Εισαγωγή στη Java

©2004 Κωνσταντίνος Μαργαρίτης, markos@debian.org

1.Σχετικά με τη Java

Η Java είναι μια σύγχρονη αντικειμενοστραφής (object oriented) γλώσσα προγραμματισμού με αρκετά χαρακτηριστικά που δεν συναντώνται σε άλλες γλώσσες. Αναπτύχθηκε το 1991 από την εταιρεία Sun Microsystems με σκοπό τη δημιουργία μιας γλώσσας που θα τρέχει σε πληθώρα λειτουργικών συστημάτων και επεξεργαστών χωρίς αλλαγή.

Ως γλώσσα βασίστηκε στη C++ - για την ακρίβεια οι δύο γλώσσες έχουν την ίδια σύνταξη – με αρκετά όμως πρόσθετα χαρακτηριστικά και διαφοροποιήσεις.

Ιδιομορφίες της Java

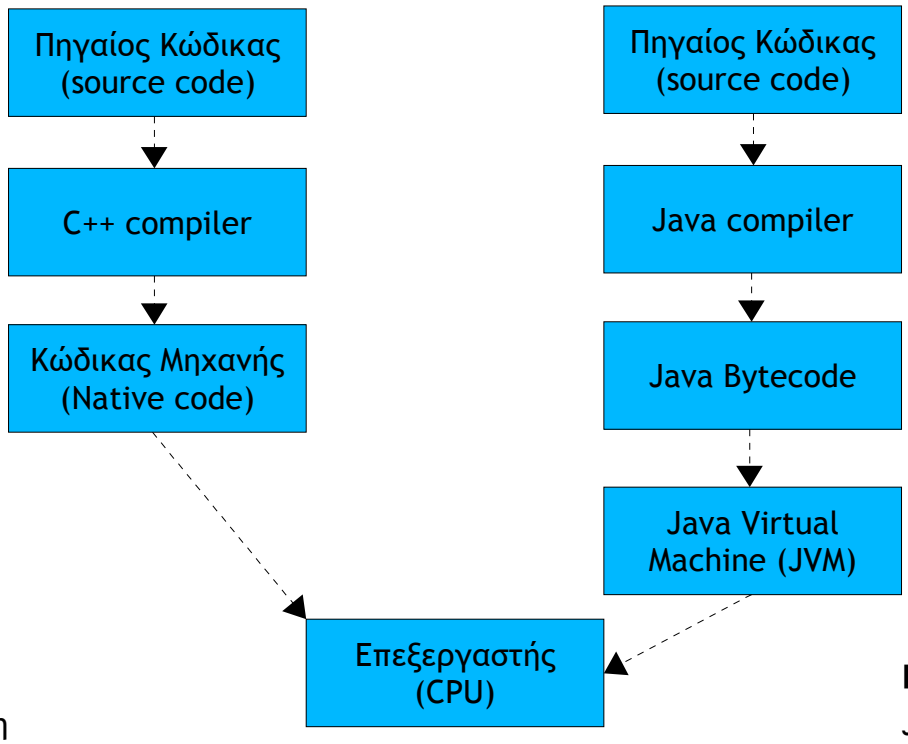
Σε αντίθεση με τις υπόλοιπες γλώσσες προγραμματισμού η Java έχει σχεδιαστεί ώστε ένα πρόγραμμα Java να μπορεί να τρέχει σε κάθε αρχιτεκτονική (δηλαδή συνδυασμό επεξεργαστή και λειτουργικού συστήματος) χωρίς την ανάγκη για εκ νέου μεταγλώττιση. Η μόνη απαίτηση είναι η ύπαρξη ενός Java Virtual Machine (JVM) για τη συγκεκριμένη αρχιτεκτονική.

Μια συνηθισμένη γλώσσα προγραμματισμού (π.χ. C, C++, PASCAL) μεταγλωττίζει τον πηγαίο κώδικα του προγράμματος σε εκτελέσιμη μορφή που να καταλαβαίνει ο επεξεργαστής. Η μορφή αυτή είναι η γλώσσα μηχανής, και διαφέρει για κάθε εξεπεργαστή και αρχιτεκτονική. Η εκτελέσιμη μορφή ενός προγράμματος για έναν συγκεκριμένο επεξεργαστή δεν είναι δυνατό να χρησιμοποιηθεί σε διαφορετικής αρχιτεκτονικής επεξεργαστή, δηλαδή δεν μπορούμε να χρησιμοποιήσουμε το εκτελέσιμο πρόγραμμα για Windows σε ένα σύστημα Macintosh. Έτσι, είναι απαραίτητη η εκ νέου μεταγλώττιση του πηγαίου κώδικα (source code) για κάθε αρχιτεκτονική στην οποία θέλουμε να τρέξουμε το πρόγραμμα.

Αντίθετα, η Java μεταγλωττίζει τον πηγαίο κώδικα σε μια εξειδικευμένη κωδικοποιημένη μορφή, το bytecode. Η μορφή αυτή είναι σαν την γλώσσα μηχανής ενός υποθετικού επεξεργαστή, που δεν υπάρχει στη πραγματικότητα. Αυτός ο εικονικός επεξεργαστής είναι το Java Virtual Machine (JVM) και εκτελεί το Java bytecode. Το JVM υλοποιείται ως πρόγραμμα σε κάθε πραγματικό λειτουργικό σύστημα και αναλαμβάνει να μετατρέψει κάθε εντολή του bytecode σε εντολή του πραγματικού επεξεργαστή (java bytecode to native code translation, όπως λέγεται η διαδικασία). Καθώς η διαδικασία αυτή γίνεται κατά την εκτέλεση του προγράμματος της Java, αυτός είναι και ένας λόγος που η Java είναι γενικά πιο αργή από τις παραδοσιακές γλώσσες προγραμματισμού (π.χ. C++).

Αν και με σύγχρονες τεχνικές, όπως οι Just In Time (JIT) compilers και εξελιγμένο scheduling των εντολών στο JVM, η απόδοση της Java έχει βελτιωθεί σε μεγάλο βαθμό (ένα καλογραμμένο πρόγραμμα Java μπορεί να φτάσει το 90% της απόδοσης ενός αντίστοιχου προγράμματος σε C++), λόγω της φύσης της δε θα πάψει να είναι πιο αργή από αυτές τις παραδοσιακές γλώσσες.

Στο παρακάτω σχήμα απεικονίζεται η διαδικασία της μεταγλώττισης ενός προγράμματος C++ και ενός προγράμματος Java.



Διαφορές με C++

Αν και η Java έχει βασιστεί σε μεγάλο βαθμό στη C++ έχει ορισμένες σημαντικές διαφορές που πρέπει να σημειωθούν. Συγκεκριμένα:

Java	*C++*
Παράγει Java Bytecode	Παράγει Native code
Portable	Χρειάζεται εκ νέου μεταγλώττιση
Ασφαλής (τρέχει μέσω του JVM)	Μόνο όσους περιορισμούς ορίζει το λειτουργικό σύστημα
Εκτέλεση κώδικα Java από το Web, μέσω Applets	Δεν είναι δυνατή η εκτέλεση κώδικα μέσα από το Web

Διαδικαστικός ή Δομημένος Προγραμματισμός (Procedural ή Structured Programming)

Οι παλαιότερες γλώσσες προγραμματισμού όπως οι C, PASCAL, FORTRAN έδιναν έμφαση στην διαδικασία και στα στάδια που ακολουθούνται για την επίτευξη κάποιου στόχου. Το αντικείμενο ήταν ο κώδικας (code-centric γλώσσες προγραμματισμού). Ο προγραμματισμός γινόταν με τον καθορισμό της ροής εκτέλεσης από κάποιο στάδιο Α σε κάποιο στάδιο Β, και την υλοποίηση των αντίστοιχων υπορουτινών.

Αντικειμενοστραφής προγραμματισμός (Object-Oriented Programming)

Οι αντικειμενοστραφείς γλώσσες προγραμματισμού (C++, Eiffel, Smalltalk και φυσικά Java) δίνουν έμφαση στα δεδομένα παρά στον κώδικα. Το πρόγραμμα αναπτύσσεται γύρω από τα δεδομένα (data-centric) τα οποία ορίζουν από μόνα τους τον τρόπο με τον οποίο μπορουμε να τα διαχειριστούμε.

Ο φυσικός και ο τεχνητός κόσμος που ζούμε είναι πιο κοντά στη φιλοσοφία του Αντικειμενοστραφή προγραμματισμού παρά του Δομημένου προγραμματισμού.

Ένα απλό παράδειγμα που μπορούμε να χρησιμοποιήσουμε για την κατανόηση της φιλοσοφίας του αντικειμενοστραφούς προγραμματισμού είναι το αυτοκίνητο.

Κάθε αυτοκίνητο είναι ένα αντικείμενο που ανήκει σε μια κλάση που ορίζει τα βασικά χαρακτηριστικά του αυτοκινήτου. Αυτά μπορεί να διαφέρουν ανάμεσα στους κατασκευαστές αλλά όλα θα παρέχουν τα βασικά χαρακτηριστικά που ορίζει η κλάση "αυτοκίνητο" για τη χρήση του. Δηλαδή τιμόνι, γκάζι, φρένο, συμπλέκτης και ταχύτητες. Αυτά είναι τα δεδομένα. Κάθε ένα από αυτά ορίζει τον τρόπο χρήσης του. Το τιμόνι στρίβει αριστερά/δεξιά, τα πεντάλ πιέζονται ή αφήνονται και οι ταχύτητες αλλάζουν διακριτά έχοντας μηχανισμό ασφαλείας – μπορούμε να αλλάξουμε ταχύτητα σε όπισθεν ενώ το αυτοκίνητο κινείται με ταχύτητα.

Η υλοποίηση καθενός από αυτούς τους μηχανισμούς διαφέρει σε κάθε κατασκευαστή, αλλά η χρήση τους είναι η ίδια για όλους. Δηλαδή η χρήση του τιμονιού και των ταχυτήτων γίνεται με τον ίδιο τρόπο ανεξαρτήτως κατηγορίας, κατασκευαστή και μοντέλου του αυτοκινήτου. Επίσης, ο μηχανισμός με τον οποίο γίνεται η χρήση των δεδομένων αυτών είναι κρυμμένος από τον χρήστη (δηλαδή τον οδηγό). Ο χρήστης δεν ενδιαφέρεται για τον τρόπο μετάδοσης της κίνησης από το τιμόνι στους τροχούς ώστε το αυτοκίνητο να στρίψει. Επίσης, η χρήση ενός αυτοκινήτου δεν αλλάζει με την επέκτασή του ή αλλαγή ορισμένων χαρακτηριστικών του (όπως π.χ. νέα μηχανή, λάστιχα, κλπ). Αλλάζει η συμπεριφορά του αλλά όχι η χρήση του.

Με αυτόν τον τρόπο, περιγράψαμε με ένα απλό παράδειγμα τα πιο σημαντικά χαρακτηριστικά του αντικειμενοστραφούς προγραμματισμού:

- Encapsulation ()

  Η διαδικασίες κρύβονται από το χρήστη και τα ίδια τα δεδομένα προσδιορίζουν τους τρόπους διαχείρισης τους.

- Polymorphism (πολυμορφισμός)

  Αντικείμενα που ανήκουν σε παρόμοιες κλάσεις μπορούν να έχουν κοινό τρόπο προσπέλασης, με αποτέλεσμα ο χρήστης να μπορεί να τα χειριστεί με τον ίδιο τρόπο χωρίς να χρειάζεται να μάθει νέες διαδικασίες.

- Inheritance (κληρονομικότητα)

  Μπορούμε να δημιουργήσουμε ένα νέο αντικείμενο παίρνοντας ως βάση ένα άλλο ήδη υπάρχον. Το νέο αντικείμενο θα έχει τα χαρακτηριστικά του παλιού ενώ θα μπορέι να τα τροποποιήσει, να τα επεκτείνει και να προσθέσει καινούρια για να καλύψει συγκεκριμένες ανάγκες.

2.Δομή της Java

Όπως και κάθε γλώσσα προγραμματισμού, έτσι και η Java ακολουθεί κάποιους κανόνες στη σύνταξη και υλοποίηση ενός προγράμματος. Οι κανόνες αυτοί αφορούν τη δήλωση μεταβλητών, τους τύπους δεδομένων, τους ορισμούς κλάσεων κλπ.

Μεταβλητές

Η έννοια της μεταβλητής είναι ίσως η πιο θεμελιώδης στον προγραμματισμό γενικά. Η μεταβλητή δεν είναι άλλο από μια ονομασμένη θέση μνήμης που μπορεί να περιέχει δεδομένα οποιουδήποτε είδους. Το περιεχόμενο της μεταβλητής μπορεί να μεταβάλλεται -εξ' ού και το όνομά της- με δεδομένα παρόμοιου είδους.

Οι μεταβλητές ορίζονται ως εξής:

```
type var;
```

όπου `type` το είδος της μεταβλητής και `var` το όνομά της. Ως είδος μπορούμε να διαλέξουμε έναν από τους διαθέσιμους τύπους δεδομένων της Java ή το όνομα μιας κλάσης αντικειμένων (σε επόμενη παράγραφο).

Έυρος μεταβλητών

Μια μεταβλητή θεωρείται έγκυρη, δηλαδή μια αναφορά σε αυτήν έχει νόημα μόνο μέσα στο πλάισιο στο οποίο ορίζεται. Το πλαίσιο μπορεί να είναι μια κλάση, μια συνάρτηση ή μέθοδος, το πεδίο ισχύος ενός loop `for/while/do` ή των εντολών `if/else/switch`.

Παράδειγμα:

```
type var1;
if (συνθήκη) {
   // Επεξεργασία της var1 (σωστό).
   // Επεξεργασία της var2 (λάθος).
   type var2;
   // Επεξεργασία της var2 (σωστό)
}
// Επεξεργασία της var1 (σωστό)
// Επεξεργασία της var2 (λάθος)
```

Τύποι δεδομένων

Οι βασικοί τύποι δεδομένων στη Java είναι σχεδόν οι ίδιοι με αυτούς της C και C++ (καθώς η Java έχει βασιστεί σε μεγάλο βαθμό στη C++). Πέρα από μικρές διαφορές στους ίδιους τους τύπους, είναι σχετικά εύκολο για κάποιον προγραμματιστή C++ να μεταβεί στη Java και να κρατήσει παρόμοια τη δομή ενός προγράμματος και ένας από τους λόγους που γίνεται αυτό είναι η ομοιότητα στους τύπους δεδομένων.

Η πιο σημαντική διαφορά είναι ότι η Java δεν έχει pointers.

Ακολουθεί πίνακας με τους διαθέσιμους τύπους δεδομένων της Java:

Όνομα	*Μέγεθος (σε bytes)*	*Όρια*
`byte`	1	-128 έως 127
`short`	2	-32768 έως 32767
`int`	4	-2147483648 έως -2147483647
`long`	8	-9223372036854775808 έως -9223372036854775807
`float`	4	$1.4 * 10^{-45}$ έως $3.4 * 10^{38}$
`double`	8	$4.9 * 10^{-324}$ έως $1.8 * 10^{308}$
`bool`	1	true / false
`char`	2	-
`String`	μεταβλητό	-

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

Παραδείγματα δηλώσεων και αρχικοποιήσεων μεταβλητών:

```
int an_integer;
an_integer = 10;
long a_long = an_integer *1000;
double verysmallnumber = 0.000000000003;
bool am_i_hungry = false;
char alpha = 'a';
String text = "this is a text";
```

Σταθερές (constants)

Αν και οι μεταβλητές έχουν σχεδιαστεί να μεταβάλλονται υπάρχουν περιπτώσεις που χρειαζόμαστε κάποιες σταθερές, όπως στα Μαθηματικά. Μπορούμε να δηλώσουμε μια μεταβλητή ως σταθερά (δηλαδή που να μη μεταβάλλεται) με τη λέξη `final`. Δηλαδή,

```
final double pi = 3.1415;
```

Τελεστές (operators)

Μπορούμε να εκτελέσουμε διάφορες πράξεις μεταξύ των μεταβλητών με τη χρήση των τελεστών. Οι τελεστές είναι σύμβολα που αντιστοιχούν σε αριθμητικές ή λογικές πράξεις μεταξύ των αντικειμένων. Υπάρχουν 4 είδη τελεστών στη Java (θα αναφέρουμε τα τρία πιο σημαντικά, και θα αφήσουμε τους δυαδικούς τελεστές προς το παρόν).

Αριθμητικοί τελεστές

Σύμβολο	*Είδος*
+	Πρόσθεση
-	Αφαίρεση
*	Πολλαπλασιασμός
/	Διαίρεση
%	Υπόλοιπο Διαίρεσης
++	Αυξητικός τελεστής
--	Αφαιρετικός τελεστής

Ειδικά για τους αυξητικούς και αφαιρετικούς τελεστές, πρέπει να αναφέρουμε ότι αυτοί αυξάνουν ή αφαιρούν την τιμή της μεταβλητής κατά μία μονάδα (κατά συνέπεια χρησιμοποιούνται μόνο σε ακέραιες μεταβλητές, όχι σε δεκαδικές). Παράδειγμα:

```
int x = 10;
x++; // τώρα η τιμή x έχει την τιμή 11
x--; // και πάλι την τιμή 10
```

Σχεσιακοί Τελεστές

Οι σχεσιακοί τελεστές αναφέρονται στην σχέση μεταξύ δύο αντικειμένων.

Σύμβολο	*Είδος*
==	Ισότητα
!=	Ανισότητα
>	Μεγαλύτερο από
<	Μικρότερο από
>=	Μεγαλύτερο από ή ίσο με
<=	Μικρότερο από ή ίσο με

Λογικοί Τελεστές

Οι λογικοί τελεστές αναφέρονται στους τρόπους συνδυασμών αληθών και ψευδών προτάσεων.

Σύμβολο	*Είδος*
&	AND (και)
|	OR (ή)
^	XOR (ή το ένα ή το άλλο)

Σύμβολο	*Είδος*
&&	Short-circuit AND
||	Short-circuit OR
!	NOT (άρνηση)

Η διαφορά του & και του && είναι ότι στη δεύτερη περίπτωση, αν με τον υπολογισμό της πρώτης παράστασης ο συνδυασμός βγαίνει αληθής ή ψευδής τότε αποφεύγεται ο υπολογισμός της δεύτερης.

Για παράδειγμα, αν έχουμε την παράσταση

```
(alpha == true) & (beta == 1)
```

το πρόγραμμα θα υπολογίσει και θα εξετάσει και τίς δύο προτάσεις ακόμη και αν η πρώτη (`alpha == true`) είναι ψευδής.

Με τη χρήση του &&, το πρόγραμμα θα επιστρέψει αμέσως ψευδή τιμή (false) αν η πρώτη είναι ψευδής, και θα προχωρήσει στον υπολογισμό της δεύτερης μόνο αν η πρώτη είναι αληθής.

Στην πραγματικότητα είναι θέμα εξοικονόμησης χρόνου.

Τελεστές καταχώρησης

Οι τελεστές καταχώρησης χρησιμοποιούνται για να μεταβάλουν τα περιεχόμενα μιας μεταβλητής. Μπορούν να συνδυαστούν και με άλλους τελεστές για συντόμευση κάποιων πράξεων. Παραδείγματα τελεστών καταχώρησης:

```
int x = 4;
x = 10;
x += 20; (είναι το ίδιο με την εντολή x = x + 20, τελικό αποτέλεσμα 30)
x /= 10; (το ίδιο με x = x / 10, αποτέλεσμα 3).
```

Σύμβολο	*Είδος*
x += y	το ίδιο με x = x + y
x -= y	το ίδιο με x = x - y
x *= y	το ίδιο με x = x * y
x /= y	το ίδιο με x = x / y
x %= y	το ίδιο με x = x % y
x &= y	το ίδιο με x = x & y
x |= y	το ίδιο με x = x | y
^=	το ίδιο με x = x ^ y

Παραστάσεις (Expressions)

Οποιοσδήποτε έγκυρος συνδυασμός μεταβλητών, σταθερών, αριθμών και τελεστών καλείται μια παράσταση. Το αποτέλεσμα της παράστασης μπορεί να είναι κάποιος από τους τύπους δεδομένων της Java (`int`, `long`, `double`, `bool`, κλπ) αλλά όχι κάποιο αντικείμενο ή `String`.

Παραδείγματα:

```
int b = 10, i;
i = 2*b*b; (παράδειγμα παράστασης)
if (b * b <= 100 && i > 0)
   System.out.println("The expression is true");
```

3.Ελεγχόμενη ροή προγράμματος

Σε κάθε πρόγραμμα είναι απαραίτητο να ελέγχουμε τη ροή του αναλόγως κάποιες συνθήκες και να την ανακατευθύνουμε κατάλληλα. Υπάρχουν πολλοί τρόποι να κατευθύνουμε τη ροή του προγράμματος και ο κάθε ένας εξυπηρετεί συγκεκριμένο σκοπό.

Η εντολή if

Η εντολή `if` χρησιμοποιείται όταν θέλουμε να εκτελέσουμε κάποιες εντολές μόνο όταν ικανοποιείται κάποια συνθήκη:

```
if (συνθήκη)
{
   εντολές;
}
else
{
   εντολές;
}
```

Μπορούν να συνδυαστούν πολλές εντολές `if` μεταξύ τους, όπως στο ακόλουθο παράδειγμα:

```
if (x == 1) {
   System.out.println("x is one.");
} else if (x == 2) {
   System.out.println("x is two.");
} else if (x == 3) {
   System.out.println("x is three.");
} else if (x == 4) {
   System.out.println("x is four.");
} else {
   System.out.println("x is not between 1-4.");
}
```

Η εντολή switch

Η εντολή `switch` χρησιμοποιείται όταν έχουμε πολλαπλές επιλογές ή τιμές για μια μεταβλητή και θέλουμε να εκτελεστούν διαφορετικές εντολές για κάθε τιμή. Με την `switch`, το προηγούμενο παράδειγμα θα πάρει τη μορφή:

```
switch (x) {
   case 1;
      System.out.println("x is one.");
      break;
   case 2;
      System.out.println("x is two.");
      break;
   case 3:
      System.out.println("x is three.");
      break;
   case 4:
      System.out.println("x is four.");
      break;
   default:
```

```
      System.out.println("x is not between 1-4.");
      break;
   }
```

Προσέξατε την χρήση της `break`. Χωρίς την `break` η εκτέλεση του προγράμματος θα συνεχίσει μέχρι την επόμενη `break` ή μέχρι το τέλος της εντολής `switch`. Αυτό μπορούμε να το εκμεταλευτούμε αν θέλουμε κοινή αντιμετώπιση ορισμένων περιπτώσεων.

```
switch (x) {
   case 1;
      System.out.println("x is one.");
   case 2;
      System.out.println("x is two.");
   case 3;
      System.out.println("x is three.");
   case 4;
      System.out.println("x is four.");
      System.out.println("x is between one and four.");
      break;
   case 5;
      System.out.println("x is five.");
   case 6;
      System.out.println("x is six.");
   case 7;
      System.out.println("x is seven.");
   case 8;
      System.out.println("x is eight.");
      System.out.println("x is between five and eight.");
      break;
   default:
      System.out.println("x is not between one and eight.");
      break;
   }
```

Η εντολή for

Η εντολή `for` χρησιμοποιείται για επανάληψη (loop) υπό συνθήκη κάποιων εντολών. Θεωρείται ίσως η πιο δυνατή και ευέλικτη εντολή τύπου `for`, σε οποιαδήποτε γλώσσα (έχει ακριβώς την ίδια λειτουργία με την αντίστοιχη της C/C++).

Η σύνταξή της είναι η εξής:

```
for (εντολή αρχικοποίηση; συνθήκη; εντολή επανάληψης) {
   εντολές;
}
```

Η εντολή αρχικοποίησης (initialization) εκτελείται στην αρχή του loop και στη συνέχεια ελέγχεται αν είναι αληθής ή ψευδής η συνθήκη (condition). Αν είναι αληθής, εκτελούνται οι εντολές και στο τέλος εκτελείται η εντολή επανάληψης (iteration) και ελέγχεται και πάλι αν είναι αληθής η συνθήκη. Αν είναι αληθής εκτελούνται οι εντολές, κ.ο.κ.

