελούνται από περισσότερα του ενός bytes. Τα ρεύματα αυτά ονομάζονται *ρεύματα δεδομένων* (data streams) και μπορούν να χρησιμοποιηθούν ως φίλτρα πάνω σε κάθε τύπου ρεύματα.

Οι κλάσεις `DataInputStream` και `DataOutputStream` υλοποιούν αντίστοιχα δυο διαπροσωπείες με ονόματα `DataInput` και `DataOutput`, οι οποίες καθορίζουν τις μεθόδους ρευμάτων ανάγνωσης και εκτύπωσης συμβολοσειρών και στοιχειωδών τύπων της Java, με τρόπο που εξαρτάται από τη μηχανή.

Ένα αντικείμενο τύπου `DataInputStream` μπορεί να κατασκευαστεί απευθείας από ένα τύπου `InputStream`. Κατόπιν, μπορεί να χρησιμοποιηθεί μια μέθοδος όπως η `readLine` για την ανάγνωση μιας συμβολοσειράς δεδομένων:

```
DataInputStream s = new DataInputStream(System.in);
String line = s.readlLine();
```

Στο παραπάνω παράδειγμα, το `DataInputStream` περιβάλλει το ρεύμα κοινής εισόδου και το χρησιμοποιεί για την ανάγνωση μιας γραμμής κειμένου. Η μέθοδος `readLine` διαβάζει bytes έως ότου συναντήσει το χαρακτήρα `<CR>`, το χαρακτήρα `<LF>` ή συνδυασμό τους.

Η ανάγνωση ακέραιων τιμών από το ρεύμα γίνεται με τη μέθοδο `readInt`:

```
int i = s.readInt();
```

Η μέθοδος αυτή διαβάζει τέσσερα bytes πληροφορίας από το ρεύμα και τα μετατρέπει σε έναν ακέραιο 32-bit. Όλες οι μέθοδοι του `DataInputStream` που διαβάζουν στοιχειώδεις τύπους μπορούν να διαβάσουν και δυαδική πληροφορία.

Η κλάση `DataOutputStream` παρέχει μεθόδους εγγραφής που αντιστοιχούν στις μεθόδους ανάγνωσης της `DataInputStream`. Για παράδειγμα, η μέθοδος `writeInt` γράφει έναν ακέραιο αριθμό σε δυαδική μορφή στο ρεύμα εξόδου του εσωτερικού στρώματος. Αντικείμενα

τύπου `DataOutputStream` μπορούν να κατασκευαστούν απευθείας από αντικείμενα τύπου `OutputStream`.

6.2.4. Ρεύματα με προσωρινή αποθήκευση

Οι κλάσεις `BufferedInputStream` και `BufferedOutputStream` προσαρτούν μια αποθήκη δεδομένων συγκεκριμένου μεγέθους σε κάποιο ρεύμα. Τα ρεύματα που προκύπτουν κατ' αυτό τον τρόπο ονομάζονται *ρεύματα με προσωρινή αποθήκευση* (buffered streams). Η ύπαρξη προσωρινών αποθηκών μπορεί να αυξήσει την αποδοτικότητα, μειώνοντας τον αριθμό των λειτουργιών φυσικής ανάγνωσης και εγγραφής που αντιστοιχούν σε κλήσεις των μεθόδων `read` και `write`. Ένα ρεύμα με προσωρινή αποθήκευση δημιουργείται από το κατάλληλο ρεύμα εισόδου ή εξόδου και μια αποθήκη κάποιου μεγέθους. Επιπλέον, άλλα ρεύματα μπορούν να περιβάλλουν τα ρεύματα με προσωρινή αποθήκευση και να επωφελούνται από αυτή την ιδιότητα.

Ένα τυπικό ρεύμα με προσωρινή αποθήκευση δημιουργείται ως εξής:

```
BufferedInputStream myBufferedStream =
   new BufferedInputStream(myInputStream, 4096);

myBufferedStream.read();
```

Στο παράδειγμα αυτό, καθορίζεται μια προσωρινή αποθήκη μεγέθους 4096 bytes. Αν δεν οριστεί το μέγεθος της αποθήκης στον κατασκευαστή, χρησιμοποιείται μια τιμή ορισμένη από το σύστημα.

6.2.5. Ρεύματα εκτύπωσης (Print Streams)

Μια πολύ χρήσιμη κλάση που υλοποιεί ένα φιλτραρισμένο ρεύμα είναι η `PrintStream`. Η κλάση αυτή παρέχει ένα σύνολο υπερφορτωμένων μεθόδων `print` που μετατρέπουν τα ορίσματά τους σε συμβολοσειρές και τα ωθούν προς το ρεύμα. Μια σημαντική διαφορά μεταξύ του `PrintStream` και των άλλων ρευμάτων είναι ότι, ακόμα και αν προκύψει σφάλμα εισόδου-εξόδου, καμιά από τις μεθόδους δεν προκαλεί `IOException`. Για τον έλεγχο των λαθών παρέχεται η μέθοδος `checkError`.

6.2.6. Αγωγοί

Συνήθως, σε κάποια δεδομένη χρονική στιγμή, οι διάφορες εφαρμογές εμπλέκονται άμεσα με το ένα μόνο άκρο ενός ρεύματος. Μερικές φορές όμως είναι χρήσιμο να ελέγχονται και τα δυο άκρα. Οι κλάσεις `PipedInputStream` και `PipedOutputStream` επιτρέπουν τη διασύνδεση δυο ρευμάτων, έτσι ώστε το ένα άκρο του ενός να συνδέεται με το ένα άκρο του άλλου. Μια τέτοια δομή ονομάζεται *αγωγός* (pipe).

Για τη δημιουργία ενός αγωγού απαιτείται να κατασκευασθούν δυο αντικείμενα, το ένα τύπου `PipedInputStream` και το άλλο τύπου `PipedOutputStream`. Αρχικά κατασκευάζεται το ένα άκρο και στη συνέχεια το δεύτερο, χρησιμοποιώντας ως όρισμα στον κατασκευαστή το πρώτο:

```
PipedInputStream  pin  = new PipedInputStream();
PipedOutputStream pout = new PipedOutputStream(pin);
```

ή εναλλακτικά:

```
PipedOutputStream pout = new PipedOutputStream();
PipedInputStream  pin  = new PipedInputStream(pout);
```

Για κάθε ένα από τα παραπάνω παραδείγματα το αποτέλεσμα είναι η παραγωγή ενός ρεύματος εισόδου `pin` και ενός ρεύματος εξόδου `pout`, που συνδέονται μεταξύ τους. Τα δεδομένα που γράφονται στο `pout` μπορούν να διαβαστούν από το `pin`. Είναι επίσης δυνατό τα `PipedInputStream` και `PipedOutputStream` να δημιουργηθούν ξεχωριστά και κατόπιν να συνδεθούν με χρήση της μεθόδου `connect`.

Από τη στιγμή που τα δυο άκρα ενός αγωγού συνδεθούν μεταξύ τους, τα δυο ρεύματα χρησιμοποιούνται με τον ίδιο τρόπο που χρησιμοποιούνται και τα άλλα ρεύματα εισόδου και εξόδου. Η μέθοδος `read` χρησιμοποιείται για την ανάγνωση δεδομένων από το `PipedInputStream` και η `write` για την εγγραφή δεδομένων στο `PipedOutputStream`. Φυσικά, συνήθως αυτά τα ρεύματα χρησιμοποιούνται μέσω φίλτρων. Εάν γεμίσει ο εσωτερικός αποθηκευτικός χώρος του αγωγού, μπλοκάρεται η λειτουργία εγγραφής, η οποία και μπαίνει σε αναμονή έως ότου προκύψει διαθέσιμος χώρος. Αντίστροφα, αν ο αγωγός είναι άδειος, μπλοκάρεται η λειτουργία ανάγνωσης έως ότου εμφανιστούν δεδομένα προς ανάγνωση.

6.3. Αρχεία

Με εξαίρεση ορισμένους περιορισμούς για λόγους ασφαλείας του συστήματος, οι εφαρμογές Java μπορούν να διαβάσουν και να γράψουν αρχεία που βρίσκονται στο τοπικό σύστημα αρχείων, χρησιμοποιώντας τα προνόμια πρόσβασης του χρήστη που τις εκτελεί. Το API της Java ορίζει την κλάση `File` προκειμένου να υλοποιηθούν αντικείμενα που απεικονίζουν αυτά τα αρχεία. Η κλάση αυτή αποτελεί απλώς μέσο πρόσβασης στα αρχεία του συστήματος. Δεν παρέχει άμεσα λειτουργίες Ε/Ε: υπάρχουν εξειδικευμένα ρεύματα για το σκοπό αυτό, που θα περιγραφούν σε επόμενη παράγραφο.

6.3.1. Διαχείριση αρχείων

Το πρώτο πράγμα που ορίζει κανείς, προκειμένου να διαχειρισθεί ένα αρχείο του τοπικού συστήματος, είναι ένα αντικείμενο τύπου `File`. Υπάρχουν διάφοροι κατασκευαστές τέτοιων αντικειμένων, σημαντικότεροι από τους οποίους είναι οι δυο που χρησιμοποιούνται στο ακόλουθο παράδειγμα:

```
File f1 = new File("/one/path/or/another/file.txt");

File f2 = new File("/one/path/or/another", "file.txt");
```

Στον πρώτο καθορίζεται το πλήρες όνομα ενός αρχείου, ενώ στο δεύτερο καθορίζεται το όνομα του αρχείου και το directory στο οποίο βρίσκεται. Σημειώνεται ότι τα αντικείμενο τύπου `File` κατασκευάζεται πάντοτε, ακόμα και αν τα αρχεία που καθορίζονται δεν υπάρχουν.

Μόλις κατασκευασθεί ένα τέτοιο αντικείμενο, είναι δυνατή η επεξεργασία του με τη βοήθεια μεθόδων που παρέχει η κλάση `File`. Οι μέθοδοι αυτές περιγράφονται στον Πίνακα 6.2.

6.3.2. Ρεύματα αρχείων

Η Java παρέχει δυο εξειδικευμένα ρεύματα για την ανάγνωση και την εγγραφή αρχείων: τα `FileInputStream` και `FileOutputStream`. Τα ρεύματα αυτά παρέχουν τη βασική λειτουργικότητα των `InputStream` και `OutputStream`, προσαρμοσμένη στην ανάγνωση και την εγγραφή δεδομένων που περιέχονται σε αρχεία. Τα ρεύματα αρχείων μπορούν επίσης να συνδυαστούν με φιλτραρισμένα ρεύματα, ώστε να επιτυγχάνονται αποτελέσματα παρόμοια με εκείνα που προαναφέρθηκαν για τα υπόλοιπα ρεύματα επικοινωνίας.

Ένα `FileInputStream` μπορεί να δημιουργηθεί με βάση μια συμβολοσειρά, που περιγράφει το όνομα του αρχείου, ή ένα αντικείμενο `File`, όπως φαίνεται στο παρακάτω παράδειγμα:

Πίνακας 6.2. Μέθοδοι της κλάσης `File`.

Μέθοδος	**Περιγραφή**
`boolean` **`canRead`** `()` `boolean` **`canWrite`** `()`	Ελέγχουν αν το αρχείο ή το directory μπορεί να διαβασθεί ή να γραφεί.
`boolean` **`delete`** `()`	Διαγράφει το αρχείο ή το directory.
`boolean` **`exists`** `()`	Ελέγχει αν το αρχείο ή το directory υπάρχει.

`String` **`getAbsolutePath`** `()`
`String` **`getName`** `()`
`String` **`getPath`** `()`
`String` **`getParent`** `()`
	Για κάποιο αρχείο ή directory, επιστρέφουν αντίστοιχα: το πλήρες όνομα, το όνομα, το μονοπάτι, και το όνομα του directory που το περιέχει.

