ΕΘΝΙΚΟ ΜΕΤΣΟΒΕΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ Η/Υ
ΤΟΜΕΑΣ Επικοινωνιών, Ηλεκτρονικής και Συστημάτων Πληροφορικής

ΕΙΣΑΓΩΓΗ ΣΤΗ ΓΛΩΣΣΑ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ JAVA

ΕΡΓΑΣΤΗΡΙΟ ΠΟΛΥΜΕΣΩΝ

ΠΕΡΙΕΧΟΜΕΝΑ

1. Εισαγωγή

Η ραγδαία εξάπλωση του Internet και του World-Wide Web δημιούργησαν την ανάγκη νέων τρόπων ανάπτυξης και διανομής του λογισμικού. Οι απαιτήσεις αυτές οδήγησαν στην δημιουργία της γλώσσας προγραμματισμού Java, από την εταιρία Sun microsystems ™. Η Java σχεδιάστηκε με σκοπό την ανάπτυξη εφαρμογών που θα τρέχουν σε ετερογενή δικτυακά περιβάλλοντα.

Η Java έχει τα ακόλουθα χαρακτηριστικά

- Αντικειμενοστραφής (ομοιότητες εντολών με τη C++).
- Δημιουργία ανεξάρτητων εφαρμογών και applets (applet = προγράμματα που περιλαμβάνονται σε HTML σελίδες και εκτελούνται από τον Web Browser).
- Είναι Interpreted γλώσσα. Αυτό σημαίνει ότι ο java compiler δεν παράγει εκτελέσιμο κώδικα αλλά μια μορφή ψευδοκώδικα (bytecode) το οποίο από μόνο του δεν τρέχει σε καμία μηχανή. Προκειμένου λοιπόν να εκτελεστεί απαιτείται η χρήση ενός interpreter (=διερμηνέα) για να μετατρέψει το bytecode σε πραγματικό εκτελέσιμο κώδικα. Αυτό το χαρακτηριστικό δίνει τη δυνατότητα στα java bytecodes να μπορούν να τρέξουν σε οποιοδήποτε μηχάνημα, κάτω από οποιοδήποτε λειτουργικό, αρκεί να έχει εγκατασταθεί ένας java interpreter. Επίσης ένα άλλο χαρακτηριστικό του java bytecode είναι το μικρό του μέγεθος, (μόλις λίγα Kilobytes). Αυτό το κάνει ιδανικό για μετάδοση μέσω του δικτύου.
- Κατανεμημένη (distributed). Δηλαδή ένα πρόγραμμα σε Java είναι δυνατό να το φέρουμε από το δίκτυο και να το τρέξουμε. Επίσης είναι δυνατό διαφορετικά κομμάτια του προγράμματος να έρθουν από διαφορετικά sites.
- Ασφαλής (secure). Στο δίκτυο όμως ελλοχεύουν πολλοί κίνδυνοι για τον χρήστη - παραλήπτη μιας δικτυακής εφαρμογής, γι' αυτό η Java έχει σχεδιαστεί έτσι ώστε να ελαχιστοποιείται η πιθανότητα προσβολής του συστήματος του χρήστη από κάποιο applet γραμμένο για τέτοιο σκοπό.
- Είναι multithreaded. Η Java υποστηρίζει εγγενώς την χρήση πολλών threads. Προκειμένου να το επιτύχει αυτό σε συστήματα με έναν επεξεργαστή, το Java runtime system (interpreter) υλοποιεί ένα δικό χρονοδρομολογητή (scheduler), ενώ σε συστήματα που υποστηρίζουν πολυεπεξεργασία η δημιουργία των threads ανατίθεται στο λειτουργικό σύστημα. Φυσικά όλα αυτά είναι αόρατα τόσο στον προγραμματιστή όσο και στον χρήστη.
- Υποστηρίζει multimedia εφαρμογές. Με αυτό εννοούμε ότι η Java παρέχει ευκολίες στη δημιουργία multimedia εφαρμογών. Αυτό επιτυγχάνεται τόσο με την ευελιξίας της σαν γλώσσα όσο και με τις πλούσιες και συνεχώς εμπλουτιζόμενες βιβλιοθήκες της.

1.1 Τα εργαλεία της Java

Ακολούθως παρουσιάζονται εν συντομία όλα τα εργαλεία που έρχονται με το Java Java 2 Platform Standard Edition (J2SE), της Sun microsystems. Η παρούσα έκδοση της Java και του J2SE είναι η Java 2 έκδοση 5.0.