Παράδειγμα:

```
for (int i = 1; i < 20; i += 3)  {
   System.out.println(i);
}
```

Το αποτέλεσμα θα είναι το εξής:

```
1
4
7
10
13
16
19
```

Κάθε μία από τις εντολές αρχικοποίησης, επανάληψης και η συνθήκη μπορεί να παραλειφθεί αλλάζοντας τη συμπεριφορά της εντολής `for`. Σημειώνουμε ότι χωρίζονται μεταξύ τους με το ερωτηματικό ';'. Παραδείγματα:

```
int x = 10;
for (; x < 5; x++) {
   System.out.println(x);
}

double y = 20000; // (y = pi)
for (; y >= 10.0;) {
   // υπολογίζει την τετραγωνική του y και τοποθετεί
   // το αποτέλεσμα πάλι στο y.
   // Το for loop θα εκτελείται όσο το y είναι μεγαλύτερο από 10.0
   System.out.println(y);
   y = Math.sqrt(y);
}

for (;;) { // infinite loop
   wait_for_signal();
}
```

Η εντολή while

Η εντολή `while` χρησιμοποιείται αντί της `for` όταν δε μπορούμε να προβλέψουμε εύκολα πόσες φορές θέλουμε να εκτελεστούν οι εντολές, ή όταν δεν έχει σημασία ο αριθμός των επαναλήψεων αλλά η ικανοποιήση ή όχι της συνθήκης:

while (συνθήκη) {

    εντολές;

}

Παράδειγμα:

```
bool exit_from_loop = false;
while (exit_from_loop = false) {
   // υποθετική ρουτίνα που διαβάζει δεδομένα από ένα αρχείο
   read_bytes(file1);
   // άλλη ρουτίνα που γράφει δεδομένα σε ένα άλλο αρχείο
   write_bytes(file2);
```

```
   if (file_empty(file1) == true)
      exit_from_loop = true;
}
```

Η εντολή do/while

Η εντολή `do/while` είναι παρόμοια με την `while`, με μία διαφορά, οι εντολές εντός του loop θα εκτελεστούν τουλάχιστον μια φορά ανεξαρτήτως αν είναι αληθής η συνθήκη ή όχι.

```
do {
   εντολές;
} while (συνθήκη);
```

Παράδειγμα:

```
int x = 10;
do {
   System.out.println(x);
   x++;
} while (x <10);
```

Το αποτέλεσμα θα είναι:

```
10
```

Οι εντολές μέσα στο loop θα εκτελεστούν την πρώτη φορά ακόμη και αν δεν ισχύει η συνθήκη (x < 10).

Οι εντολές break/continue

Πολλές φορές θέλουμε να διακόψουμε την εκτέλεση των εντολών σε ένα loop πρίν από τη προκαθορισμένη στιγμή. Π.χ. να φύγουμε από ένα infinite loop, ή όταν το πρόγραμμα λάβει κάποιο σήμα (signal) για έξοδο (π.χ. Control-C).

```
for (int i = 1; i < 20; i += 3)  {
   if ( i*i > 100)
      break;
   System.out.println(i);
}
```

Το αποτέλεσμα θα είναι:

```
1
4
7
```

Η εντολή `break` χρησιμοποιείται σε όλα τα loops, `for`, `while`, `do/while`.

Αντίστοιχη με την `break` είναι η `continue`, η οποία όμως διακόπτει την εκτέλεση μόνο του τρέχονος iteration και όχι ολόκληρου του loop. Συνεχίζει από το επόμενο iteration.

Στο προηγούμενο παράδειγμα:

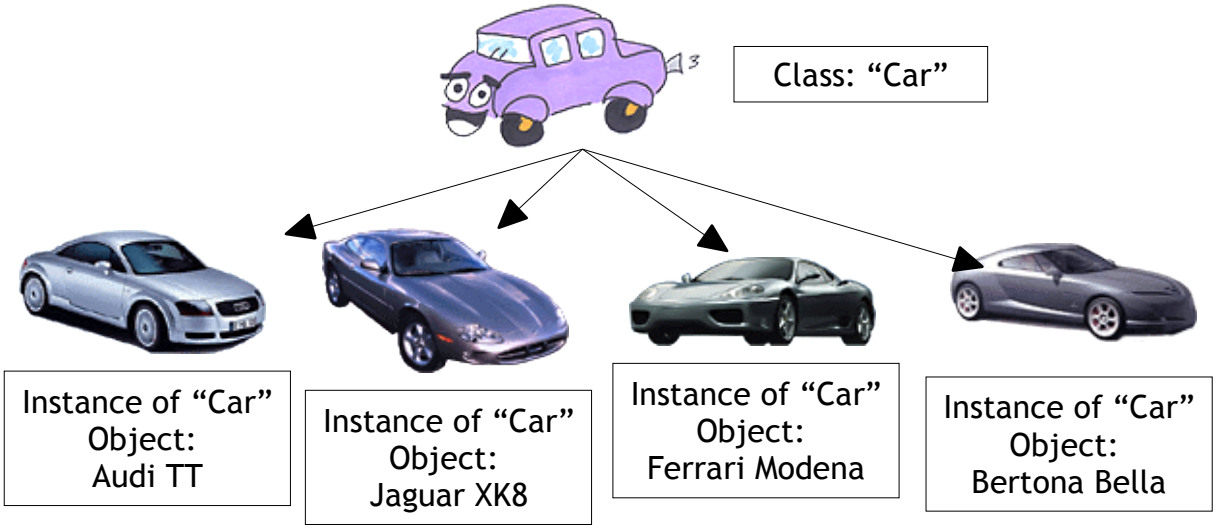
```
for (int i = 1; i < 20; i += 3)  {
   // να μην τυπωθούν οι άρτιοι(ζυγοί) αριθμοί
   if ( i % 2 = 0)
```

```
      continue;
   System.out.println(i);
}
```

Το αποτέλεσμα θα είναι:

```
1
7
13
19
```

4.Classes



Η αντικειμενοστραφής πλευρά της Java φαίνεται με την χρήση των αντικειμένων και των κλάσεων στις οποίες ανήκουν τα αντικείμενα. Όλα τα αντικείμενα που έχουν κοινά χαρακτηριστικά ανήκουν στην ίδια κλάση και κάθε ένα από αυτά λέγεται ότι είναι “στιγμιότυπο” (instance) της κλάσης. Για παράδειγμα, η κλάση “αυτοκίνητο” θεωρεί μια γενική εικόνα του αυτοκινήτου με κάποια χαρακτηριστικά που υπάρχουν σε όλα τα στιγμιότυπα. Συγκεκριμένα, η κλάση αυτοκίνητο πιθανώς να ορίζει ότι είναι τετράτροχο, έχει τιμόνι, ταχύτητες, πεντάλ και καθίσματα για τους οδηγούς αλλά δεν ορίζει με σαφήνεια το σχήμα ή τους μηχανισμούς όλων των εξαρτημάτων, ούτε τα χαρακτηριστικά της μηχανής (κυβικά, ίπποι, κύλινδροι, κλπ). Αυτά είναι χαρακτηριστικά που αφορούν το κάθε αντικείμενο ξεχωριστά (ή κάποια υποκατηγορία/υποκλάση της κλάσης “αυτοκίνητο”). Για καλύτερη κατανόηση, δείτε το παραπάνω σχήμα.

Στη Java (και στη C++ στην οποία βασίστηκε η Java) η κλάση δηλώνεται με τη λέξη `class`. Στο παράδειγμα του αυτοκινήτου, έστω ότι θέλουμε να δηλώσουμε μια κλάση με το όνομα Car:

```
class Car
{
   (δεδομένα/μεταβλητές)
   (συναρτήσεις/μέθοδοι)
}
```

member variables

Τα χαρακτηριστικά του αντικειμένου είναι τα δεδομένα γύρω από τα οποία πρέπει να αναπτύξουμε το πρόγραμμά μας. Στο παράδειγμα με το αυτοκίνητο, τα δεδομένα αυτά είναι: η γωνία στρέψης του τιμονιού, η πίεση που ασκούμε στα πεντάλ, η θέση του μοχλού ταχυτήτων και άλλα. Στον προγραμματισμό, αυτά τα δεδομένα θα πρέπει να μοντελοποιηθούν, δηλαδή να γίνει η αντιστοιχία τους σε μεταβλητές τις οποίες μπορούμε να επεξεργαστούμε στο πρόγραμμά μας.

Για την κλάση Car, μπορούμε να έχουμε:

```
class Car
{
   // Γωνία στρέψης του τιμονιού
   float steering_angle;

   // Ποσοστό πατήματος του γκαζιού (0 = καθόλου, 100 = τερματισμένο!)
   float gas_pedal;

   // Ποσοστό πατήματος του φρένου (0 = καθόλου, 100 = τερματισμένο!)
   float break_pedal;

   // Ποσοστό πατήματος του συμπλέκτη (0 = καθόλου,
   // 100 = τερματισμένο!)
   float clutch;

   // θέση της τρέχουσας ταχύτητα (πιθανές τιμές: 0, 1,2,3,4,5,
   // 0 = νεκρό, -1 = όπισθεν)
   int gear;

   // μεταβλητές που καθορίζουν την επιτάχυνση, την ταχύτητα του
   // αυτοκινήτου και τις στροφές του κινητήρα
   float acceleration, speed, rpm;
}
```

Εδώ πρέπει να σημειωθεί ότι οι μεταβλητές της κλάσης έχουν διαφορετικές τιμές για κάθε αντικείμενο, και κάθε αντικείμενο έχει πρόσβαση μόνο στις δικές του μεταβλητές. Επίσης, ο τύπος δεδομένων που επιλέχθηκε για κάθε μεταβλητή εξαρτάται από το είδος της πληροφορίας που θα κρατάει αυτή η μεταβλητή. Για παράδειγμα, εφόσον η γωνία στρέψης του τιμονιού θα είναι σε μοίρες, είναι λογικό να χρησιμοποιήσουμε ένα τύπο που θα μπορεί να κρατήσει δεκαδικούς αριθμούς, όπως ο `float`.

Αυτά, λοιπόν, είναι τα δεδομένα μας ομαδοποιημένα υπό την έννοια της κλάσης “Car”. Αυτή τη στιγμή δεν είναι τίποτε παραπάνω από μια ομάδα μεταβλητών χωρίς να έχουμε ορίσει τον τρόπο επεξεργασίας τους. Γι' αυτό είναι απαραίτητη η ύπαρξη του κατάλληλου interface των δεδομένων για τον χρήστη. Το interface επιτυγχάνεται με την ύπαρξη των μεθόδων της κλάσης (member methods).

member methods

Οι μέθοδοι (methods) σε μια κλάση, δεν είναι τίποτε παραπάνω από συναρτήσεις (υπορουτίνες) που προσφέρουν πρόσβαση στα δεδομένα του εκάστοτε αντικειμένου της κλάσης. Η αλληλεπίδραση του χρήστη με κάθε αντικείμενο γίνεται μέσω των μεθόδων της κλάσης, οι οποίες καθορίζουν και τον τρόπο χειρισμού των μεταβλητών του αντικειμένου. Στο παράδειγμά μας, οι μέθοδοι θα καθορίζουν με ποιον τρόπο θα στρίβουμε το τιμόνι, θα αυξάνουμε το πάτημα στο γκάζι ή στο φρένο, θα αλλάζουμε ταχύτητες αλλά και πώς θα μπορούμε να γνωρίζουμε την ταχύτητα του αυτοκινήτου, τις στροφές του κινητήρα δηλαδή πληροφορίες που δε μπορούμε να ελέγξουμε άμεσα.

Πιθανές μέθοδοι για την κλάση “Car” θα μπορούσαν να είναι οι εξής:

```
// Αλλαγή της γωνία στρέψης του τιμονιού, <relative_angle> μοίρες
// σε σχέση με την τρέχουσα γωνία.
void turn_wheel(float relative_angle);

// Πάτημα πεντάλ γκαζιού
void press_gas_pedal(float amount);

// Πάτημα πεντάλ φρένου
void press_break_pedal(float amount);

// Πάτημα πεντάλ συμπλέκτη
void press_clutch_pedal(float amount);

// Αλλαγή της ταχύτητας. Επιστρέφει true αν η αλλαγή ήταν επιτυχής
// ή false αν ήταν ανεπιτυχής (π.χ. από 5 σε όπισθεν).
bool change_gear(int new_gear);

// προβολή της τρέχουσας ταχύτητας, επιτάχυνσης και στροφών του
// κινητήρα
float get_acceleration();
float get_speed();
float get_rpm();
```

Οι παραπάνω μέθοδοι όπου κρίνεται αναγκαίο επιστρέφουν κάποιο αποτέλεσμα, είτε ως δεκαδικός αριθμός (`float`) στην περίπτωση των μεθόδων `get_*()`, ή ως `bool` στην `change_gear()`. Ο τύπος `void` είναι ειδική περίπτωση που δεν επιστρέφει κάποιο αποτέλεσμα. Χρησιμοποιείται όταν μας ενδιαφέρει η εκτέλεση κάποιας διαδικασίας χωρίς όμως να είναι απαραίτητο να γνωρίζουμε το αποτέλεσμά της, ή μπορεί να μην επιστρέφει κάποιο αποτέλεσμα εξ' αρχής.

Υλοποίηση μιας μεθόδου

Οι παραπάνω είναι απλώς οι δηλώσεις των μεθόδων, δηλαδή δεν περιέχουν καθόλου κώδικα. Για να είναι ολοκληρωμένος ο ορισμός μιας μεθόδου θα πρέπει να συνοδεύεται και από την υλοποίησή της (implementation). Η υλοποίηση μπορεί να γίνει ταυτόχρονα με τη δήλωση, για παράδειγμα η υλοποίηση της `turn_wheel()` θα μπορούσε να είναι:

```
void turn_wheel(float relative_angle)
{
   steering_angle += relative_angle;
   if (steering_angle <= -720.0)
       steering_angle = -720.0;
   if (steering_angle >= 720.0)
       steering_angle = 720.0;
}
```

Όπως βλέπουμε στην υλοποίηση της μεθόδου προσθέσαμε και επιπλέον κώδικα ασφαλείας, ο οποίος απαγορεύει στο τιμόνι να κάνει περισσότερες από 2 στροφές αριστερά ή δεξιά. Παρόμοιοι περιορισμοί σε ένα κανονικό αυτοκίνητο γίνονται με μηχανικό τρόπο, αλλά σε ένα πρόγραμμα (π.χ. εξομοιωτή οδήγησης), το κάθε αντικείμενο πρέπει να θέσει τους δικούς του περιορισμούς και δικλείδες ασφαλείας με τη μορφή κώδικα στις μεθόδους.

Δημιουργία αντικειμένων με τη new

Έχοντας ορίσει την κλάση μας, θα πρέπει να δημιουργήσουμε τα αντικείμενα -τα στιγμιότυπα (instances) της κλάσης. Για το σκοπό αυτό, χρησιμοποιούμε την εντολή `new` της Java.

Η `new` δημιουργεί μια φυσική αναπαράσταση της κλάσης, ένα μοναδικό στιγμιότυπο, και επιστρέφει ένα δείκτη αναφοράς (reference) σε αυτό. Με αυτό το δείκτη αναφοράς μπορούμε να προσπελάσουμε το αντικείμενο με οποίον τρόπο θέλουμε (και εφόσον το επιτρέπει το ίδιο το αντικείμενο). Ο τρόπος χρήσης της `new` είναι ο εξής:

```
Car acar = new Car();
Car anothercar = new Car();
```

Οι μεταβλητές acar και anothercar είναι οι δείκτες αναφοράς στα αντίστοιχα αντικείμενα που δημιουργούνται με τη new. Η δημιουργία του αντικείμένου δεν είναι αναγκαίο να γίνει κατά την δήλωσή του. Το ίδιο αποτέλεσμα μπορούμε να έχουμε και με τις εντολές:

```
Car acar;
acar = new Car();
```

Τα δεδομένα του κάθε αντικειμένου (οι μεταβλητές) αλλά και οι μέθοδοι της κλάσης που αντιστοιχούν στο αντικείμενο, μπορούν να προσπελλαστούν ως εξής:

```
// Η γωνία στρέψης του τιμονιού του acar
acar.steering_angle

// Η γωνία στρέψης του τιμονιού του anothercar
anothercar.steering_angle

// Εντολή στο acar να στρίψει δεξιά 13.4 μοίρες.
acar.turn(13.4);

// Επιστρέφει την τρέχουσα ταχύτητα του acar
float speed = acar.get_speed();

// Εντολή στο anothercar να στρίψει αριστερά 32 μοίρες
anothercar.turn(-32.0);

// Εντολή στο anothercar να βάλει όπισθεν
bool result = anothercar.ghange_gear(-1);
```

Όπως βλέπουμε ορισμένοι μέθοδοι δέχονται κάποιες παραμέτρους (εντός παρενθέσεων). Οι παράμετροι μπορεί να είναι μεταβλητές οποιουδήποτε αποδεκτού τύπου στη Java ή ακόμη και άλλα αντικείμενα. Αν έχουμε περισσότερες από μία παραμέτρους τις διαχωρίζουμε με κόμμα ','.

Constructors

Όταν δημιουργείται ένα αντικείμενο με την εντολή new, στην πραγματικότητα η Java, αφού δεσμεύσει την απαραίτητη μνήμη για το αντικείμενο, εκτελεί μια συγκεκριμένη μέθοδο της κλάσης, τον δημιουργό (constructor). Ο δημιουργός πραγματοποιεί τις απαραίτητες ενέργειες για να καταστεί κάποιο αντικείμενο έτοιμο προς χρήση. Αυτές μπορεί να είναι κάτι απλό όπως η ρύθμιση κάποιων μεταβλητών με αρχικές τιμές, ή πιο

περίπλοκο όπως η δημιουργία σύνδεσης με μια βάση δεδομένων, η αρχικοποίηση των πινάκων SQL, η δέσμευση κάποιων δικτυακών θυρών (sockets) για κάποιο server ή ακόμη και το άνοιγμα κάποιου παραθύρου για εμφάνιση γραφικής πληροφορίας.

Ο δημιουργός έχει τη μορφή μιας μεθόδου με το όνομα της κλάσης και χωρίς τύπο (δηλαδή δεν δηλώνεται ο τύπος δεδομένων που επιστρέφει, το οποίο είναι διαφορετικό από το να δηλωθεί ως void).

Για παράδειγμα, στην κλάση "Car", ένας πιθανός δημιουργός μπορεί να είναι:

```
Car()
{
   steering_wheel = 0.0;
   gas_pedal = 0.0;
   break_pedal = 0.0;
   float clutch = 0.0;
   int gear = 0;
   acceleration = 0.0;
   speed = 0.0;
   rpm = 0.0;
}
```

Δηλαδή, ο δημιουργός πραγματοποιεί την αρχικοποίηση (initialization) των μεταβλητών του αντικειμένου ώστε αυτό να είναι έτοιμο προς χρήση.

Μπορούμε να έχουμε περισσότερους από έναν δημιουργούς, οι οποίοι να δέχονται διαφορετικές παραμέτρους ο καθένας.

Για παράδειγμα, αν υποθέσουμε ότι μπορούσαμε να ορίσουμε χαρακτηριστικά του κινητήρα (`engine_cc`: κυβικά, `engine_hp`: ίπποι) στη δημιουργία του αντικειμένου, θα μπορούσαμε να έχουμε έναν επιπλέον δημιουργό:

```
Car(int cc, int hp)
{
   engine_cc = cc;
   engine_hp = hp;
   // Ακολουθούν οι υπόλοιπες εντολές αρχικοποίησης του αντικειμένου
}
```

επιπλέον του αρχικού δημιουργού.

Η δημιουργία περισσότερων από έναν δημιουργούς καλείται constructor overloading και είναι υποπερίπτωση του χαρακτηριστικού της Java, method overloading (σε επόμενη παράγραφο).

Που είναι οι Destructors? Garbage collection στη Java

Όσοι γνωρίζουν C++, θα θυμούνται πιθανόν ότι παράλληλα με τους δημιουργούς ενός αντικειμένου υπάρχουν και οι καταστροφείς του (destructors). Στη C++, ό,τι αντικείμενο δημιουργούμε πρέπει να το καταστρέφουμε (συνήθως με την εντολή `delete`) ελευθερώνοντας όλους τους πόρους που δεσμεύσαμε κατά την ύπαρξή του (κλείσιμο αρχείων, αποσύνδεση από βάσεις δεδομένων, τερματισμός threads, αποδέσμευση μνήμης, κλπ). Στη Java κάτι τέτοιο δεν είναι απαραίτητο, για την ακρίβεια δεν παρέχεται καν αυτή η δυνατότητα!

Η Java παρέχει ένα δικό της μηχανισμό "περισυλλογής σκουπιδιών" (garbage collection)

όπως λέγεται, με τον οποίο ελευθερώνει τα αντικείμενα που δεν χρησιμοποιούνται πλέον από το πρόγραμμα. Έτσι πλέον, δεν χρειάζεται αλλά και δεν έχουμε τη δυνατότητα να καταστρέψουμε πλέον κάποια αντικείμενα όπως σε άλλες αντικειμενοστραφείς γλώσσες (π.χ. C++).

Ο δείκτης αναφοράς this

Μέσα σε μια μέθοδο, μπορούμε να χρησιμοποιήσουμε μια μεταβλητή της κλάσης απλώς με το όνομά της, αναφερόμενοι φυσικά στην τιμή που έχει η μεταβλητή για το συγκεκριμένο αντικείμενο. Τις περισσότερες φορές κάτι τέτοιο είναι αρκετό, υπάρχουν όμως και περιπτώσεις που χρειάζεται πιο ρητή αναφορά στο συγκεκριμένο αντικείμενο. Για το σκοπό αυτό μπορούμε να χρησιμοποιήσουμε τη μεταβλητή `this` που επιστρέφει πάντα ένα δείκτη αναφοράς στο τρέχον αντικείμενο (δηλαδή αυτό που καλεί την μέθοδο). Με το δείκτη αναφοράς `this`, η μέθοδος `turn_wheel()` που είδαμε παραπάνω μετασχηματίζεται ως εξής:

```
void turn_wheel(float relative_angle)
{
   this.steering_angle += relative_angle;
   if (this.steering_angle <= -720.0)
       this.steering_angle = -720.0;
   if (this.steering_angle >= 720.0)
       this.steering_angle = 720.0;
}
```

Ο δείκτης αναφοράς `this` είναι ιδιαίτερα χρήσιμος ειδικά σε περιπτώσεις που μεταχείριζομαστε περισσότερα από ένα όμοια αντικείμενα στην ίδια μέθοδο, για αποφυγή σύγχυσης.

Method Overloading

Σε ένα πρόγραμμα, υπάρχει περίπτωση να χρειαστεί να εκτελέσουμε την ίδια διαδικασία με ελαφρώς διαφορετικά δεδομένα. Αυτό συνήθως σημαίνει ότι θα χρειαστεί να έχουμε διαφορετικές συναρτήσεις/μεθόδους για κάθε διαφορετική παράμετρο που δίνουμε. Για παράδειγμα, ας υποθέσουμε ότι ο χρήστης καλεί την μέθοδο `turn_wheel()` με παράμετρο ένα `int` αντί για `float`. Σε μια γλώσσα όπως η C θα έπρεπε να έχουμε δύο συναρτήσεις/μεθόδους, μια για κάθε διαφορετική παράμετρο και μάλιστα με διαφορετικό όνομα:

```
void turn_wheel_int(int relative_angle)
{
   this.steering_angle += (float) relative_angle;
   if (this.steering_angle <= -720.0)
       this.steering_angle = -720.0;
   if (this.steering_angle >= 720.0)
       this.steering_angle = 720.0;
}

void turn_wheel_float(float relative_angle)
{
   steering_angle += relative_angle;
   if (steering_angle <= -720.0)
       steering_angle = -720.0;
   if (steering_angle >= 720.0)
```

```
      steering_angle = 720.0;
}
```

Φυσικά, το συγκεκριμένο παράδειγμα είναι αρκετά απλό και δεν αποτελεί πρόβλημα. Μπορούμε μάλιστα να τροποποιήσουμε την `turn_wheel_int()` ως εξής:

```
void turn_wheel_int(int relative_angle)
{
   turn_wheel_float((float) relative_angle);
}
```

Στην πραγματικότητα, αν έχουμε κάποιο πιο περίπλοκο πρόγραμμα με μερικές δεκάδες παραλλαγές μεθόδων και κάποιες χιλιάδες γραμμές κώδικα, η παραπάνω τεχνική δυσκολεύει το debugging, προκαλεί διπλασιασμό του κώδικα και δυσχεραίνει τη συντήρηση του προγράμματος. Στην περίπτωση αυτή επιβάλλεται η χρήση μιας Object-Oriented γλώσας όπως η Java και η C++ που παρέχουν δυνατότητες Method Overloading. Η τεχνική του method overloading, επιτρέπει τον ορισμό διάφορων παραλλαγών μιας μεθόδου αναλόγως τις ζητούμενες παραμέτρους που δέχεται αυτή. Το παραπάνω παράδειγμα, θα μετασχηματιστεί στην εξής μορφή:

```
void turn_wheel(float relative_angle)
{
   steering_angle += relative_angle;
   if (steering_angle <= -720.0)
       steering_angle = -720.0;
   if (steering_angle >= 720.0)
       steering_angle = 720.0;
}

void turn_wheel(int relative_angle)
{
   turn_wheel((float) relative_angle);
}
```

Το όνομα της μεθόδου παραμένει `turn_wheel()` και μπορούμε να έχουμε όσες παραλλαγές θέλουμε όσον αφορά το είδος και το πλήθος των παραμέτρων που δέχεται. Παρ' όλ' αυτά, ο τύπος δεδομένων που επιστρέφει η μέθοδος (ή `void` αν δεν επιστρέφει κάτι) θα πρέπει να είναι ο ίδιος σε κάθε παραλλαγή της μεθόδου.