`boolean` **`isAbsolute`** `()`	Ελέγχει αν ένα όνομα αρχείου ή directory είναι πλήρες.
`boolean` **`isDirectory`** `()` `boolean` **`isFile`** `()`	Ελέγχουν αν πρόκειται για directory ή για αρχείο αντίστοιχα.
`long` **`lastModified`** `()`	Επιστρέφει το χρόνο τελευταίας αλλαγής.
`long` **`length`** `()`	Επιστρέφει το μήκος του αρχείου σε bytes.
`String []` **`list`** `()`	Επιστρέφει μια λίστα όλων των αρχείων που περιέχονται σε κάποιο directory.
`boolean` **`mkdir`** `()`	Κατασκευάζει ένα directory.
`boolean` **`mkdirs`** `()`	Κατασκευάζει όλα τα directories στο μονοπάτι.
`boolean` **`renameTo`** `(File f)`	Μετονομάζει ένα αρχείο ή directory, χρησιμοποιώντας το f ως νέο όνομα.

```
FileInputStream s1 = new FileInputStream("/dir/file");

File            f  = new File("/tmp/foo.txt");
FileInputStream s2 = new FileInputStream(f);
```

Όταν δημιουργείται ένα `FileInputStream`, η Java επιχειρεί να ανοίξει το συγκεκριμένο αρχείο. Εάν το αρχείο αυτό δεν υπάρχει, οι κατασκευαστές της `FileInputStream` προκαλούν μια εξαίρεση τύπου `FileNotFoundException`. Σε περίπτωση που προκληθεί κάποιο άλλο λάθος Ε/Ε η εξαίρεση είναι τύπου `IOException`. Ας σημειωθεί, ότι κρίνεται απαραίτητο ο κώδικας της εφαρμογής να ανιχνεύει και να χειρίζεται τις εξαιρέσεις αυτές. Όταν ένα ρεύμα δημιουργείται για πρώτη φορά, η μέθοδος `available` και η μέθοδος `length` του αντικειμένου `File` επιστρέφουν την ίδια τιμή. Επίσης, επιβάλλεται η κλήση της μεθόδου `close` μετά το πέρας των εργασιών που αφορούν το αρχείο.

Με τον ίδιο τρόπο κατασκευάζονται ρεύματα τύπου `FileOutputStream`. Το παρακάτω τμήμα κώδικα κατασκευάζει ένα νέο αρχείο με όνομα `message.txt` και γράφει μέσα σε αυτό τη φράση “Hello world”. Για την εκτύπωση χρησιμοποιεί ένα φίλτρο τύπου `PrintStream`. Όλες οι εξαιρέσεις που μπορούν να προκύψουν ως ενδείξεις σφαλμάτων αγνοούνται.

```
File             f = new File("message.txt");
FileOutputStream s = new FileOutputStream(f);
PrintStream      p = new PrintStream(s);

try {
   p.println("Hello world.");
   p.close();
}
catch () { /* ignore! */ }
```

6.4. Παράδειγμα: Ανάγνωση αρχείων

Το παράδειγμα αυτό υλοποιεί έναν απλό αναγνώστη αρχείων, που λόγω της υλοποίησής του δεν ενδείκνυται για την ανάγνωση μεγάλων αρχείων. Η κλάση `FileReader` περιέχει τις εξής μεθόδους:

- **`FileReader`** `(String filename)`

Κατασκευαστής, που δέχεται ως παράμετρο το όνομα του αρχείου που πρόκειται να διαβασθεί. Ανοίγει το αρχείο, διαβάζει τα περιεχόμενά του και τα αποθηκεύει στο εσωτερικό του αντικειμένου `FileReader`.

- `String` **`getContents`** `()`

Επιστρέφει μια συμβολοσειρά με τα περιεχόμενα του αρχείου.

- `static void` **`main`** `(String [] args)`

Η ύπαρξη της `main` επιτρέπει τη χρήση της κλάσης `FileReader` ως ανεξάρτητης εφαρμογής. Ο πίνακας `args` πρέπει να περιέχει μόνο ένα στοιχείο, που είναι το όνομα του αρχείου που θα διαβασθεί. Σε αντίθετη περίπτωση, εκτυπώνεται κατάλληλο μήνυμα σφάλματος.

Προκειμένου να τρέξει το παράδειγμα, απαιτείται πρώτα να μεταγλωττισθεί η κλάση `FileReader`, που περιέχεται στο αρχείο `FileReader.java`. Αυτό μπορεί να γίνει με την εκτέλεση της εντολής:

```
% javac FileReader.java
```

όπου `%` το σήμα προτροπής (prompt) του λειτουργικού συστήματος. Στη συνέχεια, μπορεί να εκτελεσθεί το πρόγραμμα, χρησιμοποιώντας μια εντολή της μορφής:

```
% java FileReader testfile
```

Τα περιεχόμενα του αρχείου `testfile` εκτυπώνονται στην κοινή έξοδο.

Κλάση FileReader

```
import java.io.*;

public class FileReader
{
   private File    file;
   private byte [] data;

   // Κατασκευαστής για την κλάση FileReader

   public FileReader (String filename)
      throws IOException
   {
      file = new File(filename);

      FileInputStream in = new FileInputStream(file);

      data = new byte [in.available()];
      in.read(data);
   }

   // Πρόσβαση στα περιεχόμενα του αρχείου

   public String getContents ()
   {
      return new String(data, 0);
   }

   // Μέθοδος main, για να μπορεί η κλάση να
   // χρησιμοποιηθεί ως ανεξάρτητη εφαρμογή

   static public void main (String [] args)
   {
      if (args.length != 1) {
         System.err.println(
            "Usage: java FileViewer <filename>");
         System.exit(0);
      }

      try {
         FileReader f = new FileReader(args[0]);
         String contents = f.getContents();

         System.out.println(contents);
      }
      catch(IOException e) {
         System.err.println(e);
      }
   }
}
```

7. Χειρισμός γραφικών

Σε αυτό το κεφάλαιο περιγράφονται οι κλάσεις του API της Java που χρησιμοποιούνται για την ανάπτυξη φιλικών προς το χρήστη, γραφικών διαπροσωπειών ανθρώπου-μηχανής, είτε σε ανεξάρτητες εφαρμογές είτε σε applets. Οι κλάσεις αυτές περιέχονται στο πακέτο java.awt και υλοποιούν ό,τι έχει σχέση με τη δημιουργία παραθύρων, κουμπιών, μενού και άλλων στοιχείων των σύγχρονων παραθυρικών διαπροσωπειών, καθώς και λειτουργίες σχετιζόμενες με γραφικά, εικόνα και ήχο.

7.1. Εισαγωγή

Οι κλάσεις του API της Java που χρησιμοποιούνται για την ανάπτυξη γραφικών διαπροσωπειών ανθρώπου-μηχανής αποτελούν το AWT (Abstract Windowing Toolkit). Όπως υποδηλώνει το όνομα του, το AWT είναι ένα αφηρημένο (δηλαδή ανεξάρτητο υλοποίησης) σύνολο εργαλείων για την δημιουργία παραθυρικών εφαρμογών. Το AWT παρέχει στους προγραμματιστές της Java μια διαπροσωπεία αρκετά υψηλού επιπέδου, αντίστοιχη του MFC στα Microsoft Windows ή του Motif στα X-Windows.

Το AWT χρησιμοποιεί ένα σύνολο *συστατικών* (components), που του παρέχει η πλατφόρμα στην οποία τρέχει κάθε φορά η εφαρμογή. Μερικά χρήσιμα συστατικά, όπως θα δούμε στη συνέχεια, είναι τα κουμπιά, τα μενού, κ.λπ. Οι εφαρμογές και τα applets που βασίζονται στο AWT έχουν την ίδια συμπεριφορά και εμφάνιση με τις υπόλοιπες εφαρμογές της πλατφόρμας στην οποία τρέχουν. Δυστυχώς, όμως, για να μπορεί το AWT να δουλεύει σωστά ανεξάρτητα από τη συγκεκριμένη υλοποίηση, υποστηρίζει μόνο συστατικά που είναι διαθέσιμα σε όλες τις υποστηριζόμενες πλατφόρμες, υλοποιεί δηλαδή έναν ελάχιστο κοινό παρονομαστή λειτουργικότητας. Πάντως δεν υπάρχει καμμία εξάρτηση της γλώσσας Java από το AWT και μπορεί κανείς, αν θέλει, να υλοποιήσει και να χρησιμοποιήσει άλλα εργαλεία και βιβλιοθήκες γραφικών.

Οι κλάσεις του AWT παρέχουν στους προγραμματιστές τη δυνατότητα για τη δημιουργία παραθυρικών εφαρμογών βασισμένων στο χειρισμό γεγονότων (event-driven), την σχεδίαση γραφικών, την (απλή) επεξεργασία εικόνων και ήχου ενώ παράλληλα παρέχει ένα πλαίσιο για την ανάπτυξη applets, το οποίο θα περιγραφεί στη συνέχεια.

7.2. Applets

Όπως αναφέραμε ένας βασικός σχεδιαστικός στόχος της Java είναι η εύκολη ανάπτυξη applets, δηλαδή μικρών εφαρμογών ενσωματωμένων σε σελίδες HTML που τρέχουν μέσα σε

περιηγητές. Στη συνέχεια θα περιγραφεί το AWT με βάση τα applets, λόγω του ιδιαίτερα απλού τρόπου με τον οποίο είναι δυνατό να προγραμματισθούν. Τα ίδια ακριβώς πράγματα όμως ισχύουν και για τις ανεξάρτητες εφαρμογές.

Για να εισαχθεί ένα applet σε μια σελίδα HTML χρησιμοποιείται η εντολή `<APPLET>`. Ένα τυπικό παράδειγμα είναι το ακόλουθο:

```
<APPLET>CODE=MyApplet WIDTH=300 HEIGHT=400></APPLET>
```

Είναι επίσης δυνατό να καθορισθούν οι παράμετροι `ALT` (εναλλακτικό κείμενο για περιηγητές που δεν μπορούν να τρέξουν Java), `CODEBASE` (URL στο οποίο αρχίζει η αναζήτηση για τα αρχεία κλάσεων), `NAME` (ένα όνομα για το συγκεκριμένο στιγμιότυπο του applet, βάσει του οποίου μπορούν να επικοινωνήσουν applets που τρέχουν στην ίδια HTML σελίδα), `ALIGN` (η προτιμώμενη στοίχιση του applet), `VSPACE`, `HSPACE` (κάθετο και οριζόντιο περιθώριο ανάμεσα στο applet και τα γειτονικά στοιχεία HTML, μετρημένο σε pixels):

```
<APPLET CODE=MyApplet WIDTH=300 HEIGHT=400
   CODEBASE="http://www.softlab.ece.ntua.gr/"
   ALIGN=center VSPACE=10 HSPACE=10>
</APPLET>
```

Επίσης μεταξύ των ετικετών `<APPLET>` και `</APPLET>` μπορούν να καθορισθούν οι παραμέτρους του applet με τη μορφή συμβολοσειρών. Αυτές μπορούν να διαβασθούν από το applet με τη `getParameter`. Για παράδειγμα, αν το αρχείο HTML περιέχει την εντολή:

```
<APPLET CODE=MyApplet WIDTH=300 HEIGHT=400>
<PARAM NAME=MyParameter VALUE=500>
</APPLET>
```

τότε η τιμή της παραμέτρου `MyParameter` μπορεί να διαβασθεί στον κώδικα του applet με την εντολή Java:

```
String myparam = getParameter("MyParameter");
```

Όπως στα περισσότερα περιβάλλοντα γραφικών διαπροσωπειών, στη Java ακολουθείται το μοντέλο του *προγραμματισμού βασισμένου σε γεγονότα* (event-driven programming). Σύμφωνα με το αυτό, ο προγραμματιστής δεν δηλώνει τη σειρά με την οποία εκτελούνται οι διάφορες λειτουργίες της εφαρμογής, αλλά απλά γράφει μεθόδους με τις οποίες χειρίζεται τα γεγονότα. Τα γεγονότα προέρχονται είτε από τον χρήστη (π.χ. είσοδος από το πληκτρολόγιο, κίνηση του ποντικιού, κ.λπ.) είτε από το περιβάλλον (π.χ. αίτηση επανασχεδιασμού ενός μέρους της οθόνης). Στην περίπτωση των applets ο περιηγητής επεξεργάζεται τα γεγονότα που έρχονται και καλεί τις αντίστοιχες μεθόδους του κατάλληλου applet.