- **javac** Είναι ο compiler της Java. Η χρήση του στο command-line είναι : *javac <όνομα αρχείου>*. Εδώ να σημειώσουμε ότι το javac δεν παράγει ένα αρχείο με όλον τον κώδικα, αλλά χωριστό αρχείο για κάθε κλάση. Τα αρχεία των κλάσεων ονομάζονται : *<όνομα κλάσης>.class.*
- **java** Είναι ο interpreter της Java. Η χρήση του είναι η εξής : *java <κλάση>, πχ java myClass και όχι java myClass.class*.
- **javaw** (MONO στα Windows 95/NT) Είναι παρόμοιο με το java με μόνη την διαφορά ότι δεν χρειάζεται shell για να τρέξει.
- **jdb** Είναι ο Java debuger.
- **javah** Κατασκευάζει C files και stub files για κάποια κλάση. Αυτά τα αρχεία είναι απαραίτητα όταν θέλουμε να υλοποιήσουμε κάποιες από τις μεθόδους της κλάσης σε C, πράγμα πολύ σπάνιο.
- **javap** Είναι ο Java disassembler.
- **javadoc** Είναι ένα πρόγραμμα για αυτόματη κατασκευή documentation. Είναι αρκετά χρήσιμο στην κατασκευή βοηθημάτων και τεχνικών αναφορών για εφαρμογές οποιουδήποτε μεγέθους.

- **appletviewer** Είναι ένα πρόγραμμα το οποίο μας επιτρέπει να τρέχουμε και να χρησιμοποιούμε τα διάφορα applets σε Java. Οι stand-alone εφαρμογές, ωστόσο, δεν τρέχουν στον appletviewer αλλά κατευθείαν στον java ή javaw.

1.2 Δημιουργία μιας Java εφαρμογής

Με τα ακόλουθα βήματα μπορεί να κατασκευάζεται μια ανεξάρτητη Java εφαρμογή (stand-alone application).
Δημιουργήστε το αρχείο πηγαίου κώδικα HelloWorldApp.java που να περιέχει τις ακόλουθες εντολές:

```
/**
 * The HelloWorldApp class implements an application that
 * simply displays "Hello World!" to the standard output.
 */
class HelloWorldApp {
    public static void main(String[] args) {
        System.out.println("Hello World!"); //Display the string.
    }
}
```

Κάντε το Compile χρησιμοποιώντας τον Java compiler:
javac HelloWorldApp.java

Ο compiler θα δημιουργήσει το αρχείο HelloWorldApp.class

Εκτελέστε την εφαρμογή καλώντας τον Java interpreter:
java HelloWorldApp

Θα δείτε στην οθόνη σας το μήνυμα "Hello World!"

1.3 Δημιουργία ενός Java Applet

Με τα ακόλουθα βήματα μπορεί να κατασκευάζεται μια Java applet.

Δημιουργήστε το αρχείο πηγαίου κώδικα HelloWorld.java που να περιέχει τις ακόλουθες εντολές:

```
import java.applet.Applet;
import java.awt.Graphics;

public class HelloWorld extends Applet {
    public void paint(Graphics g) {
        g.drawString("Hello world!", 50, 25);
    }
}
```

Κάντε το Compile χρησιμοποιώντας τον Java compiler:
javac HelloWorld.java

Ο compiler θα δημιουργήσει το αρχείο HelloWorld.class

Δημιουργήστε το HTML αρχείο Hello.html που θα καλεί το Applet. Το αρχείο αυτό θα πρέπει να βρίσκεται στο ίδιο directory με το αρχείο HelloWorldApp.class.

```
<HTML>
<HEAD>
<TITLE> A Simple Program </TITLE>
</HEAD>
<BODY>

Here is the output of my program:
<APPLET CODE="HelloWorld.class" WIDTH=150 HEIGHT=25>
</APPLET>
</BODY>
</HTML>
```

Φορτώστε το HTML αρχείο σε κάποιον Java-enabled browser πχ τον Internet Explorer και θα δείτε το αποτέλεσμα "Hello World!" μέσα από τον browser.

1.4 Χρήσιμες διευθύνσεις Internet για την Java

Κλείνοντας το κεφάλαιο αυτό παραθέτουμε κάποιες διευθύνσεις του Internet που παρέχουν πληροφορίες και χρήσιμες οδηγίες για τη γλώσσα προγραμματισμού Java.

Java Tutorial:

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

Περιέχει αναλυτικό εκπαιδευτικό υλικό και ενσωματωμένα παραδείγματα για όλα τα θέματα της γλώσσας. Μπορείτε να αναφέρεστε σε αυτό όποτε χρειάζεται να αντλήσετε περισσότερες πληροφορίες σχετικά με κάποιο θέμα που αναφέρουν οι σημειώσεις.