5.Επιπλέον Τύποι Δεδομένων

Πίνακες

Αν και έχουμε ήδη αναφέρει τους σημαντικότερους τύπους δεδομένων στη Java σε προηγούμενη παράγραφο, αφήσαμε σκόπιμα την αναφορά στους πίνακες. Ένας λόγος είναι ότι σε αντίθεση με τη C/C++, οι πίνακες στη Java υλοποιούνται ως αντικείμενα. Αυτό έχει πολλά οφέλη, όπως π.χ. το ότι δεν χρειάζεται να αποδεσμεύουμε τη μνήμη που χρησιμοποιεί ένας πίνακας (την αποδέσμευση την αναλαμβάνει το σύστημα περισυλλογής σκουπιδιών της Java) , ή το ότι είναι μεταβλητού μεγέθους από κατασκευής (κάτι που δεν ισχύει για τη C/C++).

Οι πίνακες γενικά χρησιμοποιούνται για την οργάνωση και καταχώρηση όμοιων αντικειμένων. Μπορούν να είναι μίας ή πολλών διαστάσεων και ο τρόπος προσπέλασης των στοιχείων είναι παρόμοιος με αυτόν της C/C++.

Μπορούμε να ορίσουμε ένα μονοδιάστατο πίνακα πολύ απλά ως εξής:

```
type table[];
```

Ενώ αν θέλουμε να τον δημιουργήσουμε κιόλας, χρησιμοποιούμε την new:

```
table = new type[size];
```

Ή μπορούμε να πραγματοποιήσουμε και τα δύο στάδια με μία εντολή:

```
type table[] = new type[size];
```

Ως `type` θεωρούμε τον τύπο δεδομένων των αντικειμένων του πίνακα και μπορεί να είναι είτε ένας από τους απλούς τύπους (`bool`, `byte`, `short`, `int`, `long`, `float`, `double`, `char`, `String`) είτε το όνομα μιας κλάσης αντικειμένων (της Java ή δική μας). Το `size` απεικονίζει το μέγεθος του πίνακα `table`.

Όπως και στη C/C++, μπορούμε να προσπελλάσουμε τα στοιχεία του πίνακα με την σύνταξη `table[i]`, όπου `i` η θέση του στοιχείου που μας ενδιαφέρει. Εδώ πρέπει να σημειωθεί ότι σε αντιστοιχία με τη C/C++ η Java πραγματοποιεί την αρίθμηση των πινάκων από το μηδέν (`0`) ως το `size-1`. Δηλαδή αν έχουμε έναν πίνακα `A` με 10 στοιχεία το πρώτο στοιχείο είναι το `A[0]` και το τελευταίο το `A[9]`. Ακολουθεί ένα παράδειγμα για καλύτερη κατανόηση:

```
int data[] = new int[10];
int i;
System.out.println("Size of array data: " + data.length);
for (i = 0; i < data.length; i++) {
   data[i] = i*i;
   System.out.println("data[" + i + "] = " + data[i]);
}
```

Το αποτέλεσμα του κώδικα αυτού θα είναι:

```
Size of array data: 10
data[0] = 0
data[1] = 1
data[2] = 4
data[3] = 9
data[4] = 16
data[5] = 25
```

```
data[6] = 36
data[7] = 49
data[8] = 64
data[9] = 81
```

Στο συγκεκριμένο παράδειγμα χρησιμοποιήσαμε τη μέθοδο `length` που επιστρέφει το τρέχον μέγεθος του πίνακα.

Σε περίπτωση που θέλουμε να αρχικοποιήσουμε ένα πίνακα, δηλαδή να ορίσουμε αρχικές τιμές για τα στοιχεία του, αυτό μπορούμε να το πετύχουμε με την εξής συντακτική δομή:

```
int dataset[] = { 22, 3, 54, 43, 199, 20, 20, 67, 7, 80 };
```

Αξίζει να σημειωθεί ότι δεν είναι απαραίτητη η χρήση της `new` κατά την δήλωση του πίνακα `dataset`. Εννοείται η χρήση της στη δημιουργία του αντικειμένου του δεξιού μέρους της ισότητας.

Πολυδιάστατοι πίνακες

Δε θα ήταν πλήρης η υποστήριξη των πινάκων σε μια γλώσσα προγραμματισμού όπως η Java, αν αυτή δεν υποστήριζε πολυδιάστατους πίνακες.

Για παράδειγμα, ένας δισδιάστατος πίνακας μπορεί να δηλωθεί ως εξής:

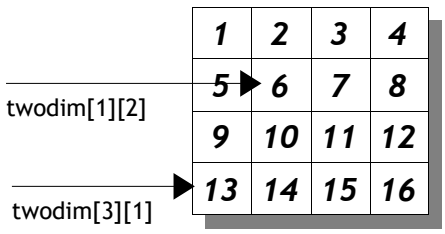
int twodim[][] = new int[4][4];

```
int i, j, counter = 1;
for (i = 0; i < twodim.length; i++) {
   for (j = 0; j < twodim[i].length; j++) {
      twodim[i][j] = counter;
      counter++;
      System.out.print(twodim[i][j] + " ");
   }
   System.out.println();
}
```

Το αποτέλεσμα αυτού του κώδικα θα είναι:

```
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
```

Δημιουργήσαμε έτσι έναν δισδιάστατο πίνακα, του οποίου τα στοιχεία τα προσπελλάζουμε όπως δείχνει το σχήμα:



Περισσότερα για τα Strings

Τα strings τα έχουμε ήδη αναφέρει και τα χρησιμοποιήσαμε μερικές φορές στην εντολή `System.out.println()`. Αντίθετα με τα strings σε άλλες γλώσσες προγραμματισμού που είναι απλώς πίνακες χαρακτήρων, στη Java τα strings είναι κανονικά αντικείμενα, που υλοποιούνται με την κλάση `String`.

Πέρα από την απευθείας χρήση τους που είδαμε στην `println()`, μπορούμε να δημιουργήσουμε αντικείμενα `String`, με τον ίδιο τρόπο όπως και με κάθε άλλο αντικείμενο, δηλαδή με τη χρήση της `new`.

```
String str = new String("Hello");
System.out.println(str);
```

Ειδικά για τα `String` υπάρχει και ένας πιο σύντομος τρόπος δημιουργίας τους:

```
String str2 = "Another string";
System.out.println(str2);
```

Η κλάση `String` παρέχει ορισμένες μεθόδους, οι οποίες είναι αρκετά χρήσιμες για επεξεργασία του κειμένου μέσα στο `String`. Παραθέτουμε τις σημαντικότερες από αυτές στον ακόλουθο πίνακα:

Όνομα	*Λειτουργία*
`boolean equals(String str)`	Επιστρέφει `true` αν τα δύο `String` αντικείμενα έχουν το ίδιο περιεχόμενο.
`int length()`	Επιστρέφει το μήκος (σε χαρακτήρες) του `String`.
`char charAt(int index)`	Επιστρέφει τον χαρακτήρα που βρίσκεται στη θέση `index` του `String`.
`int compareTo(String str)`	Συγκρίνει δύο αντικείμενα `String`. Αν το καλόν αντικείμενο (δηλ. αυτό που καλεί την `compareTo()`) είναι μικρότερο από το `str`, τότε επιστρέφει αρνητικό αποτέλεσμα, μηδέν αν έχουν το ίδιο περιεχόμενο, ή θετικό αποτέλεσμα αν το καλόν `String` είναι μεγαλύτερο από το `str`.
`int indexOf(String str)`	Αναζητά το `str` μέσα στο καλόν αντικείμενο `String`. Αν βρεθεί επιστρέφει τη θέση της πρώτης εμφάνισής του, αλλιώς `-1`.
`int lastIndexOf(String str)`	Αναζητά το `str` μέσα στο καλόν αντικείμενο `String`. Αν βρεθεί επιστρέφει τη θέση της τελευταίας εμφάνισής του, αλλιώς `-1`.

Ακολουθεί ένα παράδειγμα χρήσης των strings στη Java:

```
String str1 = "Hello there, from Java!";
String str2 = "One two three four";
String str3 = "Java strings are cool!";
String str4 = new String(str3);
int index, result;

System.out.println("str1 is " + str1.length() + " characters long.");
```

```
for (int i=0; i < str2.length(); i++)
   System.out.print(str2.charAt(i) + "|");
System.out.println();

if (str3.equals(str4))
   System.out.println("str3 == str4");
else
   System.out.println("str3 != str4");

if (str3.equals(str2))
   System.out.println("str3 == str2");
else
   System.out.println("str3 != str2");

result = str3.compareTo(str1);
if (result < 0)
   System.out.println("str3 < str1");
else if (result == 0)
   System.out.println("str3 == str1");
else
   System.out.println("str3 > str1");

index = str1.indexOf("Java");
if (index)
   System.out.println("'Java' exists in str1 in position " + index);
else
   System.out.println("'Java' does not exist in str1");

index = str2.indexOf("Java");
if (index)
   System.out.println("'Java' exists in str2 in position " + index);
else
   System.out.println("'Java' does not exist in str2");

index = str3.indexOf("Java");
if (index)
   System.out.println("'Java' exists in str3 in position " + index);
else
   System.out.println("'Java' does not exist in str3");
```

Το αποτέλεσμα του κώδικα αυτού θα είναι:

```
H|e|l|l|o| |t|h|e|r|e|,| |f|r|o|m| |J|a|v|a|!|
str3 == str4
str3 != str2
str3 > str1
'Java' exists in str1 in position 18
'Java' does not exist in str2
'Java' exists in str3 in position 1
```

6.Κατασκευάζοντας ένα πρόγραμμα

Στο σημείο αυτό έχουμε ορίσει τα περισσότερα από τα βασικά εργαλεία για να γράψουμε ένα πλήρες πρόγραμμα σε Java. Δεν έχουμε αναφέρει όμως τη δομή ενός προγράμματος, τον τύπο αρχείων που θα χρησιμοποιηθούν ή πως θα τα μεταγλωττίσουμε και εκτελέσουμε.

Η ρουτίνα main()

Στη Java, όπως ακριβώς και στις γλώσσες στις οποίες βασίστηκε, τις C/C++, κάθε πρόγραμμα ξεκινά την εκτέλεσή του από την ρουτίνα/μέθοδο main(). Αντίθετα με τις παραπάνω γλώσσες όμως, η main() δεν βρίσκεται σε κάποιο αρχείο μόνη της, αλλά ως μέθοδος σε κάποια κλάση. Ακολουθεί ένα παράδειγμα προγράμματος σε Java:

```
class Example {
   public static void main(String args[]) {
      System.out.println("hello everyone");
   }
}
```

Οι λέξεις `public`, `static` και `void` έχουν ειδική σημασία, καθώς επίσης και η παράμετρος `args[]`.

Όσον αφορά τη `void` έχουμε ήδη αναφέρει ότι χρησιμοποιείται όταν η μέθοδος δεν επιστρέφει κάποιο αποτέλεσμα. Στην περίπτωση της `main()` σημαίνει ότι το πρόγραμμα δεν επιστρέφει κάποιο κωδικό επιτυχίας στο λειτουργικό σύστημα.

Η `public` αφορά την πρόσβαση σε κάποια μεταβλητή ή μέθοδο μιας κλάσης και σημαίνει ότι το στοιχείο που χαρακτηρίζει είναι προσβάσιμο από οποιοδήποτε άλλο αντικείμενο, ακόμη και εκτός της ίδιας της κλάσης. Για τη `main()` είναι απαραίτητος χαρακτηρισμός γιατί εκτελείται από κώδικα εκτός της ίδιας της κλάσης (το λειτουργικό σύστημα).

Τέλος, η `static` χαρακτηρίζει δεδομένα ή μεθόδους τα οποία θέλουμε να καλέσουμε πριν αυτά δημιουργηθούν σε κάποιο αντικείμενο. Διαφορετικά, θα ήταν αναγκαία η δημιουργία ενός αντικειμένου της κλάσης `Example` με τη `new`, για να μπορέσουμε να καλέσουμε τη `main()`. Αλλά κάτι τέτοιο δεν είναι δυνατό, γιατί θα έπρεπε να εκτελέσουμε κώδικα πρίν εκτελεστεί η `main()`, αλλά η `main()` είναι αυτή που εκτελείται πρώτη σε ένα πρόγραμμα Java.

Αρχεία .java

Ο πηγαίος κώδικας ενός προγράμματος (source code) ή μονάδα μεταγλώττισης (compilation unit) όπως καλείται στη Java, περιέχεται σε αρχεία των οποίων η κατάληξη είναι `.java`. Το όνομα του αρχείου πρίν την κατάληξη *πρέπει* να είναι το ίδιο με το όνομα της κλάσης την οποία περιέχει. Σε άλλες γλώσσες προγραμματισμού, κάτι τέτοιο δεν είναι αναγκαίο, αλλά για την Java είναι απαραίτητο για επιτυχημένη μεταγλώττιση του πηγαίου κώδικα.

Η μεταγλώττιση γίνεται χρησιμοποιώντας την εντολή `javac` (java compiler), ως εξής:

```
(source code directory)# javac Example.java          (UNIX/Linux)

C:\(source code directory)\> javac Example.java      (Windows)
```

Η εντολή αυτή θα παράγει τον εκτελέσιμο κώδικα της Java (το java bytecode, όπως αναφέραμε στην αρχή του παρόντος κειμένου), υπό τη μορφή αρχείου με το ίδιο όνομα

αλλά με διαφορετική κατάληξη. Η κατάληξη του java bytecode είναι .class.

Αρχεία .class

Τα αρχεία .class είναι αυτά που εκτελούνται μέσω του Java Virtual Machine (JVM). Η κλήση του JVM δε γίνεται άμεσα, αλλά μέσω της εντολής java.

```
(source code directory)# java Example                    (UNIX/Linux)
hello everyone

C:\(source code directory)\> java Example                (Windows)
hello everyone
```

Σημειώστε ότι δε προσθέτουμε την κατάληξη .class στο όνομα του αρχείου Example.

Για κάθε αρχείο .java, ο compiler της java δημιουργεί ένα αντίστοιχο αρχείο .class. Για την εκτέλεση ενός προγράμματος απαιτούνται όλα τα αρχεία .class που αντιστοιχούν στα αρχεία .java που χρησιμοποιεί το πρόγραμμα.

Αρχεία .jar

Στην περίπτωση που χρησιμοποιούμε πολλές κλάσεις σε ένα πρόγραμμα, δηλαδή πολλά αρχεία .java και κατά συνέπεια πολλά αρχεία .class, για να εκτελέσουμε ένα αρχείο σε ένα άλλο υπολογιστή, θα έπρεπε να μεταφέρουμε όλα τα αρχεία .class και να τα εκτελέσουμε επί τόπου. Κάτι τέτοιο δεν είναι ιδιαίτερα βολικό ειδικά για μεγάλο αριθμό αρχείων.

Για το σκοπό αυτό, η Java παρέχει ένα σύστημα ομαδοποίησης των αρχείων .class σε ένα αρχείο JAR (Java Archive) με αντίστοιχη κατάληξη .jar. Η δημιουργία ενός τέτοιου αρχείου είναι αρκετά απλή και χρησιμοποιεί την εντολή jar (με τον ίδιο τροπο σε Windows και UNIX/Linux):

```
C:\(source code directory)# jar cvf classes.jar MyClass1.class
Another.class
```

Η εκτέλεση του κώδικα που βρίσκεται στο αρχείο classes.jar γίνεται με τον εξίσου απλό τρόπο:

```
C:\(source code directory)\> java classes.jar
```

Στην πραγματικότητα τα αρχεία jar δεν είναι παρά συμπιεσμένα αρχεία zip με άλλη κατάληξη και επιπλέον πληροφορίες.

Οι παράμετροι args[] της main()

Η μέθοδος main() δέχεται κάποια παραμέτρο args[], που είναι πίνακας από αντικείμενα String. Αυτή έχει την ίδια λειτουργία όπως και οι παράμετροι argc και argv[] στη C/C++. Δηλαδή, επιτρέπει στο πρόγραμμά μας να παραμετροποιεί την δράση του με βάση κάποιες επιλογές του χρήστη στη γραμμή εντολών (DOS/Command Prompt στα Windows, shell στο Linux).

Ακολουθεί ένα απλό παράδειγμα που απλώς τυπώνει τις παραμέτρους που περνάμε στο πρόγραμμα:

```
class ArgsExample {
   public static void main(String args[]) {
      for (int i=0; i < args.length; i++)
         System.out.println(args[i]);
   }
}
```

Αυτό το αποθηκεύουμε ως `ArgsExample.java` και πολύ απλά το μεταγλωττίζουμε με την εντολή `javac` όπως έχουμε αναφέρει παραπάνω:

```
# javac ArgsExample.java
```

και το εκτελούμε δίνοντας του και κάποιες παραμέτρους:

```
# java ArgsExample hello mate "what's up?"
hello
mate
what's up?
```

Το πρόγραμμα δέχτηκε τρεις παραμέτρους (`hello`, `mate` και `"what's up?"`) τις οποίες τυπώνει μία σε κάθε γραμμή.

Εδώ σημειώνουμε ότι οι παράμετροι γενικά χωρίζονται με κενούς χαρακτήρες (space, tab) εκτός αν περικλείονται ανάμεσα σε εισαγωγικά "", οπότε θεωρούνται και οι κενοί χαρακτήρες μέρος της παραμέτρου (όπως και με τη παράμετρο `"what's up?"` πιο πάνω).

7.Προχωρημένα θέματα

Ροή ενός προγράμματος

Μέχρι τώρα παρουσιάσαμε ορισμένα παραδείγματα προγραμμάτων σε Java αλλά δεν αναλύσαμε περισσότερο τη λειτουργία τους και πολύ περισσότερο τη ροή τους. Η ροή ενός προγράμματος αναφέρεται στην διαδοχή της εκτέλεσης των εντολών στον επεξεργαστή (είτε είναι πραγματικός είτε εικονικός στη περίπτωση της Java με το JVM). Η κατανόηση και εμπέδωση του προγραμματισμού ως έννοια, προϋποθέτει την αναγνώριση της ροής ενός προγράμματος από τη μελέτη του πηγαίου του κώδικα.

Από τη μελέτη αυτή μπορούμε να βγάλουμε τα εξής συμπεράσματα:

- Αναγνώριση των κλάσεων που χρησιμοποιεί το πρόγραμμα και των σχέσεων μεταξύ τους.
- Πληροφορίες για τα δεδομένα και τους τύπους δεδομένων που χρησιμοποιούνται σε κάθε κλάση.
- Πληροφορίες για τις συναρτήσεις/μεθόδους που χρησιμοποιεί η κάθε κλάση.
- Πληροφορίες για τη σειρά κλήσης κάθε μεθόδου των κλάσεων.
- Πληροφορίες για τη δομή της κάθε μεθόδου και πιθανές παραμετροποιήσεις της λειτουργίας των.
- Από ποιά κλάση ξεκινάει το πρόγραμμα (δηλαδή σε ποιά κλάση περιέχεται η μέθοδος `main()`).

Απώτερος σκοπός είναι να συλλάβουμε τη γενική εικόνα της λειτουργίας του προγράμματος.

Θα θεωρήσουμε το ακόλουθο παράδειγμα για καλύτερη κατανόηση των προαναφερθέντων σημείων:

```
1  class ArgsExample {
2     public static void main(String args[]) {
3        for (int i=0; i < args.length; i++) {
4           if (args[i].equals("-file") == true) {
5              if (i+1 < args.length)
6                 System.out.println("FILE: "+args[i+1]);
7           }
8        }
9     }
10 }
```

Στις γραμμές 1-3 η διαδικασία είναι γνωστή: ορίζεται η κλάση, δηλώνεται η βασική ρουτίνα `main()` και ξεκινάει το loop της εντολής `for`. Στη γραμμή 4 χρησιμοποιούμε την μέθοδο `equals()` της κλάσης `String` για να ελέγξουμε μία-μία τις παραμέτρους `args[]` αν κάποια είναι ίση με "`-file`". Αν ισχύει κάτι τέτοιο (δηλαδή αν η `equals()` επιστρέψει `true`) τότε βεβαιωνόμαστε ότι υπάρχει επόμενη παράμετρος (το `i+1` είναι μικρότερο του μεγέθους του πίνακα, γραμμή 5) και τυπώνουμε την επόμενη παράμετρο (γραμμή 6). Ο έλεγχος συνεχίζεται για τις υπόλοιπες παραμέτρους `args[]`.

Αφού μεταγλωττίσουμε το πρόγραμμά μας με τη `javac`, έστω ότι το εκτελούμε με κάποιες παραμέτρους στη γραμμή εντολών:

```
# javac ArgsExample.java
# java ArgsExample one -file two three -file
FILE: two
```

Ο πίνακας `args[]` με την εκκίνηση του προγράμματος θα έχει τις εξής τιμές:

`args[0]`	`one`
`args[1]`	`-file`
`args[2]`	`two`
`args[3]`	`three`
`args[4]`	`-file`

Το μέγεθος του πίνακα απεικονίζεται με τη μεταβλητή `args.length` και στην περίπτωση αυτή είναι `args.length = 5`.

Στον ακόλουθο πίνακα, φαίνεται αυτή η ροή του προγράμματος αναλυτικά:

i	*args[i]*	*args[0].equals("-file")*	*i+1 < args.length*	*Print?*
0	"one"	false	-	no
1	"-file"	true	true	yes
2	"two"	false	-	no
3	"three"	false	-	no
4	"-file"	true	false	no

Το "-" στη στήλη "`i+1 < args.length`" σημαίνει ότι δεν γίνεται ο έλεγχος για να μας ενδιαφέρει το αποτέλεσμα, δηλαδή το πρόγραμμα δεν εισέρχεται στην εντολή της γραμμής 5 για όλες τις επαναλήψεις του for loop (για όλες τις τιμές του `i` δηλαδή). Αυτό γιατί η εντολή `if` στη γραμμή 5 εκτελείται μόνο όταν η εντολή `if` της εντολής 4 επιστρέψει `true`.

References

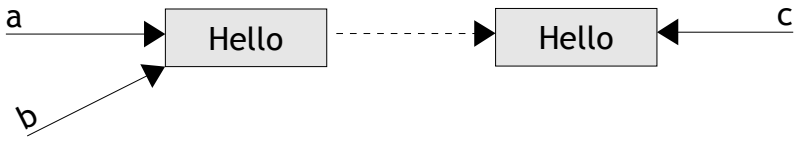
Στο προηγούμενο παράδειγμα χρησιμοποιήσαμε τη μέθοδο `equals()` για τον έλεγχο της ισότητας δύο αντικειμένων `String`. Γιατί όμως να μη χρησιμοποιήσουμε απλώς τον τελεστή ισότητας "==", όπως θα κάναμε για δύο `int` για παράδειγμα;

Στη Java, πέρα από τους απλούς τύπους δεδομένων (`bool`, `int`, `long`, `float`, κλπ), όπου έχουμε απευθείας πρόσβαση μέσω μιας μεταβλητής, για όλους τους υπόλοιπους τύπους (όλες οι κλάσσεις, συμπεριλαμβανομένης και της `String`) η μεταβλητή χρησιμοποιείται μόνο ως αναφορά στο κάθε στιγμιότυπο (instance). Δηλαδή, αν έχουμε μια μεταβλητή `str` τύπου `String`, αυτή θα περιέχει μια αναφορά σε μία θέση μνήμης όπου θα κρατείται το ίδιο το αντικείμενο. Όπως λειτουργούν δηλαδή οι pointers και τα references στη C/C++.