Όλα τα αντικείμενα applets προέρχονται από την κλάση `Applet` ή υποκλάσεις αυτής. Η `Applet` περιέχει όλες τις μεθόδους που καλούνται όταν συμβεί ένα γεγονός. Ο κώδικας αυτών των μεθόδων υλοποιεί την πιο απλή συμπεριφορά για το χειρισμό των γεγονότων, που συνήθως είναι τετριμμένη. Ένα αντικείμενο που προέρχεται από την κλάση `Applet` είναι ένα νόμιμο αλλά ελάχιστα ενδιαφέρον applet, που δε δείχνει τίποτα και δεν κάνει τίποτα. Σε υποκλάσεις της `Applet` μπορούν να ορισθούν μέθοδοι που υπερισχύουν των υπαρχόντων και υλοποιούν την επιθυμητή συμπεριφορά στα διάφορα γεγονότα.

Οι βασικές μέθοδοι της κλάσης `Applet` περιγράφονται στον Πίνακα 7.1. Εκτός από αυτές, η κλάση `Applet` κληρονομεί τις μεθόδους των κλάσεων `Component`, `Container` και `Panel`, τις οποίες χρησιμοποιεί για να επικοινωνεί με το γραφικό περιβάλλον και να δέχεται τα γεγονότα που την αφορούν. Οι περισσότερες από τις μεθόδους της `Applet` πρέπει να επανορισθούν σε υποκλάσεις, προκειμένου να κάνουν κάτι χρήσιμο.

***Πίνακας 7.1. Μέθοδοι της κλάσης* `Applet`.**

Μέθοδος	**Περιγραφή**
`init()`	Καλείται μόνο μια φορά όταν ξεκινάει η εκτέλεση του applet και χρησιμοποιείται κυρίως για αρχικοποιήσεις.
`start()`	Καλείται κάθε φορά που το applet γίνεται ορατό από το χρήστη.
`stop()`	Καλείται κάθε φορά που το applet παύει να είναι ορατό από το χρήστη και χρησιμεύει κυρίως για την απελευθέρωση πόρων του συστήματος.
`destroy()`	Καλείται μόνο μια φορά, όταν η εκτέλεση του applet έχει τελειώσει οριστικά.

Το παρακάτω πρόγραμμα Java υλοποιεί ένα applet που δείχνει απλώς το κλασσικό μήνυμα “Hello world!”:

```
import java.awt.*;
import java.applet.Applet;

class HelloWorld extends Applet
{
   public void paint (Graphics g)
   {
      g.drawString("Hello world!", 50, 50);
   }
}
```

Για να χρησιμοποιηθεί το AWT χρειάζεται η πρώτη εντολή `import`, ενώ για να ορισθεί ένα applet χρειάζεται η δεύτερη εντολή `import`. Το applet που ορίζεται στο παραπάνω παράδειγμα επανορίζει απλώς τη μέθοδο `paint`, που καλείται κάθε φορά που χρειάζεται να σχεδιασθούν τα περιεχόμενα του applet. Η μέθοδος αυτή απλώς δείχνει τη συμβολοσειρά “Hello world!” σε ένα σημείο κοντά στο κέντρο του applet, με συντεταγμένες (50, 50) (σε pixels).

7.3. Συστατικά

Όλες σχεδόν οι παραθυρικές εφαρμογές, ακόμα και αυτές που υλοποιούν σύνθετες και περίπλοκες γραφικές διαπροσωπείες, συντίθενται από ένα σχετικά μικρό αριθμό απλών *συστατικών* (components), κατάλληλα τοποθετημένων μέσα σε *αποθήκες* (containers). Κάθε συστατικό επιτελεί μια συγκεκριμένη λειτουργία. Το AWT περιέχει μια πληθώρα από συστατικά που μπορούν να χρησιμοποιηθούν για την κατασκευή γραφικών διαπροσωπειών. Η λειτουργικότητά τους ενισχύεται σε κάθε καινούρια έκδοση του API, διευκολύνοντας την ανάπτυξη όλο και πιο εξελιγμένων παραθυρικών εφαρμογών.

7.3.1. Συστατικά, αποθήκες και γεγονότα

Στο AWT, όλα τα συστατικά είναι αντικείμενα που προέρχονται από υποκλάσεις της `Component`. Μπορούμε να διακρίνουμε τα συστατικά σε δύο κατηγορίες: τα απλά συστατικά, όπως π.χ. τα κουμπιά και τα πεδία κειμένου, και τις αποθήκες που, χωρίς να παρέχουν από μόνες τους κάποια λειτουργικότητα, χρησιμοποιούνται για την ομαδοποίηση άλλων συστατικών, ακόμα και άλλων αποθηκών.

Οι βασικές αποθήκες που θα χρησιμοποιηθούν στη συνέχεια είναι αντικείμενα των κλάσεων `Frame` (αυτόνομα παράθυρα) ή `Panel` (αποθήκες που περιέχονται μέσα σε άλλα αντικείμενα `Frame` ή `Panel`). Καθώς τα applets τρέχουν μέσα στο παράθυρο ενός περιηγητή, η κλάση `Applet` κληρονομεί την `Panel`, ενώ συνήθως οι αυτόνομες εφαρμογές που χρησιμοποιούν το AWT δηλώνουν την βασική κλάση τους ως απόγονο της `Frame`. Επίσης τα πλαίσια διαλόγου είναι στιγμιότυπα της `Dialog`, απογόνου της `Frame`.

Για να δημιουργήθεί ένα συστατικό πρέπει να κληθεί ο κατασκευαστής του με τα κατάλληλα ορίσματα. Για παράδειγμα, ο κατασκευαστής της κλάσης `Button`, που υλοποιεί ένα κουμπί, δέχεται ώς όρισμα την επιγραφή του κουμπιού, π.χ.

```
Button b = new Button("My first button!");
```

Προκειμένου να εμφανισθεί ένα χαρακτηριστικό, πρέπει πρώτα να προστεθεί σε μια αποθήκη, χρησιμοποιώντας τη μέθοδο `add` της τελευταίας. Για παράδειγμα, το τμήμα κώδικα:

```
ctr.add(b);
```

προσθέτει το κουμπί `b` στην αποθήκη `ctr`. Επιτρέπεται η αποθήκευση αποθηκών μέσα σε άλλες αποθήκες, υπό την προϋπόθεση φυσικά η πιο εξωτερική αποθήκη να προέρχεται από την κλάση `Frame`, ή υποκλάσεις αυτής, και οι εσωτερικές να είναι αντικείμενα που προέρχονται από την κλάση `Panel`, ή υποκλάσεις αυτής.

Καθώς δημιουργεί κανείς τη γραφική διαπροσωπεία μιας εφαρμογής, συνήθως υλοποιεί μια δενδρική δομή απόαποθήκες, ξεκινώντας από κάποιο αρχικό παράθυρο. Οι κόμβοι αυτής της δομής είναι αποθήκες και τα φύλλα είναι απλά συστατικά.

Κάθε φορά που ο χρήστης αλληλεπιδρά με ένα συστατικό με οποιοδήποτε τρόπο, δημιουργείται ένα γεγονός. Η τυπική μέθοδος που καλείται για τα περισσότερα βασικά συστατικά σε αυτή την περίπτωση είναι η `action`. Ένας απλός τρόπος για να ανατεθεί μια λειτουργία σε ένα κουμπί, για παράδειγμα, είναι να ορισθεί μια υποκλάση της `Button` που υλόποιεί εκ νέου τη μέθοδο `action`. Το παρακάτω τμήμα κώδικα υλοποιεί ένα κουμπί που αναγράφει την τιμή ενός μετρητή. Κάθε φορά που πιέζεται, η τιμή του μετρητή αυξάνει κατά ένα:

```
import java.awt.*;
import java.applet.Applet;

public class MyApplet extends Applet
{
   public void init ()
   {
      add(new MyButton(1));
   }
}

class MyButton extends Button
```

```
{
   private int counter;

   public MyButton (int c)
   {
      counter = c;
      setLabel(counter);
   }

   public boolean action (Event e, Object o)
   {
      setLabel(++counter);
      repaint();
      return true;
   }
}
```

Η μέθοδος `action` είναι μια μόνο από τις μεθόδους χειρισμού γεγονότων που μπορεί να ορίσει κανείς σε ένα συστατικό. υπάρχουν αρκετές άλλες για γεγονότα σχετικά με το ποντίκι (`mouseUp`, `mouseDown`, `mouseDrag`), το πληκτρολόγιο (`keyUp`, `keyDown`), την εστίαση κ.α., καθώς και η γενική μέθοδος `handleEvent` η οποία μπορεί να χειριστεί οποιοδήποτε γεγονός.

Τα πιο σημαντικά συστατικά που περιέχονται στο AWT περιγράφονται συνοπτικά στον Πίνακα . Η ιεραρχία τους απεικονίζεται στο Σχήμα .

Πίνακας 7.2. Κλάσεις συστατικών του AWT.

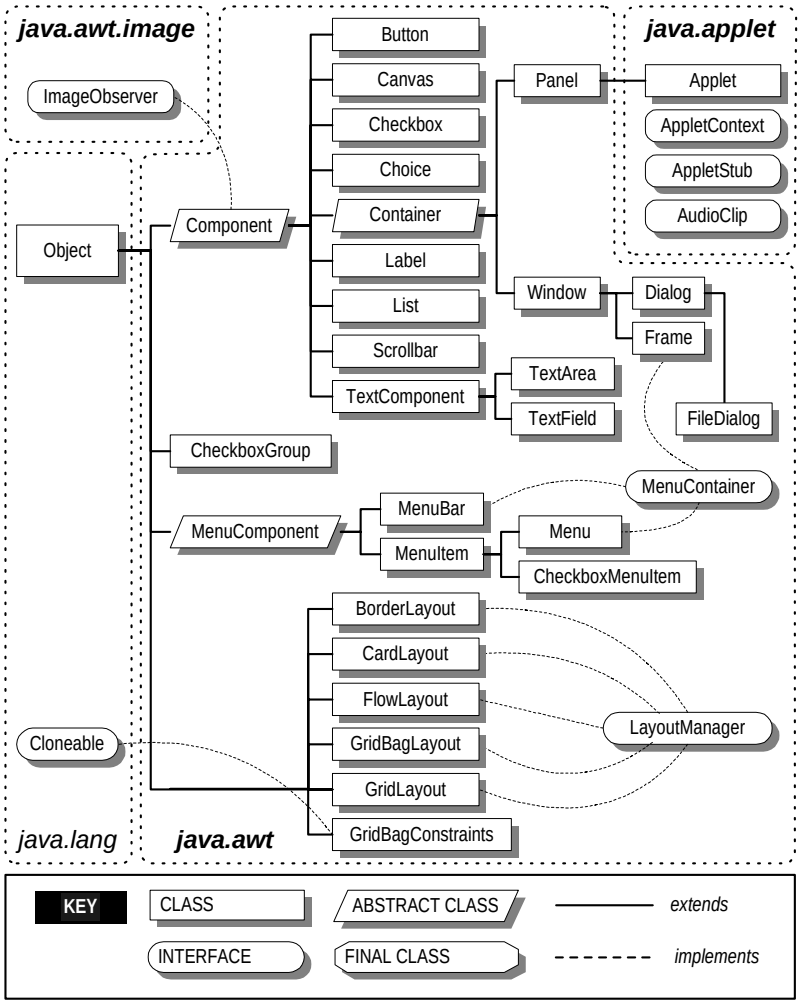
Κλάση	***Περιγραφή***
`Button`	Απλό κουμπί.
`Canvas`	Αφηρημένη κλάση που υλοποιεί τη βάση συστατικών σχετικών με γραφικά και εικόνες.
`Checkbox`	Κουμπί τύπου check box ή radio button, ανάλογα με τον κατασκευαστή που καλείται. Όταν πρόκειται για radio button πρέπει να ορίσθεί και το CheckboxGroup στο οποίο ανήκει.
`Choice`	Λίστα τύπου drop-down.
`FileDialog`	Πλαίσιο διαλόγου για την επιλογή αρχείων.
`Label`	Στατικό πεδίο κειμένου, που μπορεί να περιέχει μια μόνο γραμμή.
`List`	Λίστα συμβολοσειρών, απλής ή πολλαπλής επιλογής.
`Menu`	Αντιπροσωπεύει ένα μενού μέσα σε ένα MenuBar ή ένα υπομενού μέσα σε ένα άλλο μενού.
`MenuBar`	Αντιπροσωπεύει μια συλλογή από μενού. Χρησιμοποιείται μόνο σε αντικείμενα τύπου Frame.
`MenuItem`	Αντιπροσωπεύει μια από τις δυνατές επιλογές ενός μενού ή ενός MenuBar.
`Scrollbar`	Μπάρα που συνήθως χρησιμοποιείται για την κύλιση της ορατής επιφάνειας ενός παραθύρου.
`TextArea`	Στοιχείο για εισαγωγή κειμένου. Μπορεί να περιέχει πολλές γραμμές και διαθέτει αυτόματη κύλιση (scrolling) και τις βασικές εντολές ενός επεξεργαστή κειμένου.
`TextField`	Όπως το TextArea για μια μόνο γραμμή κειμένου.
`Window`	Αυτόνομο παράθυρο όπως το Frame. Ωστόσο, το Frame έχει πολύ

	περισσότερες δυνατότητες.