Java API :

http://java.sun.com/j2se/1.5/docs/api/index.html

Πολύ χρήσιμη διεύθυνση που είναι σκόπιμο να την έχετε ανοικτή όταν προγραμματίζεται σε Java. Περιέχει το **Application Programming Interface της Java (Java API)** δηλαδή όλες της κλάσεις που διαθέτει η Java ομαδοποιημένες ανάλογα με τις λειτουργίες που προσφέρουν σε ενότητες (πακέτα), την περιγραφή της λειτουργίας κάθε κλάσης καθώς και των μεθόδους της.

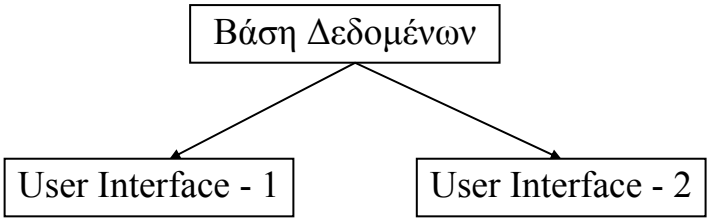
Java Platform:

http://java.sun.com/j2se/1.5.0/download.jsp

Η Java πλατφόρμα με τα εργαλεία που αναφέρθηκαν στην παράγραφο 1.1. Είναι το εκτελέσιμο πρόγραμμα που πρέπει να εγκαταστήσετε στον υπολογιστή σας ώστε να είστε σε θέση να προγραμματίζεται σε Java. Το συγκεκριμένο εκτελέσιμο σας εγκαθιστά και το JRE (Java Runtime Environment) στον Web Browser του υπολογιστή σας εάν αυτό δεν έχει ήδη εγκατασταθεί. Το JRE δίνει τη δυνατότητα στο Web Browser σας να εκτελεί java applets.

2. Εισαγωγή στον αντικειμενοστραφή προγραμματισμό

Ο αντικειμενοστραφής προγραμματισμός, (OOP - Object Oriented Programming), είναι μια προγραμματιστική φιλοσοφία όπως και ο προστακτικός ή ο λογικός προγραμματισμός. Σύμφωνα με τον OOP ένα πρόγραμμα ΔΕΝ αποτελείται από τα δεδομένα και τον κώδικα που τα επεξεργάζεται αλλά από *αντικείμενα (objects)* τα οποία εμπεριέχουν τα δεδομένα και τα οποία (αντικείμενα) ανταλλάσσουν μεταξύ τους πληροφορίες και μηνύματα προκειμένου να επιτευχθεί ο στόχος του προγράμματος. Για παράδειγμα έστω ότι έχουμε ένα σύστημα που περιλαμβάνει μια βάση δεδομένων καθώς και user interfaces με τα οποία οι χρήστες επικοινωνούν και αλληλεπιδρούν με την Β.Δ..



Τα UIs στέλνουν τις εντολές των χρηστών στην ΒΔ και παρουσιάζουν στην οθόνη τις απαντήσεις που λαμβάνουν. Δηλαδή όλο το πρόγραμμα αποτελείται από ένα αντικείμενο ΒΔ και 2 αντικείμενα UI τα οποία είναι ολόιδια (αλλά εξυπηρετούν άλλους χρήστες). Συνεπώς ο κώδικας που θα γράφαμε θα περιείχε τον ορισμό ενός αντικειμένου ΒΔ, τον ορισμό ενός αντικειμένου UI και την κατασκευή ενός ΒΔ και 2 UI αντικειμένων.

Ορολογία

Ο κώδικας που ορίζει ένα αντικείμενο λέγεται **κλάση** (class) του αντικειμένου αυτού. Η κλάση χρησιμοποιείται σαν *μήτρα* για την κατασκευή πανομοιότυπων αντιγράφων - αντικειμένων. Τα αντικείμενα - αντίγραφα, λέγονται στιγμιότυπα (instances) της κλάσης αυτής και η κατασκευή και αρχικοποίηση ενός από αυτά λέγεται instantiation.

Σημείωση

Το αντικείμενο αυτό καθ' εαυτό είναι μια οντότητα στη μνήμη η οποία περιέχει δεδομένα καθώς και μεθόδους μέσω των οποίων μπορούμε να αλλάξουμε τα δεδομένα ή να επικοινωνήσουμε με το αντικείμενο. Αντίθετα η κλάση είναι απλώς ένα πρότυπο για την δημιουργία αντιγράφων του ίδιου αντικειμένου. Είναι δηλαδή ότι είναι ο Τύπος Δεδομένων για τις μεταβλητές (στον δομημένο προγραμματισμό).