Για παράδειγμα, έστω δύο αντικείμενα string:

```
String a, b;
a = "Hello";
b = a;
String c = new String(a);
```

Για καλύτερη κατανόηση ακολουθεί μια σχηματική αναπαράσταση της μνήμης που καταναλώνουν τα αντικείμενα αυτά.



Οι μεταβλητές a και b είναι στην πραγματικότητα αναφορές που δείχνουν στη θέση μνήμης που περιέχει το string "Hello". Εξισώνοντας τη μεταβλητή b με την a, απλώς έχουμε δύο αναφορές στο ίδιο αντικείμενο. Συνεπώς αλλάζοντας το περιεχόμενο της b, αλλάζει ταυτόχρονα και η a. Αντίθετα δημιουργώντας ένα νέο αντικείμενο c, δεσμεύεται εκ νέου μνήμη και αντιγράφεται το αντικείμενο a στη νέα θέση η οποία έχει ως αναφορά τη μεταβλητή c. Αλλάζοντας τα περιεχόμενα της c, η a παραμένει αναλλοίωτη.

Encapsulation

Ας ορίσουμε μια κλάση Person η οποία θα περιέχει ορισμένες πληροφορίες για ένα άτομο, όπως π.χ. το όνομά του, την ηλικία του, το τηλέφωνό του και τη διεύθυνση email του. Μια τέτοια κλάση μπορεί να χρησιμοποιηθεί π.χ. σε ένα πρόγραμμα ατζέντας ή ακόμη και ως βάση σε πρόγραμμα πελατολογίου, ασθενών, κλπ.

```
class Person
{
   // οι μεταβλητές της κλάσης
   String Firstname_, Lastname_;
   int Age_;
   String Telephone_;
   String Email_;

   // ο constructor
   Person(String fname, String lname, int age, String tel,
        String email)
   {
      Firstname_ = new String(fname);
      Lastname_ = new String(lname);
      Age_ = age;
      Telephone_ = new String(tel);
      Email_ = new String(email);
   }
}
```

Ίσως παρατηρήσατε ότι τα ονόματα των μεταβλητών της κλάσης λήγουν σε "_". Φυσικά, κάτι τέτοιο δεν είναι αναγκαίο, είναι όμως μια συνηθισμένη πρακτική και βοηθάει στην αναγνώριση και διαχωρισμό των απλών μεταβλητών από τις μεταβλητές μέλη μιας κλάσης.

Αφού ορίσαμε την κλάση, μπορούμε να προχωρήσουμε στην δημιουργία κάποιων αντικειμένων της κλάσης:

```
Person bilbo = new Person( "Bilbo", "Baggins", 111, "+306970123456",
             "bilbobaggins@theshire.net");
```

Με αυτόν τον τρόπο, δημιουργήσαμε το αντικείμενο `bilbo` που αντιστοιχεί στο άτομο Bilbo Baggins, 111 ετών με τηλ. 306970123456 και email bilbobaggins@theshire.net.

Όπως είναι οι πληροφορίες που περιγράφουν το άτομο είναι προσβάσιμες σε όλους. Αυτό σημαίνει ότι αν με κάποιον τρόπο αποκτήσουμε πρόσβαση στο αντικείμενο `bilbo`, θα μπορούμε να αλλάξουμε οποιαδήποτε πληροφορία θελήσουμε και με οποιονδήποτε τρόπο. Δηλαδή, να δώσουμε μια μη έγκυρη διεύθυνση email στο πεδίο `email_` ή μια άσχετη πληροφορία στο πεδίο `telephone_`. Και μάλιστα με πολύ απλό τρόπο:

```
bilbo.Firstname_ = "μπίλμπο";
bilbo.Lastname_  = "μπαγκινσόπουλος";
bilbo.Age_       = 3;
bilbo.Email_     = "this is definitely not a valid email address";
bilbo.Telephone_ = "yeah, try to call this";
```

Πρόκειται για τρανή παραβίαση των προσωπικών δεδομένων!!!

Πώς μπορούμε να αποφύγουμε τέτοιου είδους μη προβλεπόμενη μετατροπή των δεδομένων ενός αντικειμένου;

Η Java καθώς και οι περισσότερες γλώσσες προγραμματισμού που έχουν σχεδιαστεί γύρω από το μοντέλο του αντικειμενοστραφούς προγραμματισμού, προβλέπουν τον περιορισμό της πρόσβασης των δεδομένων σε επίπεδα. Το πρώτο επίπεδο το έχουμε ήδη συναντήσει, κατά τον ορισμό της μεθόδου `main()` που ορίζεται πάντα `public` για να είναι προσβάσιμη από το λειτουργικό σύστημα.

Ακριβέστερα, η `public` ορίζει μια μέθοδο ή μια μεταβλητή ως προσβάσιμη από οπουδήποτε μέσα στην ίδια κλάση ή εκτός της κλάσης. Το αντίστροφο, δηλαδή ο περιορισμός της πρόσβασης γίνεται με τη χρήση της λέξης `private`. Η `private` περιορίζει την πρόσβαση της μεταβλητής ή της μεθόδου μόνο στην συγκεκριμένη κλάση (και φυσικά σε αντικείμενα αυτής). Οποιαδήποτε πρόσβαση εκτός, θα αποτρέπεται στο επίπεδο της μεταγλώττισης ακόμη. Για παράδειγμα η παραπάνω κλάση Person, με τη χρήση της private, θα μετασχηματιστεί ως εξής:

```
class Person
{
   // οι μεταβλητές της κλάσης
   private String Firstname_, Lastname_;
   private int Age_;
   private String Telephone_;
   private String Email_;
   ...
```

Αυτό όμως σημαίνει ότι δεν θα είναι πλέον δυνατή η πρόσβαση σε οποιαδήποτε πληροφορία του ατόμου ακόμη και για απλή ανάγνωση! Κάτι τέτοιο δεν είναι επιθυμητό και πρέπει να βρεθεί τρόπος να επιτραπεί έστω και ελεγχόμενη πρόσβαση στα δεδομένα.

Ακριβώς, αυτό επιτυγχάνουμε με την χρήση των μεθόδων της κλάσης. Ελεγχόμενη πρόσβαση για ανάγνωση αλλά και μετατροπή των δεδομένων. Συνήθως και για τις

περισσότερες περιπτώσεις κλάσεων, αρκεί ο ορισμός ενός ζέυγους μεθόδων για κάθε μεταβλητή της κλάσης, μία για ανάγνωση και μια για μετατροπή της τιμής της μεταβλητής (ένα ζεύγος getter/setter όπως λέγονται συχνά).

Για παράδειγμα, για την κλάση Person παραθέτουμε πιθανές μεθόδους get/set για ορισμένες από τις μεταβλητές (age_ και email_):

```
// Return the age
int getAge()
{
   return Age_;
}

// return the Email address
String getEmail()
{
   return new String(Email_);
}

// method to set the age of the person
bool setAge(int Age)
{
   // check if given Age is non-negative (> 0)
   if (Age > 0)
   {
      Age_ = Age;
      return true;
   } else
      return false;
}

// method to set the email address
bool setEmail(String Email)
{
   // call a helper method to check the validity of the email
   // address (if it's in the form x@y.z).
   // Ideally, the email address should be a class on its own.
   if (isValid(Email) == true)
   {
      Email_ = new String(Email);
      return true;
   } else
      return false;
}
```

Βλέπουμε πώς ελέγχεται πλέον η πρόσβαση στις μεταβλητές. Η μεταβλητή που κρατά τη διεύθυνση email του ατόμου, για παράδειγμα, αλλάζει μόνο αν έχουμε δώσει μια έγκυρη διεύθυνση email (της μορφής x@y.z).

Με τον ίδιο τρόπο που περιορίζουμε την πρόσβαση σε μεταβλητές μπορούμε να περιορίσουμε την πρόσβαση και σε μεθόδους. Θα μπορούσαμε π.χ. να έχουμε μια μέθοδο που να ελέγχει αν ο αριθμός τηλεφώνου του ατόμου είναι έγκυρος, πραγματοποιώντας αναζήτηση σε κάποια βάση δεδομένων. Οπωσδήποτε, μια τέτοια μέθοδος δεν θα θέλαμε να είναι προσβάσιμη από οποιονδήποτε εκτός της κλάσης, παρά μόνο σε άλλες μεθόδους της ίδιας της κλάσης (π.χ. στην μέθοδο `setTelephone()`).

Inheritance

Η κληρονομικότητα είναι ένα ακόμη χαρακτηριστικό του αντικειμενοστραφούς προγραμματισμού. Πρακτικά, μια κλάση κληρονομεί τα χαρακτηριστικά μιας υπάρχουσας κλάσης και προσθέτει καινούρια ή τροποποιεί τα ήδη υπάρχοντα.

Για καλύτερη κατανόηση της έννοιας, θα χρησιμοποιήσουμε το παράδειγμα της κλάσης Person της προηγούμενης ενότητας. Η κλάση Person ορίζει χαρακτηριστικά που περιγράφουν ένα άτομο αλλά δεν προβλέπει επιπλέον πληροφορίες, όπως π.χ. το φύλο, τι δουλειά κάνει το άτομο και τη διεύθυνση εργασίας του, αν έχει κινητό τηλέφωνο, αν είναι παντρεμένος/η με παιδιά κλπ. Το πρόβλημα είναι ότι δεν μπορούμε να προβλέψουμε όλες τις πιθανές πληροφορίες και να τις εισάγουμε στην κλάση Person γιατί οι περισσότερες θα έμεναν αναπάντητες και κάτι τέτοιο θα οδηγούσε σε σπατάλη χώρου (αφού θα έπρεπε να καταχωρήσουμε όλες τις πληροφορίες που θα ήταν κενές). Επιπλέον, θα ήταν εκνευριστικό και κουραστικό για το χρήστη να πρέπει να εισάγει άχρηστες πληροφορίες. Αν κάποιος δεν εργάζεται δεν έχει νοήμα να του ζητάμε το όνομα της ειδικότητάς του και διεύθυνση/τηλέφωνο της εργασίας του, σωστά;

Κάτι πολύ πιο χρήσιμο είναι να έχουμε μια κοινή βάση και να κρατάμε επιπλέον πληροφορίες μόνο όταν τις χρειαζόμαστε. Έστω ότι η κοινή βάση είναι η κλάση `Person` και θέλουμε να μελετήσουμε τις περιπτώσεις να είναι κάποιος υπάλληλος (Clerk) ή δάσκαλος (Teacher). Και οι δύο κατηγορίες ατόμων μοιράζονται κοινά χαρακτηριστικά που θεωρούμε ότι περιέχονται στην κλάση `Person`. Μπορούμε δηλαδή να ορίσουμε δύο νέες κλάσεις που θα κληρονομούν την κλάση `Person`. Η δήλωση της κληρονομικότητας μιας κλάσης γίνεται ως εξής (ορίζουμε ταυτόχρονα και την πρόσβαση στις μεταβλητές και τον δημιουργό της κλάσης):

```
class Clerk extends Person
{
   private String JobTitle_;
   private String CompanyName_;
   private String JobAddress_;
   private String JobEmail_;
   private String JobTel_;
   private String JobFax_;
   private String JobDescription_;

   public Clerk(String fname, String lname, int age, String tel,
           String email, String jobtitle, String companyname,
           String jobaddress, String jobemail,
           String jobtel, String jobfax,
           String jobdescription)
   {
      Firstname_ = new String(fname);
      Lastname_ = new String(lname);
      Age_ = age;
      Telephone_ = new String(tel);
      Email_ = new String(email);
      JobTitle_ = new String(jobtitle);
      CompanyName_ = new String(companyname);
      JobAddress_ = new String(jobaddress);
      JobEmail_ = new String(jobemail);
      JobTel_ = new String(jobtel);
      JobFax = new String(jobfax);
      JobDescription_ = new String(jobdescription);
```

```
   }
   // ακολουθούν οι μέθοδοι get/set για κάθε μεταβλητή με τους
   // απαραίτητους ελέγχους...
   ...

   // η ακόλουθη μέθοδος δίνει συνοπτικές πληροφορίες για τον
   // υπάλληλο.
   String getInfo() {
      return new String(getFirstname()+" "+getLastname()
             +" works at "+CompanyName_
             +", at "+JobAddress_
             +".\n Email: "+getEmail()+"\n"
             +"Tel: "+JobTel_);
   }
}
```

Αντίστοιχα, ορίζουμε την κλάση `Teacher`:

```
class Teacher extends Person
{
   private String Title_;
   private String School_;
   private String SchoolAddress_;
   private String SchoolTel_;
   private String CourseName_;
   private String CourseDescription_;

   public Teacher(String fname, String lname, int age, String tel,
          String email, String title, String school,
          String schooladdress, String schooltel,
          String coursename, String coursedescription)
   {
      Firstname_ = new String(fname);
      Lastname_ = new String(lname);
      Age_ = age;
      Telephone_ = new String(tel);
      Email_ = new String(email);
      Title_ = title;
      School_ = school;
      SchoolAddress_ = schooladdress;
      SchoolTel_ = jobtel;
      CourseName_ = coursename;
      CourseDescription_ = coursedescription;
   }
   // ακολουθούν οι μέθοδοι get/set για κάθε μεταβλητή με τους
   // απαραίτητους ελέγχους...
   ...

   // Η ακόλουθη μέθοδος δίνει συνοπτικές πληροφορίες για τον
   // καθηγητή.
   String getInfo() {
      return new String(getFirstname()+" "+getLastname()
             +" teaches "+CourseName_+" at "+School_
             +", "+SchoolAddress_+".\n"
             +"Email: "+getEmail()+"\n"
```

```
            +"Tel: "+SchoolTel_);
    }
}
```

Προσέξτε ότι χρησιμοποιήσαμε τις μεθόδους `get()` της κλάσης `Person` για την πρόσβαση στις μεταβλητές της (εφόσον είναι δηλωμένες `private`). Θυμίζουμε ότι αν μια μεταβλητή ή μέθοδος είναι δηλωμένη `private`, είναι προσβάσιμη *μόνο* από μεθόδους της ίδιας της κλάσης. Δεν είναι προσβάσιμη από τις μεθόδους μιας υποκλάσης αυτής.

Πώς όμως φαίνεται η χρησιμότητα αυτής της κατασκευής; Δείτε το ακόλουθο παράδειγμα κώδικα όπου χρησιμοποιούμε και τις τρεις κλάσεις:

```
Person bilbo = new Person( "Bilbo", "Baggins", 111, "+306970123456",
            "bilbobaggins@theshire.net");
Clerk sam = new Clerk( "Samwise", "Gamgee", 33, "+30697654321",
          "samgamgee@theshire.net",
          "Gardener", "Baggins Inc.",
          "Bag End, Hobbiton, The Shire",
          "gardener@baggins.com",
          "+302103456789", "+302103456780",
          "Garden Dept. Director");
Teacher pippin = new Teacher( "Peregrin", "Took", 27, "+30690090090",
          "pippin@theshire.net", "Dr.",
          "King's College", "Hobbiton",
          "+30210000001", "Philosophy",
          "Deal with the important matters of life, eg. what do we
eat?");
```

Μπορούμε πολύ εύκολα να χρησιμοποιήσουμε για κάθε ένα από αυτά τα αντικείμενα, πέρα από τις μεθόδους της κλάσης στην οποία ανήκει, και τις μεθόδους της γονικής κλάσης (δηλαδή την κλάση της οποίας τα χαρακτηριστικά κληρονομεί):

```
System.out.println("bilbo has email address: " + bilbo.getEmail());
```

αυτή η εντολή θα τυπώσει:

```
bilbo has email address: bilbobaggins@shire.net
```

Ενώ η εντολή:

```
System.out.println("sam works as a "+ sam.getJobTitle() + " at " +
                    sam.getCompanyName());
```

θα τυπώσει:

```
sam works as a Gardener at Baggins Inc.
```

Παράλληλα, η εντολή:

```
System.out.println("pippin teaches "+ pippin.getCourseName() + " at " +
                    pippin.getSchool());
```

θα τυπώσει:

```
pippin teaches Philosophy at King's College
```

Τέλος, οι εντολές:

```
System.out.println("sam's private telephone is " + sam.getTel());
System.out.println("pippin is " + pippin.getAge() + " years old");
```

θα τυπώσουν:

```
sam's private telephone is +30697654321
pippin is 27 years old
```

Καλέσαμε δηλαδή μεθόδους της κλάσης Person και από τα τρία αντικείμενα!!! Αυτή είναι η ουσία της κληρονομικότητας των κλάσεων! Κάτι τέτοιο μας επιτρέπει να επαναχρησιμοποιούμε κώδικα που έχουμε γράψει παλαιότερα, απλώς επεκτείνοντάς τον όπως μας βολεύει κάθε φορά (*code reusability*). Έχοντας δηλαδή μερικές κλάσεις με ορισμένα μόνο τα βασικά χαρακτηριστικά, μπορούμε αναλόγως το πρόγραμμα που πρέπει να υλοποιήσουμε να προσθέσουμε ή να τροποποιήσουμε χαρακτηριστικά κατά βούληση, ώστε να επιτύχουμε το επιθυμητό αποτέλεσμα.

Polymorphism

Τί γίνεται όμως αν θέλουμε να αλλάξουμε τη λειτουργία μιας μεθόδου στην νέα κλάση που υπάρχει και στην παλιά; Στο προηγούμενο παράδειγμα, η νέες κλάσεις ορίζουν μια μέθοδο την `getInfo()`, η οποία επιστρέφει πληροφορίες για τον υπάλληλο ή τον δάσκαλο αντίστοιχα. Όπως είναι δηλωμένη, η κάθε κλάση προσφέρει τη δική της εκδοχή αλλά η αρχική `Person`, δεν έχει ορισμένη μια τέτοια μέθοδο `getInfo()`.

Αν η `getInfo()` οριστεί και για την κλάση `Person`, π.χ. ως εξής:

```
String getInfo() {
   return new String(getFirstname()+" "+getLastname()
           +"is "+getAge()+" years old");
}
```

τότε δημιουργείται το εξής ερώτημα: ποια εκδοχή της `getInfo()` θα καλείται για κάθε αντικείμενο;

Η απάντηση είναι ίσως εμφανής. Θα καλείται η `getInfo()` της κλάσης που ανήκει το αντικείμενο από το οποίο καλούμε την μέθοδο.

Για παράδειγμα, ο ακόλουθος κώδικας:

```
System.out.println(bilbo.getInfo());
System.out.println(sam.getInfo());
System.out.println(pippin.getInfo());
```

θα παράγει

```
Bilbo Baggins is 111 years old
Samwise Gamgee works at Baggins Inc., at Bag End, Hobbiton, The Shire.
Email: gardener@baggins.com
Tel: +302103456789
Peregrin Took teaches Philosophy at King's College, Hobbiton.
Email: pippin@theshire.net
Tel: +30210000001
```

Τι ακριβώς έγινε εδώ; Καλέσαμε την ίδια μέθοδο και για τα τρία αντικείμενα (την `getInfo()`) η οποία όμως ορίζεται και στις τρεις κλάσεις. Η γονική κλάση `Person` ορίζει την `getInfo()` με έναν απλό τρόπο ("`Bilbo Baggins is 111 years old`") και η ίδια

θα χρησιμοποιούνταν στις κλάσεις `Clerk` και `Teacher` αν δεν ορίζονταν και εκεί. Δηλαδή, αν δεν ορίζαμε την `getInfo()` στην κλάση `Clerk`, η `getInfo()` για το αντικείμενο `sam` θα επέστρεφε "`Samwise Gamgee is 33 year old`". Επειδή όμως η κλάση `Clerk` περιέχει περισσότερες πληροφορίες που πρέπει να απεικονιστούν με την `getInfo()`, η τελευταία επαναορίστηκε, αποκτώντας νέα λειτουργία.

Η τεχνική αυτή καλείται *Method Overriding* και είναι το θεμελιώδες χαρακτηριστικό του πολυμορφισμού.

Ο πολυμορφισμός μας δίνει τη δυνατότητα δημιουργίας αφηρημένων εννοιών/κλάσεων/μεθόδων στις οποίες θα κολλάμε κάθε φορά τη σωστή υλοποίηση αναλόγως τις ανάγκες μας.

Το ακόλουθο παράδειγμα έχει σκοπό να δείξει πώς ακριβώς χρησιμοποιούμε τον πολυμορφισμό. Θεωρούμε τα αντικείμενα που έχουμε ορίσει πριν (`bilbo`, `sam`, `pippin`).

```
Person who[3];
who[0] = bilbo;
who[1] = sam;
who[2] = pippin;
for (int i=0; i < who.length; i++)
   System.out.println(who[i].getInfo());
```

Το παραπάνω παράδειγμα θα τυπώσει ό,τι και το προηγούμενο μόνο που τώρα χρησιμοποιήσαμε έναν πίνακα αντικειμένων `Person` για να οργανώσουμε την πληροφορία. Αλλά το αντικείμενο `sam` είναι τύπου `Clerk` και το `pippin` ανήκει στην κλάση `Teacher`. Πώς τα καταχωρήσαμε στον πίνακα `who`; Η απάντηση βρίσκεται στην σχέση που έχουν οι κλάσεις `Person`, `Clerk` και `Teacher` μεταξύ τους. Ένα αντικείμενο `Clerk` είναι ταυτόχρονα και αντικείμενο `Person`, όπως και ένα αντικείμενο `Teacher` είναι επίσης `Person`. Ένας `Clerk` δεν είναι όμως ταυτόχρονα `Teacher`. Έτσι μπορούμε να τα αντιμετωπίζουμε κατά βούληση ως `Clerk` και `Teacher` ή `Person`, αναλόγως τις ανάγκες μας. Ακόμη όμως και αν τα προσπελάσουμε ως `Person`, η `getInfo()` του καθενός θα είναι αυτή που ορίζει η κλάση του.

Τί γίνεται όμως αν χρειάζεται να καλέσουμε και την γονική μέθοδο (δηλαδή αυτή που επαναορίζουμε στη νέα κλάση); Στην περίπτωση αυτή χρησιμοποιούμε την αναφορά στη γονική κλάση με τη λέξη `super` (εννοώντας την κλάση "πάνω" από την τρέχουσα στο δέντρο της ιεραρχίας της κληρονομικότητας). Η χρήση της `super` ισχύει και κατά τη δημιουργία ενός αντικειμένου στο δημιουργό (όπου καλούμε τον constructor της γονικής κλάσης). Ένα παράδειγμα είναι αναγκαίο για καλύτερη κατανόηση. Μετασχηματίζουμε το δημιουργό και τη μέθοδο `getInfo()` της κλάσης `Teacher` στα εξής:

```
public Teacher(String fname, String lname, int age, String tel,
        String email, String title, String school,
        String schooladdress, String schooltel,
        String coursename, String coursedescription)
{
   super(fname, lname, age, tel, email);
   Title_ = title;
   School_ = school;
   SchoolAddress_ = schooladdress;
   SchoolTel_ = jobtel;
   CourseName_ = coursename;
   CourseDescription_ = coursedescription;
}
```

```
   String getInfo() {
      return new String(super.getInfo()
              +" and teaches "+CourseName_+" at "+School_
              +", "+SchoolAddress_+".\n"
              +"Email: "+getEmail()+"\n"
              +"Tel: "+SchoolTel_);
   }
```

Με αυτόν τον τρόπο στον δημιουργό αρκεί πλέον να αρχικοποιούμε (initialize) μόνο τα χαρακτηριστικά που είναι νέα στην κλάση χωρίς να κάνουμε διπλό κόπο που έχει ήδη γίνει στην γονική κλάση. Επίσης, στην `getInfo()`, χρησιμοποιούμε την πληροφορία που επιστρέφεται από τη γονική κλάση και συμπληρώνουμε με τα νέα στοιχεία. Αν π.χ. αλλάζαμε το κείμενο που επιστρέφει η `getInfo()` της κλάσης `Person`, θα άλλαζε αυτόματα και το κείμενο που επέστρεφε η `getInfo()` της κλάσης `Teacher`, εφόσον χρησιμοποιούμε την αναφορά στη κλάση `Person` με τη `super`.

Σημειώνουμε ότι method overriding έχουμε μόνο όταν η δήλωση της μεθόδου είναι η ίδια στη γονική και στην θυγατρική κλάση. Δηλαδή ίδιο όνομα, επιστρεφόμενα δεδομένα και παραμέτρους. Αν ένα από αυτά είναι διαφορετικά (εκτός από το όνομα φυσικά) τότε έχουμε υπερφόρτωση μεθόδου (method overloading) που το έχουμε ήδη αναλύσει σε προηγούμενη ενότητα.