7.3.2. Χρήση των συστατικών

Απλό παράδειγμα διαλόγου

Ο καλύτερος τρόπος για να παρουσιασθούν τα συστατικά του AWT είναι μέσω μιας σειράς απλών παραδειγμάτων. Στο πρώτο παράδειγμα κατασκευάζεται ένα απλό πλαίσιο διαλόγου επιβεβαίωσης, το οποίο θα έχει μια περιοχή μηνυμάτων και δύο κουμπιά με ετικέττες `YES` και `NO`. Στο παράδειγμα αυτό έμφαση δίνεται στην κατασκευή του διαλόγου και όχι τον μηχανισμό με τον οποίο ανταποκρίνεται στις ενέργειες του χρήστη.



Σχήμα 7.1. Ιεραρχία των κλάσεων συστατικών του AWT.

Το πλαίσιο διαλόγου είναι ένα αντικείμενο της κλάσης `Dialog`. Στον κατασκευαστή του θα πρέπει να κατασκευασθούν και να εμφανισθούν τα διάφορα συστατικά που αποτελούν το διάλογο. Μια προσέγγιση του ορισμού της κλάσης `ConfirmDialog` ακολουθεί, από την οποία παραλείπονται ακόμα αρκετά σημαντικά σημεία:

```
class ConfirmDialog extends Dialog
{
   public ConfirmDialog (String message)
   {
      // Ο κατασκευαστής του Dialog παίρνει μεταξύ
      // άλλων ως παράμετρο τον τίτλο του παραθύρου.

      super(null, "Confirmation", true);

      // Κατασκευή του user interface.
   }

   // Μέθοδοι χειρισμού γεγονότων.
}
```

Ο κατασκευαστής του διαλόγου παίρνει ως όρισμα τη συμβολοσειρά με το μήνυμα που θα εμφανισθεί στο χρήστη. Καλείται ρητά ο κατασκευαστής της κλάσης βάσης, η επικεφαλίδα του οποίου είναι:

```
Dialog (Frame owner, String title, boolean isModal)
```

Στην πρώτη παράμετρο ορίζεται ποιο παράθυρο είναι ο “ιδιοκτήτης” του διαλόγου. Αν δεν υπάρχει ιδιοκτήτης, όπως στην προκειμένη περίπτωση, χρησιμοποιείται η τιμή `null`. Στη δεύτερη παράμετρο ορίζεται ο τίτλος του παραθύρου διαλόγου, ενώ στην τρίτη μπορεί να ορισθεί ο διάλογος ως modal. Στην περίπτωση αυτή, η εμφάνιση του διαλόγου μπλοκάρει την είσοδο του χρήστη προς τον ιδιοκτήτη, όσο ο διάλογος είναι ανοικτός.

Στον κατασκευαστή του διαλόγου, πρέπει επίσης να καθορισθεί το αρχικό μέγεθος. Επίσης, πρέπει να κληθεί η μέθοδος `show` ώστε ο διάλογος να εμφανιστεί στην οθόνη (αρχικά είναι αόρατος):

```
resize(300, 200);
show();
```

Στη συνέχεια, το πλαίσιο διαλόγου είναι έτοιμο να δεχτεί τα διάφορα συστατικά του. Αυτά θα είναι ένα `TextArea`, για την εμφάνιση του μηνύματος, καθώς και δύο αντικείμενα της κλάσης `Button`, που θα υλοποιούν τα κουμπιά. Η αρχικοποίηση του αντικειμένου `TextArea` γίνεται με το ακόλουθο τμήμα κώδικα:

```
TextArea msgArea = new TextArea(message);
msgArea.setEditable(false);
```

Η πρώτη γραμμή κατασκευάζει το `TextArea`, θέτοντας ως αρχικό κείμενο το μήνυμα που δίνεται στο `message`. Προκειμένου να μην επιτρέπεται η αλλαγή αυτού του μηνύματος, καλείται η μέθοδος `setEditable`, με παράμετρο `false`.

Στη συνέχεια δημιουργούνται τα δύο κουμπιά και δηλώνονται οι ετικέττες τους ως εξής:

```
Button btYES = new Button("Yes");
Button btNO  = new Button("No");
```

Ως τώρα έχουν δημιουργηθεί όλα τα συστατικά του διαλόγου αλλά δεν έχουν τοποθετηθεί μέσα στο πλαίσιο. Αυτό γίνεται με τις παρακάτω εντολές:

```
add(msgArea);
add(btYES);
add(btNO);
```

Επομένως, η νέα προσέγγιση του συνολικού κώδικα της κλάσης `ConfirmDialog`, που αυτή τη φορά εμφανίζει τα συστατικά στην οθόνη, είναι η ακόλουθη:

```
class ConfirmDialog extends Dialog
{
   public ConfirmDialog (String message)
   {
      super(null, "Confirmation", true);
      resize(300, 200);

      TextArea msgArea = new TextArea(message);
      Button   btYES   = new Button("Yes");
      Button   btNO    = new Button("No");

      msgArea.setEditable(false);
      add(msgArea);
      add(btYES);
      add(btNo);
      show();
   }
}
```

Μια τυπική χρήση του παραπάνω παράθυρου διαλόγου θα ήταν για παράδειγμα η ακόλουθη:

```
ConfirmDialog cd =
   new ConfirmDialog("Do you want to proceed?");
```

Για να κλεισει ο διάλογος, αφού ο χρήστης δώσει τις απαραίτητες πληροφορίες, πρέπει να εκτελεσθεί η εντολή:

```
cd.dispose();
```

Χρήση Panel

Στη συνέχεια θα ορισθεί ένα `Panel` το οποίο θα χρησιμοποιείται για τη σύνθεση χρωμάτων στο σύστημα RGB. Θα περιέχει τρεις μπάρες κύλισης, μια για κάθε βασικό χρώμα (κόκκινο, πράσινο και μπλε) με τις οποίες ο χρήστης θα μπορεί να αλλάζει τις τιμές των τριων χρωμάτων αντίστοιχα. Επίσης, θα έχει μια περιοχή που θα δείχνει το τελικό χρώμα.

Η νέα κλάση που κατασκευάζεται ονομάζεται `ColorComposePanel` και είναι υποκλάση του `Panel`. Αφού το `Panel` περιέχεται μέσα σε ένα παράθυρο, δεν χρειάζεται να κάνει `resize` ή `show`: αυτά γίνονται αυτόματα. Μια πρώτη προσέγγιση της κλάσης είναι η εξής:

```
class ColorComposePanel extends Panel
{
   // Δηλώσεις των συστατικών

   public ColorComposePanel () {
      // Κατασκευή και εμφάνιση των συστατικών.
   }
}
```

Τα συστατικά που θα δηλωθούν είναι οι τρεις μπάρες κύλισης, με ονόματα `rscroll`, `gscroll` και `bscroll`, καθώς και ένα αντικείμενο τύπου `Canvas`, με όνομα `result`. Το αντικείμενο αυτό χρησιμοποιείται για να εμφανίζει την τελική σύνθεση. Έτσι οι δηλώσεις των συστατικών θα είναι οι εξής:

```
protected Scrollbar rscroll, gscroll, bscroll;
protected Canvas    result;
```

Η κατασκευή και εμφάνιση των συστατικών αυτών στο εσωτερικό του κατασκευαστή της κλάσης `ColorComposePanel` γίνεται με τις ακόλουθες εντολές:

```
rscroll = new Scrollbar(Scrollbar.HORIZONTAL);
gscroll = new Scrollbar(Scrollbar.HORIZONTAL);
bscroll = new Scrollbar(Scrollbar.HORIZONTAL);
result  = new Canvas();
```

Στον κατασκευαστή κάθε μπάρας κύλισης γίνεται η επιλογή της κατεύθυνσής της. Στην περίπτωσή μας οι μπάρες θα είναι οριζόντιες, κάτι που καθορίζεται με το όρισμα `Scrollbar.HORIZONTAL`.

Τέλος η εμφάνιση των συστατικών γίνεται έμμεσα με την πρόσθεσή τους στο `Panel`. Αυτό γίνεται με τις εντολές:

```
add(rscroll);
add(gscroll);
add(bscroll);
add(result);
```

Η επόμενη προσέγγιση του συνολικού κώδικα είναι η ακόλουθη:

```
class colorComposePanel extends Panel
{
   protected Scrollbar rscroll, gscroll, bscroll;
   protected Canvas result;

   public colorComposePanel ()
   {
      rscroll = new Scrollbar(Scrollbar.HORIZONTAL);
      add(rscroll);
      gscroll = new Scrollbar(Scrollbar.HORIZONTAL);
      add(gscroll);
      bscroll = new Scrollbar(Scrollbar.HORIZONTAL);
      add(bscroll);
      result = new Canvas();
      add(final);
   }

   // Κάθε φορά που θα μετακινείται κάποια μπάρα
   // θα πρέπει να υπολογίζεται το νέο χρώμα.
   // Η εμφάνισή της γίνεται με μια κλήση της
   // μεθόδου setBackground του αντικειμένου result.
}
```

7.3.3. Χαρακτηριστικά των συστατικών

Κοινά χαρακτηριστικά για όλα τα συστατικά του AWT είναι οι μέθοδοι που κληρονομούνται από την κλάση `Component`. Οι πιο ενδιαφέρουσες από αυτές συνοψίζονται στον Πίνακα 7.3.

Πίνακας 7.3. Μέθοδοι της κλάσης **`Component`.**

Μέθοδος	***Περιγραφή***
`paint()`	Καλείται κάθε φορά που χρειάζεται να σχεδιαστεί όλη η επιφάνεια του συστατικού.
`repaint()`	Δηλώνεται ότι η επιφάνεια ενός στοιχείου πρέπει να επανασχεδιαστεί. Η προκαθορισμένη έκδοση αυτής της μεθόδου καλεί απλά την paint, ενώ είναι δυνατό να ζητηθεί η επανασχεδίαση μέρους μόνο της επιφάνειας.

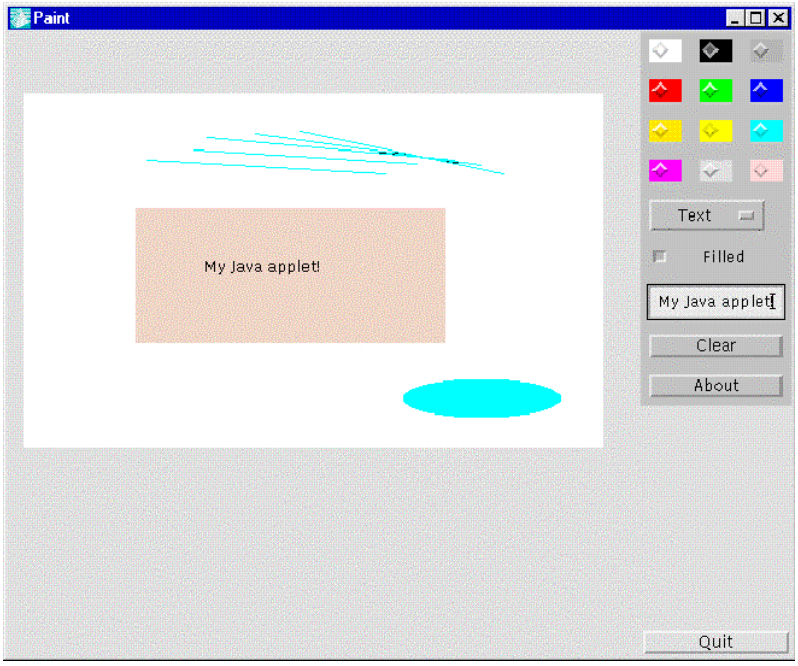
`getBackground()`
`setBackground(c)`
`getForeground()`
`setForeground(c)`
	Οι μέθοδοι αυτές επιτρέπουν το διάβασμα και την αλλαγή των χρωμάτων του ενός συστατικού. Λεπτομέρειες όσον αφορά στα χρώματα θα συζητηθούν σε επόμενη παράγραφο.