Abstract Classes

Μερικές φορές μπορεί να θέλουμε να ορίσουμε απλώς μια γενική δομή κάποιων κλάσεων αλλά χωρίς να θέλουμε όμως να προσφέρουμε μια ακριβή υλοποίηση. Δηλαδή να ορίσουμε μια "αφηρημένη" κλάση, που να παρέχει απλώς ένα πλαίσιο που θα συμπληρώνουν οι θυγατρικές κλάσεις μέσω της τεχνικής method overriding. Αυτό σημαίνει ότι στον ορισμό της κλάσης απλώς θα δηλώνονται ορισμένες μέθοδοι ως "αφηρημένες" αλλά δεν θα παρέχεται κάποιος ορισμός γι' αυτές.

Ένα κλασσικό παράδειγμα αφηρημένης κλάσης είναι η κλάση του αυτοκινήτου. Ο μηχανισμός που αλλάζει ταχύτητες σε κάθε αυτοκίνητο είναι διαφορετικός και διαφέρει ανάμεσα στους κατασκευαστές και στα διάφορα μοντέλα. Όμως ο τρόπος αλληλεπίδρασης (δηλαδή μέσω του μοχλού ταχυτήτων) είναι καλά ορισμένος έστω και με κάποια περιθώρια παραλλαγών. Ο τρόπος ορισμού της κλάσης γίνεται ως εξής:

```
abstract class Car {
   ...
   // Αλλαγή της ταχύτητας. Επιστρέφει true αν η αλλαγή ήταν επιτυχής
   // ή false αν ήταν ανεπιτυχής (π.χ. από 5 σε όπισθεν).
   abstract bool change_gear(int new_gear);
   ...
}

class FerrariModena extends Car {
   ...
   bool change_gear(int new_gear) {
      // Η συγκεκριμένη υλοποίηση βρίσκεται εδώ
   }
   ...
}
```

Final Classes

Όσο χρήσιμη και αν είναι η δυνατότητα του method overriding, μερικές φορές δεν είναι επιθυμητή ή ακόμη χειρότερα, μπορεί να αποβεί καταστροφική. Για παράδειγμα αν μια κλάση περιέχει πληροφορίες που αφορούν πρόσβαση σε ευαίσθητα δεδομένα (π.χ. κωδικούς πρόσβασης, αριθμούς καρτών/λογαριασμών, προσωπικές βάσεις δεδομένων), τότε θα ήταν δυνατή η πρόσβαση σε αυτά τα δεδομένα απλώς με την δημιουργία μιας νέας θυγατρικής κλάσης. Η νέα κλάση θα είχε πρόσβαση στα δεδομένα αυτά και θα μπορούσε να ορίσει νέες μεθόδους που να τα χειρίζονται (τα δεδομένα) με μη προκαθορισμένους ή ελεγχόμενους τρόπους. Είναι προφανές ότι κάτι τέτοιο δεν θα πρέπει να επιτρέπεται.

Αυτή η απαγόρευση επιτυγχάνεται με την δήλωση της κλάσης ή της μεθόδου με τη λέξη `final`. Η δήλωση μιας μεθόδου ως `final` απαγορεύει το overriding της συγκεκριμένης μεθόδου από οποιαδήποτε θυγατρική κλάση. Η δήλωση όμως μιας κλάσης ως `final`, απαγορεύει εξ' ολοκλήρου την διαδικασία της κληρονόμησης της κλάσης από κάποια άλλη.

Για παράδειγμα:

```
class A {
   final int amethod() {
      // sample code
   }
}
class B extends A {
   int amethod() {          // ΛΑΘΟΣ!!! Δε λειτουργεί!
      // different code
   }
}
```

ενώ αν ορίσουμε ολόκληρη την κλάση ως `final`:

```
final class A {
   // class internals
}

class B extends A {   // ΛΑΘΟΣ!!! Δε λειτουργεί!
   // class internals
}
```

Η χρήση της λέξης `final` για μεταβλητές έχει την έννοια της σταθεράς, δηλαδή ότι κάποια μεταβλητή έχει προκαθορισμένη τιμή και δεν είναι δυνατή η αλλαγή αυτής της τιμής (όπως με τη χρήση της λέξης `const` στη C++).

```
class myfinal {
   final int x = 1;         // Ο int x θα έχει την τιμή 1 για κάθε
                // αντικείμενο
}
```

Η κλάση Object

Στη Java ορίζεται εξ' ορισμού μια βασική κλάση πάνω στην οποία βασίζονται όλες οι υπόλοιπες. Η κλάση αυτή προσφέρει ορισμένες θεμελιώδεις λειτουργίες που κληρονομούνται σε όλες τις υπόλοιπες κλάσεις, ακόμη και σε αυτές που κατασκευάζουμε

μόνοι μας. Η κλάση αυτή ονομάζεται Object και στον πίνακα που ακολουθεί θα αναφέρουμε ορισμένες από τις βασικές μεθόδους της κλάσης:

Όνομα	*Λειτουργία*
`boolean equals(Object ob)`	Επιστρέφει `true` αν τα δύο `Object` αντικείμενα έχουν το ίδιο περιεχόμενο.
`Object clone()`	Δημιουργεί ένα νέο αντικείμενο, ακριβώς ίδιο με το αντικείμενο που καλεί την `clone()`.
`Class getClass()`	Επιστρέφει το όνομα της κλάσης του αντικειμένου που καλεί την `getClass()`.
`String toString()`	Επιστρέφει μια αναπαράσταση/περιγραφή του αντικειμένου σε `String`. Η μέθοδος αυτή χρησιμοποιείται αυτόματα από την εντολή `System.out.println()`, όταν τυπώνουμε ένα αντικείμενο. Η μέθοδος αυτή μπορεί να γίνει overload.

8.Exceptions

Όποιον προγραμματιστή και να ρωτήσετε (έμπειρο κατά προτίμηση!) θα σας πεί ότι δεν υπάρχει πραγματικά πρόγραμμα χωρίς προβλήματα. Τα προβλήματα μπορεί να είναι λόγω ύπαρξης bugs (δηλαδή μη εσκεμμένων λαθών στον κώδικα), λόγω εξωτερικών παραγόντων (μη διαθέσιμος χώρος στο σκληρό, κατεστραμμένα αρχεία, ανεπαρκής μνήμη, κλπ) ή ακόμη και από λάθη του χρήστη (π.χ. επέλεξε ορισμένες ασύμβατες ενέργειες με αποτέλεσμα το πρόγραμμα να βρεθεί σε κατάσταση αδιεξόδου και να τερματίσει).

Κατά το σχεδιασμό και την υλοποίηση ενός προγράμματος θα πρέπει ο προγραμματιστής να προσπαθήσει να προβλέψει όσο το δυνατόν περισσότερες πιθανές καταστάσεις που θα οδηγούσαν το πρόγραμμά του σε σφάλμα ή/και τερματισμό (crash). Φυσικά δεν είναι δυνατή η πρόβλεψη όλων αυτών των πιθανών καταστάσεων αλλά υπάρχουν ορισμένες περιπτώσεις που είναι αρκετά συνηθισμένες και πρέπει θα πρέπει να ληφθούν υπόψιν κατά τον σχεδιασμό του προγράμματος. Φυσικά, οι καταστάσεις αυτές ποικίλουν αναλόγως την φύση και τη λειτουργία του εκάστοτε προγράμματος.

Οι παλαιότερες γλώσσες προγραμματισμού (C, PASCAL) δεν παρέχουν ξεχωριστούς μηχανισμούς για τον έλεγχο των σφαλμάτων σε ένα πρόγραμμα. Ο προγραμματιστής πρέπει να ελέγχει την σωστή εκτέλεση κάθε γραμμής εντολής στο πρόγραμμά του ή τουλάχιστον αυτές που έχουν πιθανότητες να παρουσιάσουν πρόβλημα. Για παράδειγμα, αν θέλαμε να ανοίξουμε ένα αρχείο στο δίσκο, να διαβάσουμε κάποιες πληροφορίες από αυτό και να γράψουμε νέα δεδομένα στο ίδιο αρχείο, θα έπρεπε να εκτελέσουμε κώδικα που θα έχει την εξής μορφή:

```
fc = open("αρχείο.dat");
if (fc == 0 && errno != 0)
   println("error");
success = read(fc, buffer);
if (success) {
   // do stuff
}
success = write(fc, buffer);
if (success) {
   // again, do stuff
}
```

Όπως βλέπουμε, σε κάθε εντολή πραγματοποιείται έλεγχος για την σωστή εκτέλεσή της. Αν για παράδειγμα δεν έχουμε πρόσβαση στο αρχείο, η πρώτη εντολή θα επιστρέψει λάθος. Ακόμη όμως και αν εκτελεστεί επιτυχώς, θα πρέπει να επιβεβαιώσουμε και το αποτέλεσμα των υπολοίπων εντολών. Για παράδειγμα, θα μπορούσε ο χρήστης να διαγράψει το αρχείο ή να σβήσει ή να αντικαταστήσει τα περιεχόμενά του τη στιγμή που το πρόγραμμά μας το διαβάζει (ειδικά αν δεν έχουμε αποκλειστική πρόσβαση στο αρχείο).

Σε μικρά προγράμματα, τέτοιοι επιπλέον έλεγχοι δεν είναι ιδιαίτερα δύσκολοι, αλλά είναι ιδιαίτερα κουραστική διαδικασία η εισαγωγή ελέγχων σε κάθε εντολή σε ένα πρόγραμμα χιλιάδων ή και εκατομμυρίων γραμμών κώδικα, πέρα από το ότι επηρεάζεται και αρνητικά η απόδοση του προγράμματος λόγω των επιπλέον ελέγχων. Αυτό ήταν ένα δίλημμα σε μεγάλα προγράμματα που είχαν γραφτεί σε C, PASCAL, FORTRAN. Συχνά παραλείπονταν ορισμένοι έλεγχοι με αποτέλεσμα την εμφάνιση προβλημάτων κατά τη λειτουργία τους χωρίς μάλιστα να είναι πάντα εύκολη η εύρεση της αιτίας.

Οι πιο σύγχρονες γλώσσες όπως C++ και Java έχουν έναν πιο ενεργητικό μηχανισμό αντιμετώπισης προβλημάτων, τα οποία καλούν *exceptions*. Ο μηχανισμός αυτός εμφανίζει στον προγραμματιστή πιθανά προβλήματα και τον αναγκάζει να τα αντιμετωπίσει. Σε αντίθεση με την προηγούμενη μεθοδολογία (που ήταν σύγχρονη), η τεχνική των exceptions είναι ασύγχρονη, δηλαδή ο κώδικας αντιμετώπισης του προβλήματος, καλείται μόνο αν παρουσιαστεί το πρόβλημα και χωρίζεται από τον υπόλοιπο κώδικα για λόγους καλύτερης οργάνωσης. Η τεχνική είναι η εξής: θα δοκιμάζεται η εκτέλεση ενός τμήματος κώδικα (try) και για κάθε πιθανό πρόβλημα (exception) θα πρέπει να παρέχεται ένας αντίστοιχος μηχανισμός αντιμετώπισης του προβλήματος (catch). Για το προηγούμενο παράδειγμα, ο μηχανισμός θα είχε τη μορφή:

```
try {
   fc = open("αρχείο.dat");
   read(fc, buffer);
   // do stuff
   write(fc, buffer);
   // do stuff
} catch (Exception e) {
   println("error");
}
```

(σημείωση: πρόκειται για ψευδοκώδικα, οι εντολές για προσπέλλαση αρχείων στη Java έχουν διαφορετική δομή, όπως θα δούμε και σε επόμενη ενότητα)

Η παρένθεση στην εντολή `catch` περιγράφει το συγκεκριμένο exception που θέλουμε να αντιμετωπίσουμε (catch). Στη Java κάθε exception είναι στιγμιότυπο της κλάσης `Exception` και συνεπώς αντικείμενο, και χρησιμοποιούμε τη μεταβλητή `e` για να το προσπελλάσουμε. Η κλάση Exception παρέχει ορισμένες μεθόδους που προσφέρουν πληροφορίες για το ίδιο το exception, τη γραμμή του κώδικα στην οποία εμφανίστηκε, το μήνυμα λάθους, κλπ. Ακολουθεί πίνακας με τις σημαντικότερες από αυτές:

Όνομα	*Λειτουργία*
`String getLocalizedMessage()`	Επιστρέφει ένα `String` με το μήνυμα του exception στη ρυθμισμένη γλώσσα του συστήματος
`String getMessage()` `String toString()`	Επιστρέφει ένα `String` με το μήνυμα του exception. Η μέθοδος `toString()` καλείται π.χ. όταν προσπαθούμε να τυπώσουμε ένα αντικείμενο με τη `System.out.println()`.
`void printStackTrace()`	Τυπώνει το όνομα της μεθόδου στην οποία παρουσιάστηκε το σφάλμα (με τη μορφή exception) καθώς και τα ονόματα των καλούντων μεθόδων.

Φυσικά δεν είναι όλα τα σφάλματα της ίδιας φύσεως ούτε έχουν την ίδια αιτία και η κλάση `Exception` απλώς ορίζει ένα μηχανισμό διαχείρισης σφαλμάτων. Χρειάζεται περαιτέρω εξειδίκευση ώστε να διαχωρίζονται οι περιπτώσεις π.χ. της ανεπάρκειας μνήμης και των αριθμητικών σφαλμάτων (διαίρεση με το μηδέν) ή της προσπέλλασης ενός στοιχείου εκτός ορίων του πίνακα. Για το σκοπό αυτό ορίστηκαν αρκετές θυγατρικές κλάσεις της `Exception` (δηλαδή "επεκτείνουν" την κλάση `Exception`) που χρησιμοποιούν το ίδιο πλαίσιο αλλά για διαφορετικό είδος σφαλμάτων. Ακολουθεί πίνακας με ορισμένα από τα βασικότερα exceptions της Java:

Όνομα	*Είδος Exception*
ArithmeticException	Κάποιο αριθμητικό λάθος συνέβη (π.χ. διαίρεση με το μηδέν).
ArrayIndexOutOfBoundsException	Προσπέλαση ενός στοιχείου πίνακα εκτός των ορίων του.
ArrayStoreException	Ορισμός ενός στοιχείου του πίνακα με μη συμβατό αντικείμενο.
NegativeArraySizeException	Δημιουργία πίνακα με αρνητικό μέγεθος.
NullPointerException	Χρήση null αναφοράς σε αντικείμενο.
NumberFormatException	Αδύνατη μετατροπή αντικειμένου String σε αριθμό.
StringIndexOutOfBoundsException	Προσπέλαση χαρακτήρα ενός String εκτός των ορίων του.
UnsupportedOperationException	Η εντολή δεν υποστηρίζεται.
ClassNotFoundException	Η κλάση δε βρέθηκε.
IllegalAccessException	Απαγορεύεται η πρόσβαση των περιεχομένων μιας κλάσης.
InstantiationException	Απαγορεύεται η δημιουργία στιγμιότυπου (instance) αφηρημένης κλάσης.
NoSuchFieldException	Ανύπαρκτο πεδίο.
NoSuchMethodException	Ανύπαρκτη μέθοδος.

Κάθε exception πρέπει να το διαχειριστούμε στο μπλοκ εντολών της αντίστοιχης `catch`. Για καλύτερη κατανόηση, παραθέτουμε το ακόλουθο παράδειγμα:

```
class ExceptionTest {
   public static void main(String args[]) {
      int val = -5;
      int nums[] = new int[10];

      for (int i=0; i < nums.length; i++, val++)
         nums[i] = val;

      for (int i=0; i < 2* nums.length; i++) {
         try {
            // Προσπάθεια διαίρεσης του 10 με το nums[i]
            System.out.println(10 / nums[i]);
         } catch (ArithmeticException e) {
            System.out.println("Arithmetic Exception :"
                  + e.getMessage());
         } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("non-existant elements, at pos:"
                  + e.getMessage());
         }
      }
      System.out.println("Program continues to run...");
   }
```

```
}
```

Αφού μεταγλωττίσουμε το πρόγραμμα με την εντολή `javac`, το εκτελούμε με την `java`:

```
$ java ExceptionTest
-2
-2
-3
-5
-10
Arithmetic Exception :/ by zero
10
5
3
2
non-existant elements, at pos: 10
non-existant elements, at pos: 11
non-existant elements, at pos: 12
non-existant elements, at pos: 13
non-existant elements, at pos: 14
non-existant elements, at pos: 15
non-existant elements, at pos: 16
non-existant elements, at pos: 17
non-existant elements, at pos: 18
non-existant elements, at pos: 19
Program continues to run...
```

Όπως βλέπουμε, το πρόγραμμα συνεχίζει την εκτέλεσή του παρόλο που συνέβησαν σφάλματα. Χωρίς τη χρήση των exceptions, το πρόγραμμα θα είχε σταματήσει με την διαίρεση με το μηδέν και θα έβγαζε το εξής μήνυμα:

```
$ java ExceptionTest
-2
-2
-3
-5
-10
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at ExceptionTest.main(ExceptionTest.java)
```

Σε ένα απλό πρόγραμμα κάτι τέτοιο δεν είναι τραγικό, αλλά ένα τέτοιο σφάλμα σε κάποιο πιο σοβαρό πρόγραμμα, όπως για παράδειγμα σε σύστημα διαχείρισης τραπεζικών λογαριασμών, την ώρα που πραγματοποιείται μια κατάθεση, μπορεί να αποβεί μοιραίο.

Δήλωση των Exceptions με τη throws

Σε κάποιο δικό μας πρόγραμμα πιθανόν να ορίσουμε δικά μας exceptions ή να χρησιμοποιήσουμε εξωτερικές κλάσεις που να ορίζουν δικά τους exceptions. Η Java απαιτεί να δηλώνουμε κάθε exception που πιθανόν να εμφανιστεί κατά την εκτέλεση μιας μεθόδου (ακόμη και της `main()`). Τα βασικά exceptions δε χρειάζεται να δηλωθούν με κάποιο τρόπο, καθώς εννοείται η δήλωσή τους για κάθε μέθοδο. Για ένα οποιοδήποτε άλλο exception θα πρέπει να δηλώνονται κατά τον ορισμό της μεθόδου, π.χ για τη `main`: Περισσότερα παραδείγματα στην επόμενη ενότητα:

```
public static void main(String args[]) throws java.io.IOException {
```

9.Java και Αρχεία

Όπως και κάθε σύγχρονη γλώσσα προγραμματισμού, η Java προσφέρει ένα πλούσιο σύνολο κλάσεων και μεθόδων για διαχείριση αρχείων και ό,τι περιλαμβάνει αυτή. Με τη Java μπορείτε να ανοίξετε αρχεία, να διαβάσετε και να γράψετε δεδομένα στα αρχεία και μάλιστα διατίθενται κλάσεις για να αυτοματοποιηθούν ορισμένες διαδικασίες, π.χ. αυτόματη αποθήκευση ενός πίνακα από `String` σε ένα αρχείο ή φιλτράρισμα των δεδομένων που διαβάζονται από ένα αρχείο.

Αναλόγως τη γλώσσα προγραμματισμού, τα αρχεία αντιμετωπίζονται με διαφορετικό τρόπο, π.χ. στη γλώσσα C τα αρχεία μπορούν να φαίνονται ως πίνακες χαρακτήρων (memory mapped files) ή ως δομές δεδομένων (structs) τις οποίες διαχειριζόμαστε με ειδικές συναρτήσεις/ρουτίνες. Στη C++ τα αρχεία είναι αντικείμενα και η διαχείρισή τους από το χρήστη βασίζεται σε μια λειτουργία συνεχούς ροής χαρακτήρων (streams). Την ίδια φιλοσοφία, με διαφορετική σύνταξη, ακολουθεί και η Java.

I/O με Streams

Η αντιμετώπιση των streams ως ροές δεδομένων από και προς το αρχείο, έχει το άμεσο όφελος ότι δεν είναι πλέον σημαντικό το μέσον στο οποίο βρίσκεται το αρχείο αλλά μόνο η εναλλαγή της πληροφορίας από και πρός το πρόγραμμα. Για το σκοπό αυτό, η Java παρέχει ένα πλήρες σύνολο κλάσεων για την προσπέλαση των δεδομένων ενός αρχείου και μάλιστα σε δύο επίπεδα, ένα σε επίπεδο byte (byte streams) και ένα σε επίπεδο χαρακτήρων (character streams). Όπως μαρτυρά και το όνομα τους, τα byte streams χρησιμοποιούνται για την επεξεργασία των δεδομένων των αρχείων, για ανάγνωση και εγγραφή και ειδικά όταν τα δεδομένα δε μπορούν να απεικονιστούν ως κείμενο. Για την περίπτωση αρχείων κειμένου, τα character streams προσφέρουν επιπλέον λειτουργίες, όπως απευθείας κωδικοποίηση σε κάποιο σύνολο χαρακτήρων (π.χ. ISO-8859-7, UTF-8) αφού εσωτερικά λειτουργούν σε Unicode. Σε μερικές περιπτώσεις μάλιστα και ειδικά για αρχεία κειμένου, τα character streams αποδίδουν καλύτερα από τα byte streams.

Byte Streams

Τα byte streams χρησιμοποιούν μια ιεραρχία από κλάσεις στη κορυφή των οποίων βρίσκονται οι κλάσεις `InputStream` και `OutputStream`. Υπάρχουν αρκετές επιπλέον θυγατρικές κλάσεις που δημιουργήθηκαν για να αντιμετωπίσουν κάποιο διαφορετικό σκοπό (π.χ. χρήση για αρχεία με φίλτρο, χρήση για σωληνώσεις (pipes), κλπ). Όλες όμως αυτές οι κλάσεις βασίζονται στην ίδια δομή που ορίζουν οι `InputStream` και `OutputStream`. Κάτι τέτοιο έχει ως όφελος τη μείωση του χρόνου εκμάθησης της διαχείρισης των αρχείων στη Java.

Αναφέρουμε μερικές από τις βασικότερες κλάσεις που διαχειρίζονται τα byte streams.

Όνομα	*Είδος*
`InputStream`	Η βασική (αφηρημένη) κλάση που ορίζει τη συμπεριφορά ενός byte stream εισόδου (input).
`OutputStream`	Η βασική (αφηρημένη) κλάση που ορίζει τη συμπεριφορά ενός byte stream εξόδου (output).
`BufferedInputStream`	Stream εισόδου που διαβάζει από ένα buffer μνήμης.
`BufferedOutputStream`	Stream εξόδου που γράφει σε ένα buffer μνήμης.
`ByteArrayInputStream`	Stream εισόδου που διαβάζει από ένα πίνακα bytes.

Όνομα	*Είδος*
`ByteArrayOutputStream`	Stream εξόδου που γράφει σε ένα πίνακα bytes.
`DataInputStream`	Stream εισόδου που διαβάζει ακατέργαστα (raw) bytes.
`DataOutputStream`	Stream εξόδου που γράφει ακατέργαστα (raw) bytes.
`FileInputStream`	Stream εισόδου που διαβάζει από ένα αρχείο.
`FileOutputStream`	Stream εξόδου που γράφει σε ένα αρχείο.
`FilterInputStream`	Φιλτραρισμένο Stream εισόδου.
`FilterOutputStream`	Φιλτραρισμένο Stream εξόδου.
`PipedInputStream`	Stream εισόδου που διαβάζει από μια σωλήνωση (pipe).
`PipedOutputStream`	Stream εξόδου που γράφει σε μια σωλήνωση (pipe).
`PrintStream`	Stream εξόδου που ορίζει τις μεθόδους `print()` και `println()`.
`PushBackInputStream`	Stream εισόδου που επιτρέπει και εγγραφή.
`RandomAccessFile`	Stream εισόδου που υποστηρίζει τυχαία προσπέλλαση.
`SequenceInputStream`	Stream εισόδου που είναι συνδυασμός δύο ή περισσοτέρων streams εισόδου, που θα διαβαστούν διαδοχικά.

Τρία από τα προκαθορισμένα streams της Java είναι τα γνωστά `System.out` για την έξοδο, `System.in` για την είσοδο και `System.err` για την έξοδο λαθών (η `stderr` για τη C). Το `System.in` είναι στιγμιότυπο (instance) της `InputStream` ενώ οι `System.out` και `System.err` είναι στιγμιότυπα της `PrintStream` (γι' αυτό και προσφέρουν τις μεθόδους `print()` και `println()`).

Περιγραφή των InputStream & OutputStream

Οι βασικές μέθοδοι της κλάσης `InputStream` περιγράφονται στον ακόλουθο πίνακα:

Όνομα μεθόδου	*Λειτουργία*
`int available()`	Επιστρέφει το αριθμό των διαθέσιμων bytes για ανάγνωση από το stream.
`void close()`	Κλείνει το stream εισόδου.
`int read()`	Επιστρέφει το επόμενο διαθέσιμο byte από το stream εισόδου ως αριθμό τύπου `int`, ή `-1` αν έχουμε φτάσει στο τέλος του αρχείου.
`int read(byte buffer[])`	Διαβάζει τα επόμενα διαθέσιμα bytes (το πλήθος δίνεται από το μέγεθος του πίνακα `buffer[]`, `buffer.length`) και τα αποθηκεύει στον πίνακα `buffer`. Επιστρέφεται το πραγματικό πλήθος των bytes που διαβάστηκαν, ή `-1` αν έχουμε φτάσει στο τέλος του αρχείου.

Όνομα μεθόδου	*Λειτουργία*
`int read(byte buffer, int offset, int maxBytes)`	Διαβάζει τα επόμενα διαθέσιμα bytes (το μέγιστο πλήθος `maxBytes`) και τα αποθηκεύει στον πίνακα `buffer`, ξεκινώντας από τη θέση `offset`. Επιστρέφεται το πραγματικό πλήθος των bytes που διαβάστηκαν, ή `-1` αν έχουμε φτάσει στο τέλος του αρχείου.
`long skip(long numBytes)`	Αγνοεί (δηλαδή προσπερνάει) `numBytes` bytes του stream εισόδου, επιστρέφοντας τον πραγματικό αριθμό των bytes που αγνοήθηκαν.

Πίνακας 1Ορισμένες μέθοδοι της InputStream

Ακολουθεί ο αντίστοιχος πίνακας για την κλάση `OutputStream`:

Όνομα μεθόδου	*Λειτουργία*
`void close()`	Κλείνει το stream εξόδου.
`void flush()`	
`void write(int b)`	Γράφει στο stream εξόδου ένα byte `b`. Το `b` δίνεται ως αριθμός τύπου `int`, αλλά γίνεται αυτόματα casting σε τύπο `byte` πριν την εγγραφή.
`void write(byte buffer[])`	Γράφει ολόκληρο τον πίνακα `buffer` (μεγέθους `buffer.length`) στο stream εξόδου.
`void write(byte buffer, int offset, int maxBytes)`	Γράφει τα στοιχεία του πίνακα `buffer` (το πλήθος `maxBytes`) ξεκινώντας από τη θέση `offset`, στο stream εξόδου.

Πίνακας 2Ορισμένες μέθοδοι της OutputStream

Όπως ίσως σωστά αντιληφθήκατε, οι πιο σημαντικές μέθοδοι για τη χρήση των κλάσεων `InputStream` και `OutputStream` είναι οι `read()` και `write()` αντίστοιχα. Με αυτές πραγματοποιείται η ανάγνωση και εγγραφή δεδομένων από και προς το αρχείο. Σχεδόν κάθε μέθοδος των δύο κλάσεων υπάρχει πιθανότητα να εμφανίσει σφάλμα επικοινωνίας με το αρχείο (I/O) και το σφάλμα αυτό, ακολουθώντας τη φιλοσοφία της Java, θα έχει τη μορφή ενός exception (η έκφραση αγγλιστί είναι “the method *throws* an exception”).

Επειδή οι κλάσεις `InputStream`, `OutputStream` και οι θυγατρικές τους ανήκουν στο πακέτο (package) της Java `java.io`, όλα τα Exceptions που “πετιώνται” από τις μεθόδους των κλάσεων αυτών, θα βασίζονται στη βασική κλάση Exception του `java.io`, την `IOException`. Δηλαδή, κάθε μέθοδος θα “πετάει” ένα `java.io.IOException`.

> **Java packages**: τα πακέτα της Java είναι αυτόνομες μονάδες λογισμικού, που περιέχουν κλάσεις και μεθόδους που χρησιμοποιούνται για ένα συγκεκριμένο σκοπό. Έχουν την ίδια χρησιμότητα με τις βιβλιοθήκες ρουτινών, που χρησιμοποιούν άλλες γλώσσες προγραμματισμού (C, C++). Συνήθως είναι αρκετά γενικευμένης χρήσης, ώστε να επωφελείται ο κάθε προγραμματιστής από κάτι ήδη δοκιμασμένο και να ασχοληθεί με το ίδιο το πρόγραμμα του. Η

Java παρέχει πληθώρα τέτοιων πακέτων, για γραφικά, προγραμματισμό βάσεων δεδομένων, δικτυακό προγραμματισμό, κλπ. Ένα από τα βασικά τέτοια πακέτα, είναι για την διαχείριση των αρχείων, το `java.io`.

Τα πακέτα τα χρησιμοποιούμε στα προγράμματά μας με την εντολή import, εκτός της κλάσης, δηλαδή:

```
import java.io.*;
```

Η παραπάνω εντολή επιτρέπει στο πρόγραμμά μας να χρησιμοποιήσει το πακέτο `java.io` και όλες τις κλάσεις (ή άλλα πακέτα) περιέχει αυτό.

Για την καλύτερη κατανόηση της χρήσης των κλάσεων `InputStream` και `OutputStream`, θα δώσουμε ένα παράδειγμα προγράμματος αντιγραφής αρχείων (σαν την εντολή `copy` του DOS ή `cp` του UNIX). Θα ονομάσουμε το πρόγραμμά μας `CopyFile` και η χρήση του θα είναι η εξής:

```
C:\> java CopyFile file1.dat file2.dat
```

θα αντιγράφει το `file1.dat` στο `file2.dat`. Αν το `file2.dat` δεν υπάρχει, θα το δημιουργεί. Αν εκτελεστεί η εντολή χωρίς παραμέτρους θα μας επιστρέφει πληροφορίες για τη σύνταξή της (Usage).

```
import java.io.*;

class CopyFile {
   // η main() "πετάει" IOExceptions
   public static void main(String args[]) throws IOException {
      int i;
      FileInputStream fin;
      FileOutputStream fout;

      // το κυρίως block της try
      try {
         // άνοιξε το αρχείο εισόδου
         try {
            fin = new FileInputStream(args[0]);
         } catch (FileNotFoundException exc) {
            System.out.println("Input file not found: " + args[0]);
            return;
         }

         // άνοιξε το αρχείο προορισμού
         try {
            fout = new FileOutputStream(args[1]);
         } catch (FileNotFoundException exc) {
            System.out.println("Error opening output file: " + args[0]);
            return;
         }
      // αν δώσουμε λιγότερες από δύο παραμέτρους, θα βγούμε εκτός ορίων
      // του πίνακα args[] και θα έχουμε το ακόλουθο exception.
      } catch (ArrayIndexOutOfBoundsException exc) {
         System.out.println("Usage: CopyFile From To");
         return;
```

```
      }

      // Τώρα που ανοίξαμε τα αρχεία μπορούμε να πραγματοποιήσουμε την
      // αντιγραφή:
      try {
         do {
            i = fin.read();
            if (i != -1)               // δηλαδή αν δεν έχουμε φτάσει
               fout.write(i);          // στο τέλος του αρχείου fin
         } while (i != -1);
      } catch (IOException exc) {
         System.out.println("File error");
      }

      // κλείσε και τα δύο αρχεία
      fin.close();
      fout.close();
   }
}
```

Σε αυτό το παράδειγμα δείξαμε ένα συνηθισμένο τρόπο χρήσης των κλάσεων `FileInputStream` και `FileOutputStream`, όχι όμως τον καλύτερο. Αν δοκιμάσετε να εκτελέσετε το πρόγραμμα αυτό θα διαπιστώσετε ότι είναι πολύ πιο αργό από μια συνηθισμένη εντολή `copy`. Για παράδειγμα σε ένα σύστημα Linux Pentium III/ 700 Mhz, η αντιγραφή ενός αρχείου μεγέθους 56 MB με την εντολή `cp` διήρκεσε 11 δευτερόλεπτα. Με το πρόγραμμά μας, η ίδια διαδικασία διήρκεσε 4 λεπτά και 44 δευτερόλεπτα!!!

Σίγουρα υπάρχει καλύτερος τρόπος να πραγματοποιήσουμε την ίδια λειτουργία και αυτός υλοποιείται με τις παραλλαγές των μεθόδων `read()` και `write()` που χρησιμοποιούν ως παράμετρο ένα buffer. Το ίδιο πρόγραμμα με τη νέα μέθοδο, διαμορφώνεται ως εξής:

```
      // Δίνουμε ένα μέγεθος που να είναι δύναμη του 2 για
      // ταίριασμα με τα μεγέθη των blocks του δίσκου. 16384 bytes είναι
      // ένα αρκετά καλό μέγεθος που εγγυάται καλή ταχύτητα αντιγραφής σε
      // μεγάλα αρχεία
      int bufsize = 16384;
      byte buffer[] = new byte[bufsize];

      // Τώρα που ανοίξαμε τα αρχεία μπορούμε να πραγματοποιήσουμε την
      // αντιγραφή:
      try {
         do {
            // διαβάζει ένα buffer, η μεταβλητή i περιέχει τον αριθμό
            // των bytes που πραγματικά διαβάστηκαν.
            i = fin.read(buffer);
            if (i != -1)
               // αν δεν έχουμε φτάσει στο τέλος του αρχείου fin
               // γράψε στο αρχείο fout τα bytes του buffer που διαβάστηκαν
               // κατά τη προηγούμενη read()
               fout.write(buffer, 0, i);
         } while (i != -1);
      } catch (IOException exc) {
         System.out.println("File error");
      }
```

Στην περίπτωση αυτή τα αποτελέσματα είναι θεαματικά. Η εκτέλεση του νέου πρόγραμματος για το ίδιο αρχείο στο ίδιο σύστημα διήρκεσε 8,6 δευτερόλεπτα περίπου! Το όφελος είναι προφανές.

Οι υπόλοιπες κλάσεις των byte streams χρησιμοποιούνται με παρόμοιο τρόπο για διαφορετικό σκοπό (π.χ. για αποθήκευση ενός πίνακα δεδομένων απευθείας σε αρχείο με την `ByteArrayOutputStream`, ανάγνωση δεδομένων από μια συνεχή ροή δεδομένων μέσω σωλήνωσης (pipe) με την `PipedInputStream`, κλπ). Ο σκοπός της επεξήγησης της λειτουργίας της κάθε κλάσης όμως δεν είναι στα πλαίσια αυτού του οδηγού.

Streams Χαρακτήρων

Όπως αναφέραμε στην αρχή του κεφαλαίου, η Java προσφέρει και έναν επιπλέον τρόπο διαχείρισης των αρχείων, τα streams χαρακτήρων (character streams). Και σε αυτήν την περίπτωση ορίζονται δύο βασικές κλάσεις κατ' αντιστοιχία με τις κλάσεις `InputStream` και `OutputStream` και αρκετές θυγατρικές κλάσεις για διαφορετικές ανάγκες. Οι βασικές κλάσεις είναι οι (αφηρημένες) `Reader` και `Writer` ενώ ο ακόλουθος πίνακας περιγράφει συνοπτικά τις σημαντικότερες κλάσεις των streams χαρακτήρων:

Όνομα	*Είδος*
`Reader`	Η βασική (αφηρημένη) κλάση που ορίζει τη συμπεριφορά ενός character stream εισόδου (input).
`Writer`	Η βασική (αφηρημένη) κλάση που ορίζει τη συμπεριφορά ενός character stream εξόδου (output).
`BufferedReader`	Stream εισόδου που διαβάζει από ένα buffer μνήμης.
`BufferedWriter`	Stream εξόδου που γράφει σε ένα buffer μνήμης.
`CharArrayReader`	Stream εισόδου που διαβάζει από ένα πίνακα χαρακτήρων.
`CharArrayWriter`	Stream εξόδου που γράφει σε ένα πίνακα χαρακτήρων.
`FileReader`	Stream εισόδου που διαβάζει από ένα αρχείο.
`FileWriter`	Stream εξόδου που γράφει σε ένα αρχείο.
`FilterReader`	Φιλτραρισμένο Stream εισόδου.
`FilterWriter`	Φιλτραρισμένο Stream εξόδου.
`InputStreamReader`	Stream εισόδου που μετατρέπει τα bytes σε χαρακτήρες.
`LineNumberReader`	Stream εισόδου μετρά γραμμές.
`OutputStreamWriter`	Stream εξόδου που μετατρέπει τους χαρακτήρες σε bytes.
`PipedReader`	Stream εισόδου που διαβάζει από μια σωλήνωση (pipe).
`PipedWriter`	Stream εξόδου που γράφει σε μια σωλήνωση (pipe).
`PrintWriter`	Stream εξόδου που ορίζει τις μεθόδους `print()` και `println()`.
`PushBackReader`	Stream εισόδου που επιτρέπει και εγγραφή.
`StringReader`	Stream εισόδου από αντικείμενο `String`.
`StringWriter`	Stream εξόδου σε αντικείμενο `String`.

Αντίστοιχα, οι μέθοδοι που ορίζονται από τις κλάσεις `Reader` και `Writer`, ακολουθούν την ίδια δομή με των κλάσεων `InputStream` και `OutputStream`, όπως αναγράφονται στους ακόλουθους πίνακες:

Όνομα μεθόδου	*Λειτουργία*
`int available()`	Επιστρέφει το αριθμό των διαθέσιμων bytes για ανάγνωση από το stream.
`void close()`	Κλείνει το stream εισόδου.
`int read()`	Επιστρέφει τον επόμενο διαθέσιμο χαρακτήρα από το stream εισόδου ως αριθμό τύπου `int`, ή `-1` αν έχουμε φτάσει στο τέλος του αρχείου.
`int read(char buffer[])`	Διαβάζει τους επόμενους διαθέσιμους χαρακτήρες (το πλήθος δίνεται από το μέγεθος του πίνακα `buffer[]`, `buffer.length`) και τους αποθηκεύει στον πίνακα `buffer`. Επιστρέφεται το πραγματικό πλήθος των χαρακτήρων που διαβάστηκαν, ή `-1` αν έχουμε φτάσει στο τέλος του αρχείου.
`abstract int read(char buffer, int offset, int maxChars)`	Διαβάζει τους επόμενους διαθέσιμους χαρακτήρες (το μέγιστο πλήθος `maxChars`) και τους αποθηκεύει στον πίνακα `buffer`, ξεκινώντας από τη θέση `offset`. Επιστρέφεται το πραγματικό πλήθος των χαρακτήρων που διαβάστηκαν, ή `-1` αν έχουμε φτάσει στο τέλος του αρχείου.
`long skip(long numChars)`	Αγνοεί (δηλαδή προσπερνάει) `numChars` χαρακτήρων του stream εισόδου, επιστρέφοντας τον πραγματικό αριθμό των χαρακτήρων που αγνοήθηκαν.

Πίνακας 3Ορισμένες μέθοδοι της Reader

και ο αντίστοιχος πίνακας για την κλάση `Writer`:

Όνομα μεθόδου	*Λειτουργία*
`abstract void close()`	Κλείνει το stream εξόδου.
`abstract void flush()`	
`void write(int ch)`	Γράφει στο stream εξόδου ένα χαρακτήρα `ch`. Ο `ch` δίνεται ως αριθμός τύπου `int`, αλλά γίνεται αυτόματα casting σε τύπο `char` πριν την εγγραφή.
`void write(char buffer[])`	Γράφει ολόκληρο τον πίνακα `buffer` (μεγέθους `buffer.length`) στο stream εξόδου.
`abstract void write(char buffer, int offset, int maxChars)`	Γράφει τα στοιχεία του πίνακα `buffer` (το πλήθος `maxChars`) ξεκινώντας από τη θέση `offset`, στο stream εξόδου.
`void write(String str)`	Γράφει το `String str` στο stream εξόδου.

Όνομα μεθόδου	*Λειτουργία*
`void write(String str, int offset, int maxChars)`	Γράφει τα στοιχεία του `String str` (το πλήθος `maxChars`) ξεκινώντας από τη θέση `offset`, στο stream εξόδου.

Πίνακας 4Ορισμένες μέθοδοι της Writer

Όπως αναφέρθηκε, οι διαθέσιμες κλάσεις των stream χαρακτήρων χρησιμοποιούνται η κάθε μία για διαφορετικό σκοπό. Για παράδειγμα, μια από τις καλύτερες κλάσεις για ανάγνωση χαρακτήρων από την κονσόλα ή από ένα αρχείο (σε συνδυασμό με την `FileReader`) είναι η `BufferedReader`. Το ακόλουθο πρόγραμμα χρησιμοποιεί και τις δύο αυτές κλάσεις για την ανάγνωση ενός αρχείου κειμένου και την εκτύπωση του στην έξοδο (κάτι σαν την εντολή `type` του DOS ή `more/less` του UNIX). Για την εκτύπωση χρησιμοποιείται η κλάση `PrintWriter`. Η χρήση της συγκεκριμένης κλάσης συνιστάται αντί της συνηθισμένης `System.out.println()`, και ειδικά σε επαγγελματίες, γιατί προσφέρει δυνατότητα εύκολης διεθνοποίησης (δηλαδή προσαρμογή για εκτύπωση μηνυμάτων σε διάφορες γλώσσες, πέραν της αγγλικής).

```
import java.io.*;

class PrintFile {
   public static void main(String args[]) {
      // Δήλωσε τα βασικά αντικείμενα του προγράμματος
      FileReader fr;
      BufferedReader br;
      PrintWriter pw;
      String str;

      // Θα πρέπει να δώσουμε ένα ή περισσότερα αρχεία ως παραμέτρους
      // για να μπορέσουμε να τα τυπώσουμε.
      if (args.length == 0)
         System.out.println("Usage: PrintFile <file>...");

      // Το βασικό block ελέγχου των exceptions
      try {
         // Δημιούργησε το PrintWriter για την εκτύπωση στην έξοδο
         pw = new PrintWriter(System.out, true);

         // Για κάθε μία από τις παραμέτρους, δημιούργησε ένα FileReader.
         // Σε κάθε FileReader, σύνδεσε ένα BufferedReader.
         for (int i=0; i < args.length; i++) {
            fr = new FileReader(args[i]);
            br = new BufferedReader(fr);

            // Διάβαζε το κείμενο από το BufferedReader, μία γραμμή κάθε
            // φορά. Όσο έχουμε κείμενο, το τυπώνουμε στο PrintWriter
            while ((str = br.readLine()) != null)
               pw.println(str);
         }
      } catch (IOException e) {
         System.out.println("File error: " +e);
      }
   }
}
```

10.Δικτυακός Προγραμματισμός

Ο προγραμματισμός σε επίπεδο δικτύου είναι ένα από τα πιο δυνατά σημεία της Java. Σε αντίθεση με άλλες γλώσσες που προσφέρουν ένα τελείως αυτόνομο σύστημα διαχείρισης και επικοινωνίας δεδομένων σε ένα δίκτυο, η Java χρησιμοποιεί όσο το δυνατόν περισσότερο την ήδη υπάρχουσα δομή των κλάσεων των Streams. Για την Java δεν έχει σημασία αν το αρχείο/stream είναι στο δίσκο ή σε μια σελίδα στο Internet, χρησιμοποιεί τον ίδιο τρόπο προσπέλλασης και για τα δύο κάτι που διευκολύνει σε μεγάλο βαθμό τον προγραμματιστή.

Οπωσδήποτε όμως προσφέρει επιπλέον κλάσεις για τον προγραμματισμό σε δικτυακό επίπεδο. Οι κλάσεις αυτές περιλαμβάνονται στο πακέτο `java.net` και προσφέρουν επικοινωνία μέσω των πρωτοκόλλων TCP/IP και UDP. Αναφορικά, οι πιο βασικές από τις κλάσεις αυτές είναι: `URL`, `URLConnection`, `Socket`, and `ServerSocket` για το πρωτόκολλο TCP, ενώ οι `DatagramPacket`, `DatagramSocket`, and `MulticastSocket` χρησιμοποιούνται για επικοινωνία μέσω UDP.

Η κλάση URL

Η κλάση αυτή χρησιμοποιείται για τον ορισμό μιας δικτυακής διεύθυνσης (κυρίως για χρήση με πρωτόκολλα όπως HTTP, HTTPS, FTP, κλπ), όπως χρησιμοποιούνται σε προγράμματα web browsers. Το URL είναι τα αρχικά των λέξεων *Uniform Resource Locator*.

Για παράδειγμα, αν έχουμε το δικτυακό τόπο www.mysite.net, έγκυρα URLs θεωρούνται τα ακόλουθα:

http://www.mysite.net/

http://www.mysite.net/pages/article.html

Μπορείτε να δημιουργήσετε αντικείμενα URL πολύ εύκολα και ακόμη να τα συνδέσετε μεταξύ τους ώς εξής:

```
URL mysite = new URL("http://www.mysite.net/pages/");
URL mysiteArticle = new URL(mysite, "article.html");
URL mysiteForm = new URL(mysite, "form.html");
```

Στο παράδειγμα αυτό, χρησιμοποιήσαμε την κλάση URL καλώντας το δημιουργό της (constructor) με τις εξής ισοδύναμες μορφές:

```
URL(URL baseURL)
URL(URL baseURL, String relativeURL)
```

Η κλάση `URL` προσφέρει τη δυνατότητα ελέγχου της εγκυρότητας της διεύθυνσης που περνάμε ως string, με τη χρήση ενός Exception τύπου `MalformedURLException`.

```
try {
    URL myURL = new URL(. . .)
} catch (MalformedURLException e) {
    . . .
    // κώδικας χειρισμού του Exception
    . . .
}
```

Πέρα από τη δημιουργία ενός αντικειμένου URL, η κλάση URL προσφέρει και αρκετές μεθόδους για τη λήψη πληροφορίας από μια υπάρχουσα διεύθυνση. Οι μέθοδοι αυτοί είναι:

Όνομα μεθόδου	*Λειτουργία*
`String getProtocol()`	Επιστρέφει το όνομα του πρωτοκόλλου (http, ftp).
`String getHost()`	Επιστρέφει το όνομα του δικτυακού τόπου, π.χ. www.mysite.net.
`String getFile()`	Επιστρέφει το όνομα της σελίδας, π.χ. "`/pages/article.html`".
`String getPort()`	Επιστρέφει τον αριθμό της θύρας (συνήθως 80 για το http και 21 για το ftp).
`String getRef()`	Επιστρέφει το όνομα του σελιδοδείκτη της σελίδας (αυτό που δηλώνεται με '#').

Το ακόλουθο παράδειγμα, δείχνει τη χρήση των μεθόδων της κλάσης URL.

```
import java.net.*;
import java.io.*;

public class ParseURL {
    public static void main(String[] args) throws Exception {
        URL aURL = new URL("http://java.sun.com:80/docs/books/"
                           + "tutorial/index.html#DOWNLOADING");
        System.out.println("protocol = " + aURL.getProtocol());
        System.out.println("host = " + aURL.getHost());
        System.out.println("filename = " + aURL.getFile());
        System.out.println("port = " + aURL.getPort());
        System.out.println("ref = " + aURL.getRef());
    }
}
```

Και ιδού το αποτέλεσμα:

```
protocol = http
host = java.sun.com
filename = /docs/books/tutorial/index.html
port = 80
ref = DOWNLOADING
```

Η κλάση URLConnection

Για τη δημιουργία μιας σύνδεσης με μία διεύθυνση URL, χρησιμοποιείται η κλάση URLConnection, αν και σε αυτήν την κλάση η έμφαση είναι σε συνδέσεις HTTP. Η ίδια η κλάση επιτρέπει μεταφορά δεδομένων από και πρός τη διεύθυνση URL και προσφέρει αρκετές μεθόδους γι' αυτόν τον σκοπό.

Μεταφορά δεδομένων από το URLConnection

Θα παρουσιάσουμε τη χρήση της URLConnection με το ακόλουθο πρόγραμμα:

```
import java.net.*;
import java.io.*;

public class URLConnectionReader {
    public static void main(String[] args) throws Exception {
        URL yahoo = new URL("http://www.yahoo.com/");
        URLConnection yc = yahoo.openConnection();
        BufferedReader in = new BufferedReader(
                                new InputStreamReader(
                                yc.getInputStream()));
        String inputLine;

        while ((inputLine = in.readLine()) != null)
            System.out.println(inputLine);
        in.close();
    }
}
```

Στο πρόγραμμα αυτό χρησιμοποιούμε την μέθοδο `openConnection()` της κλάσης `URL`, η οποία επιστρέφει ένα αντικείμενο `URLConnection yc`. Το αντικείμενο αυτό προσφέρει το πλαίσιο με το οποίο μπορούμε να ανταλλάξουμε δεδομένα από τη διεύθυνση www.yahoo.com, και μάλιστα χρησιμοποιώντας τα πλέον γνωστά Streams (βλ. προηγούμενο κεφάλαιο). Συγκεκριμένα, δημιουργούμε ένα αντικείμενο `BufferedReader` στο οποίο περνάμε το `InputStream` που αντιστοιχεί στη σύνδεση με το www.yahoo.com. Κάθε αντικείμενο `URLConnection` προσφέρει ένα αντίστοιχο `InputStream` με τη μέθοδο `getInputStream()` από το οποίο μπορούμε να διαβάσουμε δεδομένα, με τον ίδιο τρόπο που θα διαβάζαμε δεδομένα από ένα αρχείο.