`getFont()` `setFont(f)`	Οι μέθοδοι αυτές επιτρέπουν το διάβασμα και την αλλαγή της γραμματοσειράς με την οποία εμφανίζεται το κείμενο σε ένα συστατικό. Λεπτομέρειες όσον αφορά στις γραμματοσειρές συζητούνται αργότερα.
`size()`	Επιστρέφει τις διαστάσεις ενός συστατικού.
`resize(w, h)`	Αλλάζει τις διαστάσεις (σε pixels) ενός συστατικού.
`bounds()`	Επιστρέφει τις διαστάσεις και τη θέση ενός συστατικού.
`reshape(x, y, w, h)`	Αλλάζει τη θέση και τις διαστάσεις ενός συστατικού.
`enable()` `disable()`	Οι μέθοδοι αυτές ενεργοποιούν και απενεργοποιούν ένα συστατικό. Τα ανενεργα συστατικά δεχονται είσοδο από το χρήστη.

Στη συνέχεια του κεφαλαίου, καθώς θα παρουσιάζονται τα διάφορα χαρακτηριστικά του AWT, θα αναπτυχθεί η εφαρμογή `Paint`, ένα μικρό πρόγραμμα ζωγραφικής. Για την ανάπτυξή του θα χρησιμοποιηθούν αρκετά από τα χαρακτηριστικά του AWT. Η τελική εικόνα της εφαρμογής φαίνεται στο Σχήμα 7.2.

7.4. Διαχειριστές διάταξης

Οι *διαχειριστές διάταξης* (layout managers) είναι αντικείμενα που ελέγχουν την διάταξη και το μέγεθος των στοιχείων μέσα σε μια αποθήκη. Τα αντικείμενα αυτά τοποθετούν και μεταβάλλουν το μέγεθος των διαφόρων συστατικών που περιέχονται στην αποθήκη, σύμφωνα με κάποιο *σχήμα διάταξης* (layout scheme). Το AWT διαθέτει αρκετές κλάσεις διαχειριστών διάταξης, που υλοποιούν κοινά σχήματα. Φυσικά, μπορεί κανείς να υλοποιήσει το δικό του διαχειριστή για εξειδικευμένες εφαρμογές.



Σχήμα 7.2. Η εφαρμογή **`Paint`.**

Κάθε αποθήκη έχει ένα διαχειριστή μορφής που καθορίζεται αυτόματα από το AWT. Μπορεί κανείς εύκολα να τον αλλάξει καλώντας την μέθοδο `setLayout`. Για παράδειγμα, για να χρησιμοποιήσει κανείς το `BorderLayout` στην αποθήκη `c`, αρκεί να εκτελέσει την εντολή:

```
c.setLayout(new BorderLayout());
```

Συνήθως δεν χρειάζεται να κληθούν ποτέ μέθοδοι του διαχειριστή διάταξης απευθείας, εκτός από τις περιπτώσεις των `CardLayout` και `GridBagLayout`. Η αποθήκη συμβουλεύεται τον διαχειριστή όταν προστίθενται, αφαιρούνται ή τροποποιούνται χαρακτηριστικά των συστατικών που περιέχει.

Για αυτό τον σκοπό υπάρχουν υπερφορτωμένες εκδόσεις της μεθόδου add μιας αποθήκης. Η `add` με μόνο όρισμα το στοιχείο που πρόκειται να προστεθεί αφήνει στο διαχειριστή διάταξης να επιλέξει τη θέση και τις διαστάσεις του αντικειμένου. Η `add` με επιπλέον ορίσματα χρησιμοποιείται για να δηλώσει κανείς στο διαχειριστή που και πώς θέλει να τοποθετηθεί το στοιχείο. Για παράδειγμα στο `BorderLayout`, μπορούμε να προσθέσουμε ένα στοιχείο στη "βόρεια" (πάνω) πλευρά της αποθήκης, με την εντολή:

```
c.add("North", myComponent);
```

Ένας διαχειριστής διάταξης διατάσσει τα αντικείμενα της αποθήκης που διαχειρίζεται μόνο όταν αυτό του ζητηθεί. Αυτό μπορεί να χρειαστεί όταν αλλάζει το μέγεθος της αποθήκης, όταν αλλάζει η μετακινείται κάποιο συστατικό, ή τέλος όταν προστίθενται ή αφαιρούνται συστατικά. Όταν συμβεί κάτι τέτοιο, μπορεί κανείς να δώσει εντολή στο διαχειριστή να επαναδιατάξει τα περιεχόμενα της αποθήκης καλώντας την μέθοδο `validate` της κλάσης `Container`. Αυτή με τη σειρά της καλεί την μέθοδο `layout` για να ενεργοποιήσει το διαχειριστή. Η διαδικασία είναι αναδρομική, με την έννοια ότι όταν καλείται η `validate` σε μια αποθήκη, καλείται αναδρομικά και για όλα τα συστατικά που περιέχονται σε αυτήν.

Το πιο κοινό σχήμα διάταξης σε ένα `Panel` υλοποιείται από το διαχειριστή `FlowLayout`, που προσπαθεί να τοποθετήσει τα συστατικά με το μέγεθος που αυτά προτιμούν, γεμίζοντας το παράθυρο από τα αριστερά προς τα δεξιά και από τα πάνω προς τα κάτω. Σε ένα `FlowLayout`, μπορούμε να καθορίσουμε τη στοίχιση και ένα σταθερό περιθώριο.

Στο `GridLayout`, τα στοιχεία διατάσσονται σε ένα ορθογωνικό πλέγμα, με στήλες και γραμμές, και το μέγεθός τους προσαρμόζεται στο μέγεθος του κελιού. Ο αριθμός των στηλών και των γραμμών ορίζεται στον κατασκευαστή, π.χ.

```
GridLayout(15, 10);
```

ορίζει ένα πλέγμα με 5 γραμμές και 10 στήλες. Το να δώσουμε 0 για μία διάσταση σημαίνει ότι δεν μας ενδιαφέρει πόσα αντικείμενα θα "στοιβαχθούν" σε αυτήν τη διάσταση, π.χ. αν ορίσουμε `GridLayout(5, 0)` και προσθέσουμε 10 αντικείμενα στην αποθήκη, τότε αυτά θα διαταχθούν σε 5 σειρές με 2 αντικείμενα στην κάθε μια. Το `GridLayout` προφανώς προσφέρεται όταν θέλουμε να ομαδοποιήσουμε σε μια αποθήκη πολλά όμοια αντικείμενα. Υπάρχουν επίσης κατασκευαστές με επιπλέον παραμέτρους.

Ένας άλλος κοινός διαχειριστής διάταξης είναι το `BorderLayout`, στο οποίο μπορεί κανείς να ορίσει όταν προσθέτει ένα αντικείμενο ότι θέλει να βρίσκεται σε κάποια συγκεκριμένη θέση, σχετικά με τα όρια του παραθύρου. Χρήσιμες θέσεις είναι τα τέσσερα σημεία του ορίζοντα, όπου "North" αντιστοιχεί στην πάνω πλευρά του παραθύρου, ή "Center", που αντιστοιχεί στο κέντρο. Το `BorderLayout` είναι ο αρχικός διαχειριστής σε ένα `Frame` ή `Window`.

Ας δούμε πως μπορούν να χρησιμοποιήθούν οι διαχειριστές διάταξης για να τοποθετηθούν τα διάφορα `Panel` μέσα στο `Paint` applet. Το κύριο παράθυρο που δημιουργεί η κλάση `Paint` έχει `BorderLayout`. Τα διάφορα κουμπιά είναι προσανατολισμένα "East", και τον υπόλοιπο χώρο καταλαμβάνει το `Canvas`.

```
Panel p = new Panel();

p.setLayout(new BorderLayout(5, 5));
if (isApplication)
   p.add("South", new Button("Quit"));
p.add("North", new ControlPanel());

setLayout(new BorderLayout(5, 5));
add("East", p);
add("Center", new PaintCanvas(cpanel));
```

Το AWT διαθέτει και άλλους διαχειριστές διάταξης, όπως το `CardLayout` που προσομοιώνει μία στοίβα από "τραπουλόχαρτα" και δείχνει ένα κάθε φορά, ή το `GridBagLayout`, στον οποίο μπορεί να θέσει κανείς περιορισμούς και κανόνες για την τοποθέτηση και το μέγεθος των συστατικών. Για περισσότερες λεπτομέρειες πάνω στο API, που συνεχώς εμπλουτίζεται, ο αναγνώστης παραπέμπεται στην τεκμηρίωση του AWT.

7.5. Χειρισμός γεγονότων

Τα *γεγονότα* (events) δημιουργούνται από το γραφικό περιβάλλον και στέλνονται στο αντίστοιχο συστατικό καλώντας τις αντίστοιχες μεθόδους του. Όταν μια μέθοδος χειρισμού γεγονότων καλείται, παίρνει πάντα ως παράμετρο ένα αντικείμενο τύπου `Event`, και ίσως κάποιες επιπλέον

παραμέτρους. Το αντικείμενο αυτό περιέχει πεδία που προσδιορίζουν τον τύπο του γεγονότος και από που προήλθε.

Μια εφαρμογή, ή ένα applet, μπορεί να χειριστεί ένα γεγονός σε δύο επίπεδα. Η μέθοδος `handleEvent` χειρίζεται γεγονότα σε χαμηλό επίπεδο. Καλείται για όλα τα είδη των γεγονότων που φθάνουν σε ένα συστατικό. Η προκαθορισμένη συμπεριφορά της `handleEvent` είναι να εξετάζει τον τύπο του γεγονότος και να το περνά σε μια από τις πολλές βοηθητικές μεθόδους που έχουν οριστεί για να διευκολύνουν τον χειρισμό συγκεκριμένων γεγονότων. Οι απόγονοι της κλάσης `Component` μπορούν να εξειδικεύουν τις μεθόδους για το χειρισμό γεγονότων σχετικών με το ποντίκι (`mouseDown`, `mouseUp`), το πληκτρολόγιο (`keyDown`, `keyUp`), την εστίαση (`focus`), καθώς και διαφόρων γεγονότων που αντιστοιχούν σε κουμπιά και επιλογές μενού.

Σε μια εφαρμογή όπως είδαμε τα βασικά συστατικά αποτελούν φύλλα σε μια δενδρική δομή, που έχει ως ρίζα το αρχικό παράθυρο και ενδιάμεσους κόμβους τα `Panel` που περιέχει. Αυτή την δομή διασχίζει από κάτω προς τα πάνω κάθε γεγονός, ξεκινώντας από το βασικό συστατικό στο οποίο δημιουργήθηκε και συνεχίζοντας στις αποθήκες που το περιέχουν. Η διαδικασία αυτή τερματίζεται όταν ολοκληρωθεί ο χειρισμός του.

Όταν ένα συστατικό χρησιμοποιεί μια από τις εξειδικευμένες μεθόδους χειρισμού γεγονότων, π.χ. τη `mouseDown`, εξετάζει συνήθως το αντικείμενο `Event` που δίνεται ως παράμετρος και αποφασίζει αν μπορεί και αν ενδιαφέρεται να το χειριστεί. Στη θετική περίπτωση, δηλώνει στο περιβάλλον ότι χειρίστηκε το γεγονός επιστρέφοντας την `boolean` τιμή `true`. Αν επιστραφεί η τιμή `false`, το γεγονός περνά από το συστατικό στην αποθήκη που το περιέχει, κ.ο.κ. μέχρι να “καταναλωθεί” από κάποια μέθοδο χειρισμού γεγονότος. Για ένα συστατικό που ενδιαφέρεται να χειριστεί πολλά είδη γεγονότων, ο πιο απλός τρόπος είναι να το κάνει στη μέθοδο `handleEvent`, εξετάζοντας τον τύπο του `Event` και τα δεδομένα που έρχονται μέσα στο αντικείμενο `Event`. Ένα συστατικό μπορεί επίσης να δημιουργήσει και να στείλει το ίδιο γεγονότα, καλώντας τη μέθοδο `postEvent`. Αυτός είναι ο τρόπος με τον οποίο υλοποιούνται συστατικά που παράγουν δικούς τους τύπους γεγονότων.

Το πού ακριβώς σε μια εφαρμογή είναι προτιμότερο να χειριζόμαστε τα γεγονότα εξαρτάται από πολλούς παράγοντες, όπως π.χ. ο αριθμός και το είδος των συστατικών που έχουμε, η ιεραρχία των συστατικών, κ.α.

7.6. Γραφικά και ήχοι

Το AWT περιέχει κλάσεις για την δημιουργία και απεικόνιση, απλών γραφικών αντικειμένων σε δυο διαστάσεις. Στο εμπόριο διατίθενται διαπροσωπείες Java για βιβλιοθήκες και συστήματα γραφικών σε τρεις διαστάσεις, όπως VRML και OpenGL. Προς το παρόν, όμως, λόγω της εξάρτησης αυτών των προτύπων από συγκεκριμένες πλατφόρμες, κυρίως για λόγους απόδοσης, δεν έχουν συμπεριληφθεί στην κύρια έκδοση του Java API.

Η βασική κλάση για γραφικά στην Java είναι η κλάση `Graphics`. Τα αντικείμενα της κλάσης αυτής ονομάζονται *πλαίσια γραφικών* (graphics contexts) και αντιπροσωπεύουν μια επιφάνεια πάνω στην οποία μπορεί κανείς να σχεδιάσει, είτε στην οθόνη, είτε σε κάποια δομή στη μνήμη. Τα πλαίσια γραφικών παρέχουν μεθόδους για όλες τις βασικές γραφικές λειτουργίες, κρατώντας παράλληλα πληροφορίες κατάστασης για γραμματοσειρές, χρώματα, περιοχές ψαλιδισμού (clipping regions), κ.λπ.

Υπάρχουν διάφοροι τρόποι για να πάρει κανείς ένα αντικείμενο τύπου `Graphics`. Ο πιο συχνός είναι μέσω της παραμέτρου των μεθόδων `paint` ή `update` ενός συστατικού. Αν έχει πρόσβαση σε ένα αντικείμενο τύπου `Graphics`, μπορεί κανείς να το χρησιμοποιήσει για να σχεδιάσει απλά γεωμετρικά σχήματα, κείμενο και εικόνες με τις μεθόδους που παρέχονται από την κλάση `Graphics`.

7.6.1. Μέθοδοι σχεδίασης

Οι μέθοδοι της κλάσης `Graphics` δουλεύουν σε ένα απλό σύστημα καρτεσιανών συντεταγμένων, του οποίου η αρχή των αξόνων μπορεί να μετατοπιστεί χρησιμοποιώντας την μέθοδο `translate`. Οι πιο σημαντικές διαθέσιμες μέθοδοι σχεδίασης δίνονται στον Πίνακα 7.4. Οι μέθοδοι των οποίων τα ονόματα αρχίζουν με `fill` είναι παρόμοιες με τις κοινές, με τη διαφορά ότι σχεδιάζουν τα αντίστοιχα σχήματα "γεμισμένα" με κάποιο χρώμα ή σχέδιο.

Μια πολύ χρήσιμη κλάση συστατικού που σχετίζεται με τη σχεδίαση γραφικών είναι η `Canvas`. Ένα αντικείμενο τύπου `Canvas` είναι μια απλή περιοχή σχεδίασης, που από μόνη της δεν έχει καμία λειτουργικότητα. Μπορεί όμως κανείς να το χρησιμοποιήσει ως κλάση βάση, εξειδικεύοντας τη μέθοδο `paint` και παρέχοντας χειριστές γεγονότων. Με τον τρόπο αυτό χρησιμοποιείται η κλάση `Canvas` στο παράδειγμά μας.

7.6.2. Χρώματα

Το AWT χειρίζεται τα χρώματα μέσω της κλάσης `Color`. Ένα αντικείμενο της κλάσης αυτής αντιπροσωπεύει ένα χρώμα. Η `Color` παρέχει κατασκευαστές χρωμάτων με βάση τις συνιστώσες τους στο σύστημα RGB (Red, Green, Blue). Οι τιμές των συνιστωσών κυμαίνονται από 0 ως 255, ή από 0.0 ως 1.0 για αριθμούς κινητής υποδιαστολής. Η στατική μέθοδος `getColor` δέχεται ως παράμετρο το "όνομα" ενός χρώματος και επιστρέφει το αντίστοιχο χρώμα, όπως το αντιλαμβάνεται το λειτουργικό σύστημα. Για λόγους ευκολίας η κλάση `Color` ορίζει αρκετές σταθερές τιμές, που αντιστοιχούν σε συχνά χρησιμοποιούμενα χρώματα όπως το λευκό (`Color.white`), το μαύρο (`Color.black`) και το κόκκινο (`Color.red`).

Πίνακας 7.4. Μέθοδοι της κλάσης **`Graphics`.**

Μέθοδος	***Περιγραφή***
`drawLine`	Σχεδίαση γραμμής.
`drawImage`	Απεικόνιση εικόνας.
`drawString`	Απεικόνιση του κειμένου μιας συμβολοσειράς.
`drawOval` `fillOval`	Σχεδίαση έλλειψης.
`drawArc` `fillArc`	Σχεδίαση τόξου κύκλου.
`drawPolygon` `fillPolygon`	Σχεδίαση πολυγώνου.
`drawRect` `fillRect`	Σχεδίαση παραλληλογράμμου.
`drawRoundRect` `fillRoundRect`	Σχεδίαση παραλληλογράμμου με στρογγυλεμένες γωνίες.

Μέθοδος	***Περιγραφή***
`draw3DRect` `fill3DRect`	Σχεδίαση ενός παραλληλογράμμου σε 3 διαστάσεις.

Κάθε πλαίσιο γραφικών έχει πάντα ένα τρέχον χρώμα, με το οποίο δουλεύουν όλες οι μέθοδοι σχεδιασμού που αναφέρθηκαν παραπάνω. Για να το μεταβάλλουμε, καλούμε την συνάρτηση `setColor`, όπως φαίνεται στο παρακάτω παράδειγμα, που ζωγραφίζει μια πράσινη γραμμή:

```
public void paint (Graphics g)
{
    g.setColor(Color.green);
    g.drawLine(0, 0, 100, 100);
}
```

7.6.3. Γραμματοσειρές

Το AWT παριστάνει τις *γραμματοσειρές* (fonts) ως αντικείμενα της κλάσης `Font`. Ο κατασκευαστής της `Font` κατασκευάζει γραμματοσειρές με παραμέτρους το όνομα της γραμματοσειράς, τη μορφή και το μέγεθος σε points.

Το όνομα της γραμματοσειράς είναι μια συμβολοσειρά που αντιπροσωπεύει την οικογένεια γραμματοσειρών. Οι γραμματοσειρές είναι μια βασική πηγή προβλημάτων μεταφερσιμότητας εφαρμογών. Για το λόγο αυτό, οι οικογένειες γραμματοσειρών Dialog, Helvetica, TimesRoman, Courier και Symbol είναι διαθέσιμες σε όλες τις πλατφόρμες. Το όνομα της γραμματοσειράς στη Java μεταφράζεται κάθε φορά σε μια πραγματική γραμματοσειρά για κάθε τοπική πλατφόρμα. Η μέθοδος `getFont` δέχεται ως παράμετρο το όνομα μιας γραμματοσειράς, το αναζητά στον πίνακα με τις ιδιότητες του συστήματος και επιστρέφει το αντίστοιχο αντικείμενο `Font`. Όσον αφορά τη μορφή της γραμματοσειράς, είναι πάντα διαθέσιμες οι μορφές `PLAIN`, `BOLD` και `ITALIC`. Αν το μέγεθος σε points της γραμματοσειράς που ζητάμε δεν είναι διαθέσιμο η κλάση `Font` θα το αντικαταστήσει με ένα προκαθορισμένο μέγεθος.

Όπως και στην περίπτωση των χρωμάτων, κάθε αντικείμενο τύπου `Graphics` έχει μια τρέχουσα γραμματοσειρά, που χρησιμοποιείται από την `drawString`. Αυτή μπορεί να μεταβληθεί καλώντας τη μέθοδο `setFont`, όπως φαίνεται στο παράδειγμα που ακολουθεί:

```
...
gc.setFont(new Font("TimesRoman", Font.BOLD, 14);
gc.drawString("Hello!", 50, 50);
...
```

Για να πάρει κανείς συγκεκριμένες πληροφορίες για τα χαρακτηριστικά μιας γραμματοσειράς στο σύστημα, το AWT παρέχει την κλάση `FontMetrics`, που περιέχει πληροφορίες για το πως το συγκεκριμένο σύστημα απεικονίζει κάποια γραμματοσειρά. Οι πληροφορίες αυτές είναι χρήσιμες π.χ. στην περίπτωση που ο προγραμματιστής θέλει να συνδυάσει με ακρίβεια το κείμενο του με κάποια εικόνα ή με γραφικά.

7.6.4. Εικόνες

Η Java υποστηρίζει την εμφάνιση εικόνων κωδικοποίησης GIF ή JPEG, μέσω της κλάσης `Image`. Κάθε αντικείμενο αυτής της κλάσης παριστάνει μια εικόνα, που είτε βρίσκεται αποθηκευμένη σε κάποιο αρχείο, είτε προέρχεται από κάποια δυναμική πηγή, όπως ένα video.

Ένα applet μπορεί να κατασκευάσει ένα αντικείμενο τύπου `Image` καλώντας τη μέθοδο `getImage` και δίνοντας ως παράμετρο ένα URL, απόλυτο ή σχετικό. Αυτό φαίνεται στο ακόλουθο παράδειγμα:

```
Image img = getImage("image.gif");
```

Αν το αρχείο που περιέχει την εικόνα βρίσκεται στον ίδιο χώρο με τον κώδικα του applet, μπορεί να χρησιμοποιηθεί η μέθοδος `getCodeBase` για τον προσδιορισμό της θέσης της εικόνας:

```
Image img = getImage(getCodeBase(), "image.gif");
```

Στη συνέχεια, η εικόνα μπορεί να σχεδιασθεί καλώντας τη μέθοδο drawImage, η οποία στην απλούστερη μορφή της δέχεται τέσσερεις παραμέτρους: την εικόνα, τις συντεταγμένες όπου θα την απεικονίσει, και μια αναφορά σε κάποιον *παρατηρητή εικόνας* (image observer).

Η επεξεργασία των εικόνων στη Java γίνεται ασύγχρονα, που σημαίνει ότι οι σχετικές μέθοδοι (π.χ. `getImage`, `drawImage`) επιστρέφουν αμέσως και η Java κάνει τη σχετική δουλειά (π.χ. μεταφορά από το δίκτυο, απεικόνιση) σε ξεχωριστά νήματα εκτέλεσης, χωρίς να καθυστερεί το υπόλοιπο πρόγραμμα. Για παράδειγμα η `drawImage`, αν δεν έχει έρθει όλοκληρο το αρχείο της εικόνας από το δίκτυο, απεικονίζει το κομμάτι που είναι διαθέσιμο και επιστρέφει αμέσως. Αυτή η τεχνική βελτιώνει σημαντικά την απόδοση ενός Java applet. Οι παρατηρητές εικόνας χρησιμοποιούνται προκειμένου να γίνουν οι απαραίτητες ενέργειες όταν ολοκληρωθεί η επεξεργασία των εικόνων.