Μεταφορά δεδομένων προς το URLConnection

Όπως πιθανότατα έχετε διαπιστώσει, πολλές σελίδες HTML στο Internet περιέχουν φόρμες με πεδία κειμένου ή άλλα αντικείμενα που σας επιτρέπουν να αποστείλετε δεδομένα στο server (που περιέχει τις σελίδες). Με την αποστολή των δεδομένων από το browser, τα δεδομένα λαμβάνονται από ενα πρόγραμμα CGI (γραμμένο σε γλώσσα Perl/PHP/ASP/κλπ), το οποίο κατόπιν επεξεργασίας επιστρέφει κάποια απάντηση με τη μορφή μιας νέας σελίδας HTML. Η αποστολή γίνεται με τη μέθοδο POST (POST METHOD) στις σελίδες HTML. Η Java προσφέρει χρήση αυτής της μεθόδου για αποστολή δεδομένων σε μία διεύθυνση URL, και χρησιμοποιεί τις ίδιες κλάσεις και μεθόδους όπως και για αποθήκευση δεδομένων σε αρχεία. Γενικά, τα βήματα που πρέπει να ακολουθηθούν είναι τα εξής:

- Δημιουργία ενός αντικειμένου URL (δηλαδή, μιας διεύθυνσης URL)
- Δημιουργία σύνδεσης με το URL (αντικείμενο URLConnection)
- Ενεργοποίηση αποστολής δεδομένων στο αντικείμενο URLConnection
- Δημιουργία ενός stream εξόδου προς στη σύνδεση (OutputStream). Αυτό συνδέεται με την είσοδο δεδομένων (input stream) του προγράμματος CGI στο server.
- Αποθήκευση δεδομένων στο stream εξόδου.

- Κλείσιμο του stream εξόδου.

Ακολουθεί ένα απλό πρόγραμμα που αναζητά λέξεις-κλειδιά στη σελίδα αναζήτησης της Sun για τη Java. Το αποτέλεσμα του προγράμματος είναι η σελίδα HTML που θα βλέπαμε σε κάποιο browser αν χρησιμοποιούσαμε αυτή τη μηχανή αναζήτησης.

```
import java.io.*;
import java.net.*;

public class JavaSearch {
    public static void main(String[] args) throws Exception {

        if (args.length != 1) {
            System.err.println("Usage: java JavaSearch <string>");
            System.exit(1);
        }

        String msg = URLEncoder.encode(args[0], "ISO-8859-1");

        URL baseurl = new URL("http://onesearch.sun.com/");
        URL url = new URL(baseurl, "search/developers/index.jsp");

        URLConnection connection = url.openConnection();
        connection.setDoOutput(true);

        PrintWriter out = new PrintWriter(
                              connection.getOutputStream());
        out.println("qt=" + msg);
        out.close();

        BufferedReader in = new BufferedReader(
                                new InputStreamReader(
                                connection.getInputStream()));
        String inputLine;

        while ((inputLine = in.readLine()) != null)
            System.out.println(inputLine);

        in.close();
    }
}
```

Ας εξετάσουμε τις σημαντικότερες εντολές του προγράμματος:

```
if (args.length != 1) {
     System.err.println("Usage: java JavaSearch <string>");
     System.exit(1);
}

String msg = URLEncoder.encode(args[0], "UTF-8");
```

Αυτές οι εντολές καταρχάς εξασφαλίζουν ότι ο χρήστης δίνει μόνο μια παράμετρο στο πρόγραμμα όταν το καλεί, την οποία και κωδικοποιεί με το σετ κωδικοποίησης UTF-8 και αποθηκεύει κωδικοποιημένη στο αντικείμενο `msg` (τύπου `String`). Επειδή η μεταβλητή `msg` είναι αυτή που θα ληφθεί από CGI πρόγραμμα (όπως ακριβώς αν τη γράφαμε στη

γραμμή διευθύνσεων του browser μας), θα πρέπει να κωδικοποιηθεί για να αποφευχθεί οποιαδήποτε αποστολή λανθασμένων δεδομένων. Η κωδικοποίηση αυτή επιτυγχάνεται με την κλάση `URLEncoder`.

Έπειτα το πρόγραμμα δημιουργεί το αντικείμενο `URL`, τη διεύθυνση δηλαδή που αντιστοιχεί στη μηχανή αναζήτησης, ανοίγει μια σύνδεση `URLConnection` με τη διεύθυνση αυτή και εξασφαλίζει ότι μπορούμε να στείλουμε δεδομένα:

```
URL baseurl = new URL("http://onesearch.sun.com/");
URL url = new URL(baseurl, "search/developers/index.jsp");

URLConnection connection = url.openConnection();
connection.setDoOutput(true);
```

Το πρόγραμμα τότε δημιουργει ένα stream εξόδου στη σύνδεση αυτή και το συνδέει με ένα αντικείμενο `PrintWriter`:

```
PrintWriter out = new PrintWriter(c.getOutputStream());
```

Αν η διεύθυνση αυτή δεν υποστηρίζει έξοδο δεδομένων από το πρόγραμμα (δηλαδή είσοδο δεδομένων για τη φόρμα και το CGI script) τότε η μέθοδος `getOutputStream()` θα παράγει ένα `UnknownServiceException`. Αν υποστηρίζει έξοδο δεδομένων τότε η μέθοδος αυτή θα επιστρέψει ένα stream εξόδου που μπορούμε πλέον να χρησιμοποιήσουμε για να αποστείλουμε δεδομένα στο server.

Το επόμενο βήμα είναι να γράψουμε δεδομένα στο stream εξόδου και να το κλείσουμε. Αυτό γίνεται με ακριβώς τον ίδιο τρόπο όπως γράφουμε σε ένα `PrintStream` (όπως π.χ. το `System.out`):

```
out.println("qt=" + msg);
out.close();
```

Η μέθοδος που χρησιμοποιείται είναι η γνωστή μας `println()`. Αυτές οι ομοιότητες δεν είναι τυχαίες. Η Java έχει σχεδιαστεί έτσι ώστε να προσφέρει μια ομοιομορφία σε όλο το φάσμα των κλάσεων και μεθόδων της. Ο τρόπος με τον οποίο αποστέλλονται τα δεδομένα είναι κρυμμένος από το χρήστη (π.χ. το άνοιγμα sockets σε επίπεδο TCP/IP, ο διαχωρισμός των δεδομένων σε packets, κλπ) ώστε η επικοινωνία να γίνεται με όσο το δυνατόν πιο λίγες διαφορές με τα αρχεία.

Έχοντας στείλει τα δεδομένα στο CGI script της φόρμας, πλέον αναμένουμε την απάντηση με τη μορφή μιας σελίδας HTML (όπως θα τη βλέπαμε σε ένα browser). Για το σκοπό αυτό χρησιμοποιούμε ένα αντικείμενο `InputStreamReader`, από όπου διαβάζουμε τη σελίδα γραμμή-γραμμή (με τη βοήθεια της μεθόδου `readline()`):

```
BufferReader in = new BufferedReader(
                      new InputStreamReader(c.getInputStream()));
String inputLine;

while ((inputLine = in.readLine()) != null)
    System.out.println(inputLine);
in.close();
```

Το πρόγραμμα τυπώνει αυτά που διαβάζει στην πρότυπη έξοδο του (`System.out`) την οποία αν θέλουμε ανακατευθύνουμε σε ένα αρχείο.

Αντιγραφή ενός URL σε ένα αρχείο

Στο προηγούμενο κεφάλαιο παρουσιάσαμε ένα παράδειγμα προγράμματος αντιγραφής αρχείων, το `CopyFile`. Το επόμενο παράδειγμα αντιγράφει μία διεύθυνση `URL` σε ένα αρχείο χρησιμοποιώντας την ίδια δομή του `CopyFile`. Η διαφορά είναι ότι πλέον δε χρησιμοποιείται η κλάση `FileInputStream` για είσοδο δεδομένων, αλλά ο συνδυασμός των κλάσεων `URL` και `URLConnection`.

```
import java.net.*;
import java.io.*;

public class URLConnectionReader {
    public static void main(String[] args) throws Exception {
   URL from;
   FileOutputStream fout;
   int i;

   try {

      try {
         from = new URL(args[0]);
      } catch (MalformedURLException exc) {
         System.out.println("Error opening URL: " + args[0]);
         return;
      }

      try {
         fout = new FileOutputStream(args[1]);
      } catch (FileNotFoundException exc) {
         System.out.println("Error opening output file: " + args[0]);
         return;
      }
   // αν δώσουμε λιγότερες από δύο παραμέτρους, θα βγούμε εκτός ορίων
   // του πίνακα args[] και θα έχουμε το ακόλουθο exception.
   } catch (ArrayIndexOutOfBoundsException exc) {
      System.out.println("Usage: URLConnectionReader From To");
      return;
   }

   URLConnection yc = from.openConnection();
   InputStream fin = yc.getInputStream();

   // Τώρα που ανοίξαμε τα αρχεία μπορούμε να πραγματοποιήσουμε την
   // αντιγραφή:
   int bufsize = 8192;
   byte buffer[] = new byte[bufsize]; // Δίνουμε ένα μέγεθος που να
                 // είναι δύναμη του 2 για
                 // ταίριασμα με τα μεγέθη των
                 // blocks του δίσκου

   try {
      do {
         i = fin.read(buffer);
         if (i != -1)                  // δηλαδή αν δεν έχουμε φτάσει
```

```
            fout.write(buffer, 0, i);  // στο τέλος του αρχείου fin
        } while (i != -1);
    } catch (IOException exc) {
        System.out.println("File error");
    }

        in.close();
    }
}
```

Προγραμματισμός με Sockets

Η επικοινωνία με sockets είναι ο βασικός τρόπος επικοινωνίας μεταξύ υπολογιστών σε δίκτυα TCP/IP, και κατά συνέπεια σε όλο το Internet, αφού είναι βασισμένο σε αυτό το πρωτόκολλο. Ο προγραμματισμός σε sockets είναι σαφώς χαμηλοτέρου επιπέδου από ότι με τις κλάσεις `URLConnection`, αλλά η Java και πάλι προσφέρει ένα τρόπο να μειωθεί η δυσκολία που συναντάται σε άλλες γλώσσες (π.χ. C/C++) συνδέοντας τις κλάσεις που χειρίζονται τα sockets με τις δομές των streams. Οι κλάσεις που παρέχει η Java είναι η `Socket`, για δημιουργία ενός socket από τη πλευρά του προγράμματος πελάτη (client) και την `ServerSocket`, για την δημιουργία των sockets από τη πλευρά του εξυπηρετητή/διακομιστή (server).

Λόγω της πολυπλοκότητας της υλοποίησης των `ServerSocket`, στην ενότητα αυτή θα εξετάσουμε μόνο τα απλά `Socket`. Για το σκοπό αυτό θα χρησιμοποιήσουμε ένα παράδειγμα προγράμματος πελάτη που συνδέεται σε ένα server και ανταλλάσει δεδομένα με το server.

Το πρόγραμμα αυτό (που ονομάζουμε `EchoClient`), συνδέεται σε ένα απλό server του πρωτοκόλλου echo. Το πρωτόκολλο echo είναι ένα απλό πρωτόκολλο ελέγχου και η μόνη λειτουργία του είναι να επιστρέφει τα δεδομένα που λαμβάνει. Είναι μια αρκετά διαδεδομένη υπηρεσία που χρησιμοποιεί τη θύρα 7 (port 7).

Το πρόγραμμα `EchoClient` με τη σύνδεσή του με το Echo server δημιουργεί ένα socket, στο οποίο αποστέλει οτιδήποτε γράψει ο χρήστης στο πληκτρολόγιο. Ο server απαντά αποστέλοντας πίσω στον client το ίδιο κείμενο. Αυτό είναι ένα απλό πρόγραμμα χωρίς ιδιαίτερη επεξεργασία των δεδομένων που ανταλλάσονται μεταξύ του server και του client. Συνήθως, σε πιο πολύπλοκα πρωτόκολλα όπως το FTP, το HTTP και άλλα, ο client στέλνει εντολές στον server τις οποίες ο δεύτερος επεξεργάζεται και στέλνει το ανάλογο αποτέλεσμα (π.χ. τη λίστα των αρχείων του τρέχοντος καταλόγου). Ακολουθεί η λίστα του προγράμματος `EchoClient`:

```
import java.io.*;
import java.net.*;

public class EchoClient {
    public static void main(String[] args) throws IOException {

        Socket echoSocket = null;
        PrintWriter out = null;
        BufferedReader in = null;

        try {
            echoSocket = new Socket("server", 7);
            out = new PrintWriter(echoSocket.getOutputStream(), true);
```

```
            in = new BufferedReader(new InputStreamReader(
                                        echoSocket.getInputStream()));
        } catch (UnknownHostException e) {
            System.err.println("Don't know about host: servername.");
            System.exit(1);
        } catch (IOException e) {
            System.err.println("Couldn't get I/O for "
                               + "the connection to: servername.");
            System.exit(1);
        }

        BufferedReader stdIn = new BufferedReader(
                                   new InputStreamReader(System.in));
        String userInput;

        while ((userInput = stdIn.readLine()) != null) {
            out.println(userInput);
            System.out.println("echo: " + in.readLine());
        }

        out.close();
        in.close();
        stdIn.close();
        echoSocket.close();
    }
}
```

Ας δούμε τις σημαντικότερες εντολές αυτού του προγράμματος. Οι πιο κρίσιμες για την εκτέλεση είναι οι τρεις εντολές στο μπλόκ της `try`. Αυτές δημιουργούν τη σύνδεση μέσω `Socket` με το server και δημιουργούν ένα `PrintWriter` και `BufferReader` στο `Socket`.

```
echoSocket = new Socket("server", 7);
out = new PrintWriter(echoSocket.getOutputStream(), true);
in = new BufferedReader(new InputStreamReader(
                             echoSocket.getInputStream()));
```

Η πρώτη εντολή δημιουργεί το νέο αντικείμενο `Socket` με το όνομα `echoSocket`. Οι παράμετροι του δημιουργού της `Socket` είναι δύο, το όνομα του υπολογιστή με τον οποίο θα γίνει η σύνδεση και ο αριθμός της θύρας. Το όνομα στην περίπτωση αυτή είναι `server`, ώστε το πρόγραμμα θα προσπαθήσει να συνδεθεί με ένα υπολογιστή ονόματι `server` στο τοπικό δίκτυο. Αν πρόκειται η σύνδεση να γίνει με οποιοδήποτε άλλο υπολογιστή, θα χρειαστείτε τη διεύθυνση IP ή το πλήρως ορισμένο όνομα DNS του υπολογιστή (FQDN). Η θύρα είναι η 7 (τυπική θύρα της υπηρεσίας echo).

Η δεύτερη και η τρίτη εντολή συνδέουν ένα αντικείμενο `PrintWriter` και ένα αντικείμενο `BufferReader` στο `Socket`. Χρησιμοποιούμε Reader και Writer αντικείμενα για να μπορούμε να στείλουμε χαρακτήρες `Unicode`.

Η αποστολή δεδομένων στο socket γίνεται με απλή εγγραφή στο `PrintWriter` ενώ η λήψη δεδομένων με ανάγνωση από το `BufferedReader`. Η διαδικασία δε διαφέρει από αυτή των απλών αρχείων και δε θα χρειαστεί να την εξηγήσουμε περαιτέρω.

Οι επόμενες εντολές που παρουσιάζουν ενδιαφέρον είναι το loop `while`, όπου το πρόγραμμα διαβάζει την πρότυπη είσοδο (`System.in`) ανά γραμμή, χρησιμοποιώντας και

εδώ ένα `BufferedReader`. Όσο ο χρήστης έχει κάτι να εισάγει το πρόγραμμα αποστέλει την είσοδο στο socket μέσω του `PrintWriter`:

```
String userInput;

while ((userInput = stdIn.readLine()) != null) {
    out.println(userInput);
    System.out.println("echo: " + in.readLine());
}
```

Σημειώστε ότι χρησιμοποιούμε δύο εντολές `readLine()`, μία για τη πρότυπη είσοδο από την κονσόλα (`stdIn`) και μία για την εκτύπωση από το κανάλι εισόδου του socket (`in`). Η δεύτερη `readLine()` περιμένει έως ότου ο server στείλει τα δεδομένα πίσω στον client. Όταν γίνει αυτό, τα δεδομένα θα τυπωθούν με το πρόθεμα "`echo: `".

Το πρόγραμμα θα σταματήσει να τρέχει όταν ο χρήστης στείλει κάποιο χαρακτήρα end-of-input. Τότε το loop θα σταματήσει να τρέχει και θα εκτελεστούν οι ακόλουθες εντολές:

```
out.close();
in.close();
stdIn.close();
echoSocket.close();
```

Αυτές οι εντολές είναι απαραίτητες για τη σωστή έξοδο του προγράμματος. Αρχικά κλείνουμε τους `PrintWriter` και `BufferedReader` που έχουμε συνδέσει στο socket, κατόπιν το `BufferedReader` που συνδέσαμε στην πρότυπη είσοδο και τέλος κλείνουμε το ίδιο το socket, τερματίζοντας τη σύνδεση με το server. Η σειρά με την οποία γίνεται κάτι τέτοιο έχει σημασία. Το socket πρέπει να κλείσει μόνο αφού έχουν κλείσει όλα τα streams που συνδέονται σε αυτό.

Το πρόγραμμα αυτό είναι αρκετά απλό για πρόγραμμα πελάτης, λόγω της απλότητας του πρωτοκόλλου echo. Συνήθως οι δικτυακές εφαρμογές είναι αρκετά πιο περίπλοκες και πρέπει να υλοποιούμε κάθε φορά και το server (εκτός αν γράφουμε μια εφαρμογή για κάποιο γνωστό πρωτόκολλο για το οποίο έχουμε ήδη το server). Ο server ως πρόγραμμα στηρίζεται στις ίδιες αρχές με τον client με τη διαφορά ότι θα πρέπει να έχει τη δυνατότητα (αναλόγως φυσικά το σχεδιασμό του) να απαντά σε πολλαπλές συνδέσεις ταυτόχρονα. Η διαδικασία γενικά είναι η εξής:

- Δημιούργησε ένα socket.
- Σύνδεσε ένα stream εισόδου και ένα stream εξόδου στο socket.
- Διάβασε και γράψε στα streams σύμφωνα με το πρωτόκολλο του server.
- Κλείσιμο των streams.
- Κλείσιμο των sockets.

11.Σύνδεση με βάσεις δεδομένων (JDBC)

Βάσεις δεδομένων και Java

Ένα από τα πιο δυνατά σημεία της Java είναι η διαχείριση των βάσεων δεδομένων κυρίως SQL (Oracle, DB2, MS SQL, MySQL, PostgreSQL, ODBC, κλπ). Ενώ οι περισσότερες από τις διαδεδομένες γλώσσες προγραμματισμού χαρακτηρίζονται από απελπιστική ανομοιομορφία στον προγραμματισμό των βάσεων δεδομένων (η κάθε βάση παρέχει το δικό της API, δηλαδή ξεχωριστό τρόπο προγραμματισμού) η Java έχει ορίσει ένα ενιαίο API το οποιο πρέπει να χρησιμοποιούν οι προγραμματιστές για να έχουν πρόσβαση στη βάση δεδομένων, ανεξαρτήτως ποια χρησιμοποιούν.

Δημιουργία σύνδεσης με τη βάση

Το πρώτο πράγμα που θα χρειαστεί να κάνετε είναι να δημιουργήσετε ένα δίαυλο επικοινωνίας, μία σύνδεση με τη βάση. Αυτό περιλαμβάνει δύο βήματα: α) φόρτωση του οδηγού (driver) που θα επικοινωνεί με τη βάση και β) δημιουργία της σύνδεσης.

Φόρτωση του οδηγού επικοινωνίας

Η φόρτωση του οδηγού είναι πολύ απλή και γίνεται με μία μόνο εντολή κώδικα. Για παράδειγμα, αν επιθυμείτε να χρησιμοποιήσετε τον ενδιάμεσο οδηγό JDBC-ODBC, αρκεί η επόμενη εντολή:

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

Θα πρέπει να κοιτάξετε τις οδηγίες χρήσης του οδηγού που πρόκειται να χρησιμοποιήσετε για να δείτε τον ακριβή τρόπο και το όνομα της κλάσης με τον οποίο πρέπει να το φορτώσετε. Για παράδειγμα, αν το όνομα της κλάσης ήταν `jdbc.DriverXYZ`, θα έπρεπε να χρησιμοποιήσετε την ακόλουθη εντολή για να τη φορτώσετε:

```
Class.forName("jdbc.DriverXYZ");
```

Είστε πλέον έτοιμοι να δημιουργήσετε μια σύνδεση με τη βάση δεδομένων.

Δημιουργία της σύνδεσης

Το δεύτερο βήμα είναι να δώσετε εντολή στον κατάλληλο οδηγό να πραγματοποιήσει σύνδεση με τη βάση δεδομένων, και η επόμενη εντολή επιτυγχάνει με γενικό τρόπο αυτό ακριβώς:

```
Connection con = DriverManager.getConnection(url,
                                 "myLogin", "myPassword");
```

Όπως βλέπετε και αυτή είναι αρκετά απλή διαδικασία, με μόνη δυσκολία το καθορισμό του `url`, δηλαδή της διεύθυνσης στην οποία βρίσκεται η βάση. Αν χρσιμοποιείτε τον οδηγό JDBC-ODBC, τότε το `url` θα αρχίζει με "`jdbc:odbc:`". Το υπόλοιπο τμήμα της διεύθυνσης συνήθως είναι η πηγή δεδομένων (DSN) ή το όνομα της βάσης. Έτσι, για παράδειγμα, αν θέλετε να προσπελάσσετε μια πηγή δεδομένων ODBC με το όνομα `myBase` θα πρέπει να χρησιμοποιήσετε ως `url` το "`jdbc:odbc:myBase`". Στη θέση του `myLogin` θα πρέπει να εισάγετε το όνομα του χρήστη με το οποίο συνδέεστε στη βάση και με `myPassword`, τον αντίστοιχο κωδικό πρόσβασης στην ίδια βάση, π.χ. έστω όνομα χρήστη "`Fred`" και κωδικός "`Fl1ntst0n3`". Έτσι η εντολή μετασχηματίζεται στις εξής:

```
String url = "jdbc:odbc:myBase";
Connection con = DriverManager.getConnection(url, "Fred", "Fl1ntst0n3");
```

Αν χρησιμοποιείτε έναν οδηγό JDBC από κάποιο άλλο κατασκευαστή (εκτός της Sun) θα πρέπει να ανατρέξετε στις αντίστοιχες οδηγίες, για το ποιο πρωτόκολλο σύνδεσης να χρησιμοποιήσετε, δηλαδή τί ακριβώς να βάλετε μετά το `jdbc:` στη διεύθυνση σύνδεσης (π.χ. αν το πρωτόκολλο σύνδεσης είναι `acme`, θα πρέπει να βάλετε ως `url` "`jdbc:acme`"). Το υπόλοιπο τμήμα της διεύθυνσης χρησιμοποιείται για τη δήλωση της πηγής δεδομένων.

Η μόνη μέθοδος της κλάσης `DriverManager` που θα χρειαστεί να χρησιμοποιήσετε, είναι η `getConnection()`, οι υπόλοιπες είναι απαραίτητες μόνο αν αποφασίσετε να γράψετε ένα οδηγό JDBC. Με αυτή τη μέθοδο δημιουργείται η σύνδεση με τη βάση η οποία μένει ανοιχτή για χρήση καθόλη τη διάρκεια του προγράμματος ή μέχρι να γίνει η αποσύνδεση. Η πρόσβαση στη βάση γίνεται μέσω του αντικειμένου `con`.