Η τελευταία παράμετρος της `drawImage`, είναι ένα αντικείμενο που υλοποιεί τη διαπροσωπεία `ImageObserver`. Το αντικείμενο αυτό ενημερώνεται για το ποσοστό ολοκλήρωσης της διαδικασίας και είναι ελεύθερο να χειρισθεί την πληροφορία αυτή με όποιο τρόπο κρίνει σκόπιμο. Η πιο συχνή αντιμετώπιση είναι η περιοδική κλήση της `repaint`, ώστε να ανανεώνεται η εικόνα με τα καινούρια αποτελέσματα της επεξεργασίας. Αυτήν ακριβώς τη συμπεριφορά δίνει η κλάση `Component`, που υλοποιεί το `ImageObserver`, και αν αυτό είναι το επιθυμητό, αρκεί να καθορίζεται το this ως τελευταία παράμετρος της `drawImage`, όταν αυτή καλείται μέσα από ένα `Component`. Το ακόλουθο παράδειγμα υλοποιεί ένα πλήρες applet που εμφανίζει στην οθόνη μια εικόνα:

```
class MyApplet extends Applet
{
   public void paint (Graphics gc)
   {
      Image img = getImage("image.gif");

      drawImage(img, 0, 0, this);
   }
}
```

Η μέθοδος `drawImage` είναι υπερφορτωμένη και, μέσω αυτής, είναι διαθέσιμες πολλές παρεμφερείς μέθοδοι γραφικών. Για παράδειγμα, μπορεί κανείς να απεικονίσει μια εικόνα σε μεγέθυνση ή σμίκρυνση, καλώντας την `drawImage` ως εξής:

```
drawImage(img, x1, y1, x2, y2, this);
```

Η κλήση αυτή εμφανίζει την εικόνα στο ορθογώνιο που ορίζεται από τα σημεία (x_1, y_1) και (x_2, y_2), κάνοντας την απαραίτητη μεγέθυνση ή σμίκρυνση.

Η κλάση `Image` παρέχει επίσης διάφορες άλλες μεθόδους, που επιστρέφουν πληροφορίες για την εικόνα, όπως για παράδειγμα τις `getHeight` και `getWidth`, που επιστρέφουν το ύψος και πλάτος της εικόνας σε pixels. Για περισσότερες πληροφορίες, ο αναγνώστης παραπέμπεται στη σχετική τεκμηρίωση του AWT.

7.6.5. Ήχος

Το AWT υποστηρίζει ορισμένες στοιχειώδεις λειτουργίες για το χειρισμό ήχων. Η διαπροσωπεία `AudioClip`, που ορίζεται στο πακέτο `java.applet`, παριστάνει αντικείμενα που μπορούν να παράγουν ήχο. Ένα αντικείμενο που υλοποιεί το `AudioClip`, πρέπει να παρέχει τις μεθόδους `play`, `stop` και `loop`. Η πρώτη παίζει τον ήχο μια φορά, η δεύτερη σταματά τον ήχο και η τρίτη παίζει τον ήχο συνέχεια. Ένα applet μπορεί να καλέσει την μέθοδο `getAudioClip` για να ανακτήσει έναν ήχο, ώστε στην συνέχεια να μπορεί να χρησιμοποιήσει τις μεθόδους του `AudioClip`. Το επόμενο παράδειγμα υλοποιεί ένα πλήρες applet που παίζει έναν ήχο συνεχώς.

```
class MyApplet extends Applet
{
   MyApplet ()
   {
      AudioClip audio = getAudioClip("sound.au");

      audio.play();
   }
}
```

7.7. Αυτόνομες εφαρμογές

Συχνά θέλουμε ο κώδικας που γράφουμε να τρέχει τόσο μέσα σε περιηγητές ως applet, όσο και ως αυτόνομη εφαρμογή. Αυτό δεν είναι πάντοτε δυνατό, καθώς κάποιες λειτουργίες συνήθως απαγορεύονται για τα applets από τον *διαχειριστή ασφαλείας* (security manager) των περισσότερων περιηγητών. Τέτοιες λειτουργίες είναι το διάβασμα και το γράψιμο αρχείων του τοπικού συστήματος και η σύνδεση με εξυπηρετητές εκτός από αυτόν από όπου προήλθε το applet. Σε ορισμένους περιηγητές, μπορεί κανείς να ορίσει επακριβώς το τί επιτρέπεται και τί απαγορεύεται στα applets κατά περίπτωση, ελέγχοντας την προέλευση τους ή ζητώντας επιβεβαίωση από τον χρήστη. Αυτό όμως συνήθως δεν είναι καλή ιδέα...

Για να τρέχει ένα Java applet και ως ανεξάρτητη εφαρμογή απαιτείται η κατάλληλη μετατροπή του κώδικά του. Αυτό μπορεί να γίνει με μερικές πολύ απλές προσθήκες που θα δούμε στη συνέχεια. Τα βασικά σημεία που δημιουργούν προβλήματα είναι:

- Η κλάση `Applet` είναι υποκλάση της `Panel` και, ως εκ τούτου, ένα applet δεν μπορεί να υπάρχει ανεξάρτητο μέσα σε ένα παραθυρικό περιβάλλον, αλλά μόνο μέσα σε ένα `Frame`.
- Για να ξεκινήσει η λειτουργία ενός applet θα πρέπει να κληθούν οι μέθοδοι `init` και `start`. Αυτό στην περίπτωση του applet γίνεται από τον περιηγητή. Στην περίπτωση όμως της αυτόνομης εφαρμογής, πρέπει να γίνει ρητά από τον προγραμματιστή.
- Ένα applet δεν περιλαμβάνει την μέθοδο `main` που απαιτείται για να τρέξει ως ανεξάρτητη εφαρμογή, ούτε και παρέχει κάποιο τρόπο για να τερματιστεί η λειτουργία του.

Προκειμένου να λυθούν τα παραπάνω προβλήματα πρέπει συνήθως να γίνουν τα εξής:

- Να ορισθεί μια μέθοδος `main`, μέσα στην οποία γίνονται οι απαραίτητες αρχικοποιήσεις. Η `main` προφανώς εκτελείται μόνο όταν ο κώδικάς λειτουργεί ως ανεξάρτητη εφαρμογή.
- Μέσα στην `main` κατασκευάζεται το appet ως αντικείμενο και έπειτα καλούνται οι μέθοδοί του `init` και `start`.
- Τέλος, επίσης στη `main`, δημιουργείται ένα `Frame` και τοποθετείται το applet μέσα σε αυτό.

Στο ακόλουθο παράδειγμα φαίνονται οι απαραίτητες μετατροπές για να εκτελείται η κλάση `MyApplet` ως ανεξάρτητη εφαρμογή:

```
class MyApplet extends Applet
{
   private static boolean isApplication = false;
   private static Frame applicationFrame;

   public static void main (String args [])
   {
      MyApplet theApplet = new MyApplet();

      isApplication = true;
      applicationFrame = new Frame("MyApplet");
      applicationFrame.add("Center", theApplet);
      applicationFrame.pack();
      applicationFrame.show();
      theApplet.init();
      theApplet.start();
   }

   // Rest of MyApplet...
}
```

Η ύπαρξη της μεταβλητής `isApplication` μπορεί να χρησιμοποιηθεί μετέπειτα στις μεθόδους του applet, έτσι ώστε να είναι γνωστό αν το applet χρησιμοποιείται ως ανεξάρτητη εφαρμογή ή όχι. Για παράδειγμα, αν συμβαίνει αυτό, θα ήταν καλό να κατασκευάζεται ένα κουμπί “Quit” για τον τερματισμό της εφαρμογής, κάτι που είναι άχρηστο στην περίπτωση του applet.

7.8. Παράδειγμα: Το πλήρες πρόγραμμα σχεδίασης

Ο πλήρης κώδικας του παραδείγματος δίνεται στη συνέχεια. Το παράδειγμα αυτό υλοποιεί ένα απλό πρόγραμμα σχεδίασης και μπορεί να χρησιμοποιείται ως applet ή ως ανεξάρτητη εφαρμογή.

Εισαγωγή πακέτων

```
import java.awt.*;
import java.applet.Applet;
import java.util.*;
```

Κλάση Paint

```
public class Paint extends Applet
{
   static boolean isApplication = false;
   public static Vector shapes;
   PaintCanvas pcanv;
   ControlPanel cpanel;
   static Frame fr;

   public void init ()
   {
      Panel p = new Panel();
      shapes = new Vector();
      p.setLayout(new BorderLayout(5, 5));
      cpanel = new ControlPanel();
      pcanv = new PaintCanvas(cpanel);
      cpanel.setPCanvas(pcanv);
      if (isApplication)
         p.add("South", new Button("Quit"));
      p.add("North", cpanel);
      setBackground(Color.lightGray);
      setLayout(new BorderLayout(5, 5));
      add("East", p);
      add("Center", pcanv);
   }

   public boolean action (Event e, Object o)
   {
      if (e.target instanceof Button)
         System.exit(0);
      return true;
   }

   public static void main(String args[])
   {
      isApplication = true;
      fr = new Frame("Paint");

      Paint p = new Paint();

      p.init();
      p.start();
      fr.add("Center", p);
      fr.pack();
      fr.show();
   }
}
```

Κλάση ControlPanel

```
class ControlPanel extends Panel
{
   PaintCanvas pcanv;
```

```
Choice shapes;
Checkbox fill;
TextField text;
Button clear, about;
Color cols[] = { Color.white, Color.black, Color.gray,
                 Color.red, Color.green, Color.blue,
                 Color.orange, Color.yellow, Color.cyan,
                 Color.magenta, Color.lightGray, Color.pink };
CheckboxGroup cbg;

public ControlPanel ()
{
   setBackground(new Color(150, 150, 150));    // RGB

   GridBagLayout gridbag = new GridBagLayout();
   GridBagConstraints c = new GridBagConstraints();
   setLayout(gridbag);
   c.insets = new Insets(5, 5, 5, 5);
   c.fill = c.BOTH;

   cbg = new CheckboxGroup();
   for (int i = 0 ; i < cols.length ; i++) {
      c.gridwidth = ((i % 3) == 2) ? c.REMAINDER : 1;

      Checkbox cb = new Checkbox("", cbg, false);

      cb.setBackground(cols[i]);
      gridbag.setConstraints(cb, c);
      add(cb);
   }

   c.gridwidth = c.REMAINDER;
   shapes = new Choice();
   shapes.addItem("Rectangle");
   shapes.addItem("Ellipse");
   shapes.addItem("Line");
   shapes.addItem("Text");
   gridbag.setConstraints(shapes, c);
   add(shapes);

   fill = new Checkbox("Filled");
   gridbag.setConstraints(fill, c);
   add(fill);

   text = new TextField();
   gridbag.setConstraints(text, c);
   add(text);

   clear = new Button("Clear");
   gridbag.setConstraints(clear, c);
   add(clear);

   about = new Button("About");
   gridbag.setConstraints(about, c);
   add(about);
}

public void setPCanvas (PaintCanvas pc)
{
   pcanv = pc;
}

public boolean action (Event e, Object arg)
{
   if (e.target instanceof Button) {
      String  name = (String) arg;
```

```
            if (name.equals("Clear"))
               pcanv.clear();
            else
               new AboutDialog();
            return true;
         }
         return false;
      }

      public Color getColor ()
      {
         return cbg.getCurrent().getBackground();
      }
   }
```

Κλάση PaintCanvas

```
   class PaintCanvas extends Canvas
   {
      ControlPanel  cp;
      int startX, startY, currX, currY;
      boolean dragging = false;

      public PaintCanvas (ControlPanel cp)
      {
         this.cp = cp;
      }

      public void paint (Graphics g)
      {
         int  num = Paint.shapes.size();
         for (int i = 0 ; i < num ; i++) {
            Shape sh = (Shape) Paint.shapes.elementAt(i);

            sh.paint(g);
         }
         if (dragging) {
            g.setColor(Color.black);
            g.setXORMode(Color.white);
            g.drawRect(startX, startY, currX-startX, currY-startY);
            g.setPaintMode();
         }
      }

      public void clear ()
      {
         Paint.shapes.removeAllElements();
         repaint();
      }

      public boolean mouseDown (Event e, int x,int y)
      {
         dragging = true;
         startX = x;
         startY = y;
         return true;
      }

      public boolean mouseUp (Event e, int x,int y)
      {
         if (dragging) {
            String shpName =
               cp.shapes.getItem(cp.shapes.getSelectedIndex());
```

```
            Shape sh;

            if (shpName.equals("Rectangle"))
               sh = new Rect(startX, startY, x - startX, y -startY,
                             cp.getColor(), cp.fill.getState());
            else if (shpName.equals("Ellipse"))
               sh = new Oval(startX, startY, x - startX, y - startY,
                             cp.getColor(), cp.fill.getState());
            else if (shpName.equals("Line"))
               sh = new Line(startX, startY, x, y, cp.getColor());
            else
               sh = new Text(startX, startY, cp.getColor(),
                             cp.text.getText());
            Paint.shapes.addElement(sh);
            dragging = false;
            repaint();
         }
         return true;
      }

      public boolean mouseDrag (Event e, int x,int y)
      {
         if (dragging) {
            currX = x; currY = y;
            repaint();
         }
         return true;
      }

      public Dimension preferredSize ()
      {
         return minimumSize();
      }

      public Dimension minimumSize ()
      {
         return new Dimension(500, 500);
      }
   }
```

Κλάση Shape

```
   abstract class Shape
   {
      Color color = Color.black;
      int  x, y, w, h;

      public abstract void paint (Graphics g);
   }
```

Κλάση Rect

```
   class Rect extends Shape
   {
      boolean fill = false;

      public Rect (int x1, int y1, int w, int h, Color c,
                   boolean fill)
      {
         x = x1; y = y1;
         this.w = w; this.h = h;
```

```
      this.fill = fill;
      color = c;
   }

   public void paint (Graphics g)
   {
      g.setColor(color);
      if (fill)
         g.fillRect(x, y, w, h);
      else
         g.drawRect(x, y, w, h);
   }
}
```

Κλάση Oval

```
class Oval extends Shape
{
   boolean fill = false;

   public Oval (int x1, int y1, int w, int h, Color c,
                boolean fill)
   {
      x = x1; y = y1;
      this.w = w; this.h = h;
      this.fill = fill;
      color = c;
   }

   public void paint (Graphics g)
   {
      g.setColor(color);
      if (fill)
         g.fillOval(x, y, w, h);
      else
         g.drawOval(x, y, w, h);
   }
}
```

Κλάση Line

```
class Line extends Shape
{
   public Line (int x1, int y1, int x2, int y2, Color c)
   {
      x = x1; y = y1;
      w = x2; h = y2;
      color = c;
   }

   public void paint (Graphics g)
   {
      g.setColor(color);
      g.drawLine(x, y, w, h);
   }
}
```

Κλάση Text

```
class Text extends Shape
{
   String  text;

   public Text (int x1, int y1, Color c, String txt)
   {
      x = x1; y = y1;
      text = txt;
      color = c;
   }

   public Text(String txt)
   {
      text = txt;
   }

   public void paint (Graphics g)
   {
      g.setColor(color);
      g.drawString(text, x, y);
   }
}
```

Κλάση AboutDialog

```
class AboutDialog extends Frame
{
   TextArea  ta;

   public AboutDialog ()
   {
      super("About");
      setBackground(Color.lightGray);

      ta = new TextArea("Paint has been written by:\n" +
          "\tYannis Patiniotakis\tipatini@softlab.ece.ntua.gr\n" +
          "and\n"+
          "\tVassilis Papadimos\tvpapad@softlab.ece.ntua.gr\n\n");
      ta.setEditable(false);
      add("Center", ta);
      add("South", new Button("OK"));
      pack();
      show();
   }

   public boolean action (Event e, Object arg)
   {
      dispose();
      return true;
   }
}
```

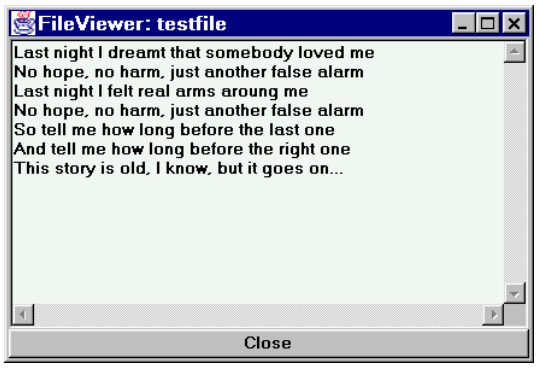
7.9. Παράδειγμα: Εμφάνιση αρχείων σε παράθυρο

Στην παράγραφο αυτή δίνεται ένα ολοκληρωμένο παράδειγμα χρήσης των κλάσεων `FileInputStream` και `File` για την ανάγνωση και εμφάνιση αρχείων σε ένα αντικείμενο `TextArea`.

Η κλάση `FileViewer` είναι βασικά σχεδιασμένη για χρήση της από άλλες κλάσεις. Παρόλα αυτά διαθέτει τη δική της μέθοδο `main`, ώστε να μπορεί να εκτελείται και ως αυτόνομο πρόγραμμα με την εντολή:

```
% java FileViewer testfile
```

Στο Σχήμα 7.3 φαίνεται το παράθυρο του `FileViewer` που δημιουργείται μετά την παραπάνω εκτέλεση.



Σχήμα 7.3. Εφαρμογή `FileViewer`.

Κλάση FileViewer

```
import java.awt.*;
import java.io.*;

public class FileViewer extends Frame
{
   Button close;

   // Κατασκευαστής για την κλάση FileViewer

   public FileViewer (String filename)
      throws IOException
   {
      super("FileViewer: " + filename);

      File f = new File(filename);

      int size = (int) f.length();
      int bytes_read = 0;
      FileInputStream in = new FileInputStream(f);
      byte[] data = new byte[size];

      while (bytes_read < size)
         bytes_read += in.read(data, bytes_read,
                               size - bytes_read);

      // Δημιουργία αντικειμένου TextArea για την
      // εμφάνιση του κειμένου
```

```
        TextArea textarea = new TextArea(
           new String(data, 0), 24, 80);

        textarea.setEditable(false);
        add("Center", textarea);

        // Δημιουργία "Close" button για το κλείσιμο
        // του παραθύρου.

        close = new Button("Close");
        add("South", close);
        pack();
     }

     // Χειρισμός του close button για το κλείσιμο
     // του παραθύρου

     public boolean action (Event e, Object what)
     {
        if (e.target == close) {
           dispose();
           return true;
        }
        return false;
     }

     // Ο FileViewer μπορεί να χρησιμοποιηθεί και σαν
     // αυτόνομη εφαρμογή με την ενσωμάτωση της παρακάτω
     // main() μεθόδου

     static public void main (String [] args)
     {
        if (args.length != 1) {
           System.out.println(
              "Usage: java FileViewer <filename>");
           System.exit(0);
        }
        try {
           new FileViewer(args[0]).show();
        }
        catch(IOException e) {
           System.out.println(e);
        }
     }
  }
```

Βιβλιογραφία

Έγγραφα προδιαγραφών: The Java Series

1. Ken Arnold and James Gosling, *The Java Programming Language*, Addison Wesley, ISBN 0-201-63455-4, 352 pages, paperback, 1996.

2. Mary Campione and Kathy Walrath, *The Java Tutorial: Object-Oriented Programming for the Internet*, Addison Wesley, ISBN 0-201-63454-6, 864 pages, paperback, 1996.

3. Patrick Chan and Rosanna Lee, *The Java Class Libraries: An Annotated Reference*, Addison Wesley, ISBN 0-201-63458-9, 1296 pages, hardback, 1997.

4. James Gosling, Bill Joy and Guy Steele, *The Java Language Specification*, Addison Wesley, ISBN 0-201-63451-1, 864 pages, paperback, 1997.

5. James Gosling, Frank Yellin and The Java Team, *The Java Application Programming Interface*, Volume 1: *Core Packages*, Addison Wesley, ISBN 0-201-63453-8, 544 pages, paperback, 1996.

6. James Gosling, Frank Yellin and The Java Team, *The Java Application Programming Interface*, Volume 2: *Window Toolkit and Applets*, Addison Wesley, ISBN 0-201-63459-7, 448 pages, paperback, 1996.

7. Doug Lea, *Concurrent Programming in Java: Design Principles and Patterns*, Addison Wesley, ISBN 0-201-69581-2, 352 pages, paperback, 1997.

8. Tim Lindholm and Frank Yellin, *The Java Virtual Machine Specification*, Addison Wesley, ISBN 0-201-63452-X, 496 pages, paperback, 1997.

Εισαγωγικά βιβλία

1. Ed Anuff, *The Java Sourcebook*, John Wiley and Sons, ISBN 0-471-14859-8, 1996.

2. Edith Au, Dave Makower and Pencom Web Works, *Java Programming Basics*, MIS Press, ISBN 1-55828-469-9, 1996.

3. Gary Cornell and Cay Horstman, *Core Java*, Prentice Hall, ISBN 0-13-565755-5 1996.

4. Ted Coombs, Jason Coombs and Don Brewer, *The Netscape LiveWire Sourcebook*, John Wiley and Sons, ISBN 0-471-15605-1, 1996.

5. John December, *Presenting Java*, Sams Net Publishing, 1995.

6. Harvey Deitel and Paul Deitel, *Java: How to Program*, Prentice Hall, ISBN 0-13-263401-5, 1997.

7. David Flanagan, *Java in a Nutshell*, O'Reilly, ISBN 1-56592-183-6, 1996.

8. Jerry Jackson and Alan McClellan, *Java By Example*, Prentice Hall, ISBN 0-13-565763-6, 1996.

9. Laura Lemay and Charles Perkins, *Teach Yourself Java in 21 Days*, Sams Net Publishing, ISBN 1-57521-030-4, 1996.

10. Michael Morrison, John December et al., *Java Unleashed*, Sams Net Publishing, ISBN 1-57521-049-5, 971 pages, 1997.

11. Patrick Naughton, *The JAVA Handbook*, Osborne McGraw-Hill, ISBN 0-07-882199-1, 1996.

12. Patrick Naughton and Herbert Schildt, *Java: The Complete Reference*, Osborne McGraw-Hill, ISBN 0-07-882231-9, 885 pages, 1997.

13. Patrick Niemeyer and Joshua Peck, *Exploring Java*, O'Reilly, ISBN 1-56592-184-4, 1996.

14. Alexander Newman et al., *Special Edition Using Java*, Que, ISBN 0-7897-0604-0, 1996.

15. Tim Richey, *Java!*, New Riders Publishing, ISBN 1-56205-533-X, 1995.

16. Ed Tittel and Mark Gaither, *The Official Internet World: 60 Minute Guide to Java*, IDG Books, 1995.

17. Paul Tyma et al., *Java Primer Plus*, Waite Group Press, ISBN 1-57169-062-X, 1996.

18. Peter van der Linden, *Just JAVA*, Prentice Hall, ISBN 0-13-565839-X, 1996.

19. Arthur van Hoff et al., *Hooked on Java*, Addison Wesley, 1996.

Χρήσιμα URL

1. http://www.javasoft.com/
   Το κεντρικό site της Sun για τη Java. Εδώ μπορεί κανείς να διαβάσει πληροφορίες για τη γλώσσα, όπως και να προμηθευθεί εργαλεία και τεκμηρίωση.

2. http://www.gamelan.com/
   Gamelan: the Official Directory for Java. Εδώ μπορεί κανείς να βρει μια μεγάλη συλλογή από χρήσιμα και άχρηστα applets.

3. http://www.java.co.uk/
   The Java Centre. Άλλη μια συλλογή από applets και κώδικα Java.

4. http://www.jars.com/
   The Java Review Service. Κριτική των Java applets που βρίσκονται στο WWW, από ένα σύνολο αντικειμενικών κριτικών.

5. http://www.yahoo.com/Computers_and_Internet/
   Programming_Languages/Java/
   Yahoo: Java. Μεγάλη συλλογή από δείκτες σε πληροφορίες σχετικές με τη Java.

6. http://java.wiwi.uni-frankfurt.de/
   The Java Repository. Μια συλλογή από δείκτες σε πληροφορίες σχετικές με τη γλώσσα Java.

7. http://apollo.mctr.umbc.edu:1080/
   Java Resource Page. Άλλη μια συλλογή από δείκτες σε πληροφορίες.