Δημιουργία πινάκων

Καταρχάς θα χρειαστεί να δημιουργήσουμε έναν πίνακα στη βάση μας. Έστω ότι ο πίνακας θα λέγεται `BIKES` και θα περιέχει τις απαραίτητες πληροφορίες για τις μοτοσυκλέτες που πουλήθηκαν την τελευταία εβδομάδα και συνολικά. Ακολουθεί η αρχική μορφή του πίνακα `BIKES`:

BIKE_NAME	*SUP_ID*	*PRICE*	*SALES*	*TOTAL*
Yamaha XT 500	101	5.000,00 €	0	0
KTM Adventurer 650	49	7.300,00 €	0	0
Kawasaki KLE 500	150	5.300,00 €	0	0
Honda TransAlp 650	101	7.900,00 €	0	0
Susuki VStrom 650	49	7.000,00 €	0	0
BMW FS 650	150	8.900,00 €	0	0

Η πρώτη στήλη κρατάει το όνομα της μοτοσυκλέτας (`BIKE_NAME`) και είναι τύπου `VARCHAR` (με μέγιστο μήκος 32 χαρακτήρες). Αυτή τη μεταβλητή θα χρησιμοποιήσουμε για το προσδιορισμό της γραμμής του πίνακα (που περιέχει τις αντίστοιχες πληροφορίες για τη συγκεκριμένη μοτοσυκλέτα) και θα είναι το κύριο κλειδί του πίνακα. Η δεύτερη στήλη, `SUP_ID`, θα περιέχει τον κωδικό του προμηθευτή και θα είναι τύπου `INTEGER`. Η τρίτη στήλη, `PRICE`, θα περιέχει την τιμή της κάθε μοτοσυκλέτας σε ευρώ και θα είναι τύπου `FLOAT`. Αν και στην SQL ορίζονται τύποι δεδομένων `NUMERIC` και `DECIMAL` που χρησιμοποιούνται για χρηματικές τιμές, δε θα τους χρησιμοποιήσουμε λόγω ασυμβατοτήτων μεταξύ των διαφορετιών βάσεων δεδομένων και για λόγους απλότητας. Οι τελευταίες στήλες, `SALES` και `TOTAL`, θα περιέχουν τον αριθμό των πωλήσεων την τρέχουσα εβδομάδα και συνολικά αντίστοιχα, και θα είναι επίσης τύπου `INTEGER`.

Θα δημιουργήσουμε και έναν ακόμη πίνακα των προμηθευτών, `SUPPLIERS`, που θα περιέχει τα ακόλουθα στοιχεία:

SUP_ID	*SUP_NAME*	*STREET*	*CITY*	*ZIP*
101	Ντόναλντ Ντακ	Οδός Γκαντεμιάς 313	Λιμνούπολη	13131
49	Μαύρος Πητ	Οδός Λούφας 7	Μίκυ Σίτυ	76767
150	Σκρουτζ ΜακΝτακ	Οδός Δολαρίων 1	Λιμνούπολη	10000

Και στους δύο πίνακες `BIKES` και `SUPPLIERS` δηλώνεται το πεδίο `SUP_ID`, που σημαίνει ότι μπορούν να συσχετιστούν οι πληροφορίες από τους δύο πίνακες σε μία εντολή `SELECT`. Το πεδίο `SUP_ID` δρα ως πρωτεύον κλειδί για τον πίνακα `SUPPLIERS` ώστε να προσδιορίζεται με μοναδικό τρόπο ο κάθε ένας από τους προμηθευτές. Στον πίνακα `BIKES` το ίδιο πεδίο καλείται *εξωτερικό κλειδί*, δηλαδή έχει εισαχθεί από άλλο πίνακα. Σε επόμενη ενότητα θα παρουσιάσουμε παραδείγματα χρήσης πρωτεύοντων και εξωτερικών κλειδιών στην εντολή `SELECT`.

Η ακόλουθη εντολή δημιουργεί τον πίνακα `BIKES` σύμφωνα με τον τρόπο που περιγράψαμε το κάθε πεδίο του πίνακα προηγουμένως.:

```
CREATE TABLE BIKES ( BIKE_NAME VARCHAR(32), SUP_ID INTEGER,
     PRICE FLOAT, SALES INTEGER, TOTAL INTEGER )
```

(Δεν είναι ανάγκη να γράψουμε το ερωτηματικό στο τέλος της εντολής, το JDBC το προσθέτει αυτόματα.)

Αυτή η εντολή για να εκτελεστεί μέσω του JDBC, θα πρέπει να την ορίσουμε ως αντικείμενο `String`. Επειδή στη Java (όπως και στις άλλες γλώσσες) δε μπορούμε να ορίσουμε ένα string που να επεκτείνεται σε περισσότερες από μία γραμμές, το χωρίζουμε σε μικρότερα τα οποίοα συνδέουμε με το '+'.:

```
String createTableBikes = "CREATE TABLE BIKES " +
    "(BIKE_NAME VARCHAR(32), SUP_ID INTEGER, PRICE FLOAT, " +
    "SALES INTEGER, TOTAL INTEGER)";
```

Δημιουργία εντολών με JDBC

Για την εκτέλεση των εντολών SQL μέσω του JDBC χρησιμοποιούμε τα αντικείμενα `Statement`. Το αντικείμενο `Statement` αποστέλει την εντολή SQL στη βάση δεδομένων και την εκτελεί, επιστρέφοντας τα αποτελέσματα με συγκεκριμένη μορφή που διευκολύνει την επεξεργασία τους. Υπάρχουν δύο μέθοδοι εκτέλεσης της εντολής με το `Statement`, η `executeQuery()` που χρησιμοποιείται για εντολές `SELECT` και η `executeUpdate()` που χρησιμοποιείται για εντολές που δημιουργούν ή μεταβάλλουν τα περιεχόμενα των πινάκων. Ο τρόπος χρήσης του `Statement` είναι ο εξής:

```
Statement stmt = con.createStatement();
stmt.executeUpdate("CREATE TABLE BIKES ( BIKE_NAME VARCHAR(32), SUP_ID"
     + "INTEGER, PRICE FLOAT, SALES INTEGER, TOTAL INTEGER)");
```

Δηλαδή, αρχικά δημιουργούμε το αντικείμενο `stmt` και του περνάμε την εντολή προς εκτέλεση με τη μέθοδο `executeUpdate()`. Επειδή έχουμε ήδη ορίσει ένα αντικείμενο `String` που περιέχει την εντολή SQL που δημιουργεί τον πίνακα `BIKES`, μπορούμε εναλλακτικά να εκτελέσουμε το εξής:

```
stmt.executeUpdate(createTableBikes);
```

Εισαγωγή δεδομένων σε πίνακα

Μέχρι αυτό το σημείο απλώς δημιουργήσαμε τον πίνακα BIKES χωρίς να του προσθέσουμε εγγραφές. Η πρόσθεση θα γίνει με τη μέθοδο executeUpdate() στην οποία τώρα θα εισάγουμε ως παράμετρο την αντίστοιχη εντολή SQL:

```
Statement stmt = con.createStatement();
stmt.executeUpdate (  "INSERT INTO BIKES " +
                      "VALUES ('Yamaha XT 500', 101, 5000, 0, 0)");
```

Οι υπόλοιπες εγγραφές μπορούν να προστεθούν στον πίνακα BIKES με τον ίδιο τρόπο. Σημειωτέον ότι δε δημιουργούμε άλλο αντικείμενο Statement αλλά χρησιμοποιούμε το ίδιο.

```
stmt.executeUpdate ( "INSERT INTO BIKES " +
                      "VALUES ('KTM Adventurer 650', 150, 7300, 0, 0)");
stmt.executeUpdate ( "INSERT INTO BIKES " +
                      "VALUES ('Kawasaki KLE 500', 150, 5300, 0, 0)");
stmt.executeUpdate ( "INSERT INTO BIKES " +
                      "VALUES ('Honda TransAlp 650', 150, 7900, 0, 0)");
stmt.executeUpdate ( "INSERT INTO BIKES " +
                      "VALUES ('Susuki VStrom 650', 150, 7000, 0, 0)");
stmt.executeUpdate ( "INSERT INTO BIKES " +
                      "VALUES ('BMW FS 650', 150, 8900, 0, 0)");
```

Με αντίστοιχο τρόπο εισάγουμε δεδομένα και στον πίνακα SUPPLIERS.

Ανάκτηση δεδομένων από πίνακα

Οι πίνακές μας πλέον έχουν κάποια δεδομένα τα οποία μπορούμε να ανακτήσουμε χρησιμοποιώντας τη SELECT. Για παράδειγμα, αν εκτελέσουμε στο κέλυφος της γλώσσας SQL την εντολή:

```
SELECT * FROM BIKES
```

αυτή θα επιστρέψει το αποτέλεσμα:

```
BIKE_NAME               SUP_ID  PRICE   SALES   TOTAL
---------------         ------  -------  -----   -----
Yamaha XT 500           101     5000,00    0       0
KTM Adventurer 650       49     7300,00    0       0
Kawasaki KLE 500        150     5300,00    0       0
Honda TransAlp 650      101     7900,00    0       0
Susuki VStrom 650        49     7000,00    0       0
BMW FS 650              150     8900,00    0       0
```

Και με παρόμοιο τρόπο, η εκτέλεση της εντολής:

```
SELECT BIKE_NAME, PRICE FROM BIKES
```

θα επιστρέψει το εξής αποτέλεσμα:

```
BIKE_NAME               PRICE
---------------         -------
Yamaha XT 500           5000,00
KTM Adventurer 650      7300,00
Kawasaki KLE 500        5300,00
```

```
Honda TransAlp 650      7900,00
Susuki VStrom 650       7000,00
BMW FS 650              8900,00
```

Ας υποθέσουμε ότι θέλουμε να δούμε μόνο ποιες μοτοσυκλέτες έχουν τιμή μικρότερη από 7100 €:

```
SELECT BIKE_NAME, PRICE FROM BIKES WHERE PRICE < 7100
```

The results would look similar to this:

```
BIKE_NAME               PRICE
---------------         -------
Yamaha XT 500           5000,00
Kawasaki KLE 500        5300,00
Susuki VStrom 650       7000,00
```

Αυτές είναι απλώς οι εντολές SQL και τα αντίστοιχα αποτελέσματά τους, αν τις εκτελέσουμε σε κάποιο κέλυφος SQL. Για να έχουμε αυτά τα αποτελέσματα στη Java θα πρέπει να χρησιμοποιήσουμε την κλάση `ResultSet` του JDBC.

Ανάκτηση τιμών από τα ResultSet

Όπως ήδη αναφέραμε, οι εντολές που μεταβάλλουν τα περιεχόμενα ενός πίνακα εκτελούνται με τη χρήση της `executeUpdate()` ενώ οι εντολές που επιστρέφουν κάποιο αποτέλεσμα, όπως η `SELECT`, εκτελούνται με τη μέθοδο `executeQuery()`. Η `executeQuery()` επιστρέφει τα αποτελέσματα ομαδοποιημένα σε ένα αντικείμενο της κλάσης `ResultSet`. Ο τρόπος χρήσης της `executeQuery()` είναι αρκετά απλώς όπως φαίνεται και στο ακόλουθο παράδειγμα:

```
ResultSet rs = stmt.executeQuery("SELECT BIKE_NAME, PRICE FROM BIKES");
```

Η μεταβλητή `rs` περιέχει το αποτέλεσμα της εντολής `SELECT`, το οποίο μπορούμε να επεξεργαστούμε. Η επεξεργασία όμως γίνεται γραμμή προς γραμμή και θα πρέπει να μπορούμε να μεταφερόμαστε με κάποιο τρόπο. Το πρότυπο JDBC 1.0 επιτρέπει μόνο την μετακίνηση προς τα κάτω (δηλαδή από την πρώτη γραμμή στις επόμενες), ενώ με το JDBC 2.0 και 3.0 είναι πλέον δυνατή η μετακίνηση σε όλο το πλήθος των γραμμών των αποτελεσμάτων. Για λόγους απλότητας, θα αναφέρουμε μόνο το πρότυπο 1.0 και θα δώσουμε παραπομπές για τις επόμενες εκδόσεις.

Χρήση της μεθόδου next

Η μετακίνηση σε επόμενες γραμμές γίνεται με τη μέθοδο `next()` του `ResultSet`. Με την εκτέλεση της μεθόδου `next()`, μετακινούμε έναν εικονικό δρομέα (cursor) στα αποτελέσματα του πίνακα και κάθε φορά επεξεργαζόμαστε τη γραμμή στην οποία βρίσκεται ο δρομέας (που καλείται και τρέχουσα γραμμή). Η αρχική τρέχουσα γραμμή είναι *πάνω* από την πρώτη γραμμή των αποτελεσμάτων που επιστρέφονται με το `ResultSet` (που με τη σειρά τους εξαρτώνται από την εντολή SQL που εκτελούμε με την `executeQuery()`). Έτσι ακόμη και αν το αποτέλεσμα `ResultSet` περιέχει μόνο μια γραμμή, πάλι θα πρέπει να εκτελέσουμε μια φορά τη μέθοδο `next()` για αυτό το `ResultSet`.

Χρήση των μεθόδων getXXX

Από την τρέχουσα γραμμή, μπορούμε να εκλάβουμε τις τιμές της γραμμής που αντιστοιχούν στα πεδία του πίνακα. Επειδή οι τιμές αυτές είναι διαφόρων τύπων η κάθε μία, αναλόγως πώς δημιουργήσαμε τον πίνακα με την CREATE TABLE, θα πρέπει να τις αντιστοιχίσουμε σε μεταβλητές αντίστοιχου τύπου στη Java. Κάτι τέτοιο μπορεί να είναι προβληματικό ειδικά για τύπους που απαιτούν κάποιου είδους μετατροπή (π.χ. χαρακτήρες σε κάποιο encoding, ISO-8859-7, ή χρονικές σφραγίδες TIMESTAMP). Η Java προσφέρει κάποιες μεθόδους που πραγματοποιούν αυτές τις μετατροπές και επιστρέφουν τη σωστή τιμή για το ζητούμενο πεδίο (πάντα για την τρέχουσα γραμμή).

Για παράδειγμα, για να εκλάβουμε την τιμή του πεδίου `BIKE_NAME` στη προηγούμενη εντολή `SELECT` (που είναι τύπου `VARCHAR`), χρησιμοποιούμε τη μέθοδο `getString()` του `ResultSet`, ενώ για τη τιμή του πεδίου `PRICE` χρησιμοποιούμε τη `getFloat()`. Στο επόμενο παράδειγμα φαίνεται η χρήση της `executeQuery()`, του `ResultSet`, της μεθόδου `next()`, καθώς και των δύο μεθόδων `getXXX()` που αναφέραμε. Όλη η διαδικασία ανάκτησης των τιμών εκτελείται για κάθε γραμμή των αποτελεσμάτων, έως ότου έχουμε επεξεργαστεί όλα τα δεδομένα.

```
String query = "SELECT BIKE_NAME, PRICE FROM BIKES";
ResultSet rs = stmt.executeQuery(query);
while (rs.next()) {
    String s = rs.getString("BIKE_NAME");
    float n = rs.getFloat("PRICE");
    System.out.println(s + "   " + n);
}
```

Το αποτέλεσμα του παραπάνω κώδικα θα είναι το εξής:

```
Yamaha XT 500   5000,0
KTM Adventurer 650   7300,0
Kawasaki KLE 500   5300,0
Honda TransAlp 650   7900,0
Susuki VStrom 650   7000,0
BMW FS 650   8900,0
```

Το JDBC προσφέρει δύο τρόπους να χρησιμοποιήσουμε τις μεθόδους getXXX() και να εκλάβουμε την τιμή ενός πεδίου. Ο πρώτος είναι να δώσουμε ως παράμετρο το όνομα του πεδίου και ο δεύτερος είναι δίνοντας τον αύξοντα αριθμό του πεδίου, με τη σειρά που έχει στο αποτέλεσμα που επιστρέφεται στο ResultSet. Το πρώτο πεδίο έχει αύξοντα αριθμό 1, το δεύτερο πεδίο έχει αύξοντα αριθμό 2, και ούτω καθεξής. Στον παραπάνω κώδικα μπορούμε να επιτύχουμε το ίδιο αποτέλεσμα με τις εξής εντολές:

```
String s = rs.getString(1);
float n = rs.getFloat(2);
```

Η διαφορά είναι πιθανόν μια μικρή βελτίωση στην απόδοση όταν χρησιμοποιούμε αριθμητικές παραμέτρους, αλλά οπωσδήποτε είναι πιο εύκολο στην κατανόηση όταν χρησιμοποιείται το όνομα του πεδίου.

Το JDBC προσφέρει αρκετές μεθόδους για μετατροπές από αντίστοιχους τύπους SQL. Οι πιο σημαντικές από αυτές φαίνονται στον ακόλουθο πίνακα:

Όνομα μεθόδου	*Συνιστώμενοι τύποι SQL*	*Άλλοι συμβατοί τύποι*	*Επιστρέφει*
`getByte()`	`TINYINT`	`SMALLINT, INTEGER, BIGINT, REAL, FLOAT, DOUBLE, DECIMAL, NUMERIC, BIT, CHAR, VARCHAR, LONGVARCHAR`	`byte`
`getShort()`	`SMALLINT`	`TINYINT, INTEGER, BIGINT, REAL, FLOAT, DOUBLE, DECIMAL, NUMERIC, BIT, CHAR, VARCHAR, LONGVARCHAR`	`short`
`getInt()`	`INTEGER`	`TINYINT, SMALLINT, BIGINT, REAL, FLOAT, DOUBLE, DECIMAL, NUMERIC, BIT, CHAR, VARCHAR, LONGVARCHAR`	`int`
`getLong()`	`BIGINT`	`TINYINT, SMALLINT, INTEGER, REAL, FLOAT, DOUBLE, DECIMAL, NUMERIC, BIT, CHAR, VARCHAR, LONGVARCHAR`	`long`
`getFloat()`	`REAL`	`TINYINT, SMALLINT, INTEGER, BIGINT, FLOAT, DOUBLE, DECIMAL, NUMERIC, BIT, CHAR, VARCHAR, LONGVARCHAR`	`float`
`getDouble()`	`FLOAT, DOUBLE`	`TINYINT, SMALLINT, INTEGER, BIGINT, REAL, DECIMAL, NUMERIC, BIT, CHAR, VARCHAR, LONGVARCHAR`	`double`
`getBoolean()`	`BIT`	`TINYINT, SMALLINT, INTEGER, BIGINT, REAL, FLOAT, DOUBLE, DECIMAL, NUMERIC, CHAR, VARCHAR, LONGVARCHAR`	`boolean`
`getString()`	`CHAR, VARCHAR`	`TINYINT, SMALLINT, INTEGER, BIGINT, REAL, FLOAT, DOUBLE, DECIMAL, NUMERIC, BIT, LONGVARCHAR, BINARY, VARBINARY, LONGVARBINARY, DATE, TIME, TIMESTAMP`	`String`
`getDate()`	`DATE`	`CHAR, VARCHAR, LONGVARCHAR, TIMESTAMP`	`Date`
`getTime()`	`TIME`	`CHAR, VARCHAR, LONGVARCHAR, TIMESTAMP`	`Time`
`getTimestamp()`	`TIMESTAMP`	`CHAR, VARCHAR, LONGVARCHAR, DATE, TIME`	`Timestamp`

Η μέθοδος getString

Η συγκεκριμένη μέθοδος είναι πολύ χρήσιμη γιατί μπορεί να χρησιμοποιηθεί για να εκλάβει την τιμή οποιουδήποτε τύπου δεδομένων SQL (εκτός από τους νεώτερους τύπους SQL3). Αυτό είναι χρήσιμο ειδικά όταν θέλουμε να χρησιμοποιήσουμε αριθμητικές (ή άλλες) τιμές ως αντικείμενα `String`, ή ακόμη και τιμές από πεδία των οποίοων τους τύπους δε γνωρίζουμε. Γενικά μπορεί να αποδειχθεί αρκετά χρήσιμο εργαλείο.

Ανανέωση πινάκων

Έστω ότι την πρώτη εβδομάδα γίνονται 7 πωλήσεις της μοτοσυκλέτας, Susuki VStrom 650, και 4 πωλήσεις της BMW FS 650. Τα αντίστοιχα πεδία SALES και TOTAL του πίνακα BIKES θα πρέπει να ανανεωθούν χρησιμοποιώντας την εντολή UPDATE της SQL. Η εντολές που πρέπει να εκτελέσουμε είναι οι εξής:

```
String updateString = "UPDATE BIKES SET SALES = 7 " +
        "WHERE BIKE_NAME LIKE 'Susuki VStrom 650'";
stmt.executeUpdate(updateString);
```

για την VStrom, και

```
String updateString = "UPDATE BIKES SET SALES = 4 " +
        "WHERE BIKE_NAME LIKE 'BMW FS 650'";
stmt.executeUpdate(updateString);
```

για την BMW.

Ο πίνακας BIKES θα έχει τώρα τη μορφή:

```
BIKE_NAME               SUP_ID  PRICE    SALES   TOTAL
--------------          ------  -------   -----   -----
Yamaha XT 500           101     5000,00    0       0
KTM Adventurer 650       49     7300,00    0       0
Kawasaki KLE 500        150     5300,00    0       0
Honda TransAlp 650      101     7900,00    0       0
Susuki VStrom 650        49     7000,00    7       0
BMW FS 650              150     8900,00    4       0
```

Δεν έχουμε όμως ανανεώσει το πεδίο TOTAL και έχει ακόμη την τιμή μηδέν. Αυτό επιτυγχάνεται με την εντολή:

```
String updateString = "UPDATE BIKES SET TOTAL = TOTAL + 75";
stmt.executeUpdate(updateString);
```

Αυτή η εντολή θα ανανεώσει τις τιμές της TOTAL όλων των γραμμών (αλλά στην πραγματικότητα το αποτέλεσμα θα αλλάξει μόνο για τις μοτοσυκλέτες Susuki VStrom και BMW FS).

Αν θέλουμε να δούμε τώρα ποιες μηχανές έχουν πωληθεί αυτήν την εβδομάδα μπορούμε να εκτελέσουμε τις ακόλουθες εντολές:

```
String query = "SELECT BIKE_NAME, SALES, TOTAL FROM BIKES " +
               "WHERE SALES > 0";
ResultSet rs = stmt.executeQuery(query);
while (rs.next()) {
   String name = rs.getString("BIKE_NAME");
   int sold = rs.getInt("SALES");
   int tot  = rs.getInt("TOTAL");
   System.out.println(name+": Sold " + sold +
                      " bikes this week, " + tot + " in total");
}
```

Το πρόγραμμα θα τυπώσει:

```
Susuki VStrom 650: Sold 7 bikes this week, 7 in total
BMW FS 650: Sold 4 bikes this week, 4 in total
```

12.Τi; Τελείωσε;

Δυστυχώς, ναι. Η Java είναι μια γλώσσα προγραμματισμού με πολύ βάθος, η οποία συνεχώς εξελίσσεται. Προς το παρόν, το μόνο που κάναμε είναι να αγγίξουμε την επιφάνεια της και να δείξουμε μερικές από τις δυνατότητές της. Για παράδειγμα κάναμε αναφορά σε Applets, σε δημιουργία εφαρμογών με γραφικά περιβάλλοντα σε Swing ή SWT, σε JavaBeans, σε Servlets, στο Java3D, στον ήχο μέσα από Java, κλπ. Η λίστα είναι τεράστια και θα ήθελε εκατοντάδες βιβλία πολλαπλάσιου μεγέθους από αυτόν το μικρό οδηγό.

Για τη συνέχεια, προτείνουμε τα βιβλία και τη διεύθυνση URL που αναγράφονται στο τέλος. Τα ίδια έχουν αρκετή βιβλιογραφία από μόνα τους. Γενικά το μόνο που χρειάζεται κάποιος για να γίνει ειδικός είναι χρόνος και όρεξη, τα υπόλοιπα θα έρθουν μόνα τους.

Βιβλιογραφία

Εισαγωγικά:

- Εισαγωγή Στην Java 2, ΛΙΑΚΕΑΣ ΓΙΩΡΓΟΣ, ΚΛΕΙΔΑΡΙΘΜΟΣ, ISBN 960209625X
- Java 2: A beginner's Guide, Schildt, Osborne, ISBN 0072225882

Προχωρημένα:

- JAVA 2 Η ΒΙΒΛΟΣ, WALSH, ΓΚΙΟΥΡΔΑΣ Μ., ISBN 960512310X
- Thinking in Java, Bruce Eckel, Prentish Hall, ISBN 0131002872

Online:

http://java.sun.com/docs/books/tutorial/