2.1 Δομή ενός προγράμματος σε JAVA

< ΕΙΣΑΓΩΓΗ ΕΤΟΙΜΩΝ ΚΛΑΣΕΩΝ ΑΠΟ ΤΗ ΒΙΒΛΙΟΘΗΚΗ >
<ΟΡΙΣΜΟΣ ΝΕΑΣ ΚΛΑΣΗΣ>
<ΟΡΙΣΜΟΣ ΝΕΑΣ ΚΛΑΣΗΣ>
κ.ο.κ.

Μία από όλες τις κλάσεις θα παίξει το ρόλο της αφετηρίας του προγράμματος, με άλλα λόγια θα είναι σαν την main() της C. Αναλυτικότερα, θα αναλάβει να κατασκευάσει τα αντικείμενα που χρειάζεται το πρόγραμμα, με τη χρήση των κλάσεων που ορίσαμε ή φέραμε από τη βιβλιοθήκη. Φυσικά την αρχική κλάση θα την υποδείξουμε εμείς όταν θα ξεκινήσουμε τον interpreter της γλώσσας.

ΣΗΜΕΙΩΣΗ: Η παραπάνω επεξήγηση δεν είναι απολύτως ακριβής ωστόσο διευκολύνει σημαντικά την κατανόηση της δομής ενός προγράμματος JAVA.

2.2 Μια συζήτηση σχετικά με τα αντικείμενα στη JAVA

Όπως ειπώθηκε και παραπάνω τα αντικείμενα περιέχουν δεδομένα και παρέχουν μεθόδους για την επεξεργασία των δεδομένων αυτών καθώς και για την επικοινωνία με άλλα αντικείμενα. Ας εμβαθύνουμε λίγο περισσότερο σε αυτές τις έννοιες.

Κατ` αρχήν τα δεδομένα μπορεί να είναι άλλα αντικείμενα που περιέχονται μέσα σε ένα άλλο αντικείμενο ή μπορεί να είναι κοινές μεταβλητές όπως τις γνωρίζουμε από την C και την PASCAL. Τις μεταβλητές και τα αντικείμενα αυτά τα καλούμε *πεδία ή instance variables* του αντικειμένου.

Σε αυτό το σημείο να παρατηρήσουμε ότι μία κλάση μοιάζει με ένα structure της C το οποίο μπορεί να περιέχει κοινές μεταβλητές ή μεταβλητές που προέκυψαν από άλλα structures.

Επιπλέον μία κλάση (ή ένα αντικείμενο) περιέχει και μεθόδους για την επικοινωνία του με τον *έξω κόσμο*, δηλαδή τα άλλα αντικείμενα.

Οι μέθοδοι αυτοί υλοποιούνται σαν συναρτήσεις παρόμοιες με αυτές της C. Όταν λοιπόν κάποιος θέλει να ζητήσει κάτι από ένα αντικείμενο, (ή εναλλακτικά να στείλει ένα μήνυμα - αίτημα), δεν έχει παρά να καλέσει - εκτελέσει την αντίστοιχη μέθοδο του αντικειμένου. Αυτό γίνεται πολύ απλά ως εξής :

<Αντικείμενο>.<μέθοδος>(<Λίστα παραμέτρων>);

Π.χ.

mycar.startEngine();

Αυτό το σχήμα μας θυμίζει πολύ τον τρόπο πρόσβασης σε μία μεταβλητή που περιέχεται σε ένα structure της C. Επίσης να σημειώσουμε ότι μια μέθοδος μπορεί να καλεί άλλες μεθόδους του ίδιου αντικειμένου ή να επεξεργάζεται τα πεδία του. Φυσικά μπορεί να καλεί και μεθόδους ξένων αντικειμένων εφόσον έχει κάποιον δείκτη σε αυτά.

ΣΗΜΕΙΩΣΗ

Εδώ είδαμε ένα πολύ σπουδαίο χαρακτηριστικό του OOP. Αυτό είναι το να μπορεί να κρατά τα δεδομένα του κρυφά από τα άλλα αντικείμενα αλλά και να προσφέρει μεθόδους με τις οποίες μπορούν άλλα αντικείμενα να επικοινωνούν και να αλληλεπιδρούν μαζί του. Επιπλέον ο κώδικας που υλοποιεί αυτές τις μεθόδους είναι άγνωστος σε άλλες κλάσεις ή αντικείμενα.

Έτσι έχουμε την δημιουργία ενός interface επικοινωνίας ανεξάρτητου από την υλοποίηση και την εσωτερική δομή του αντικειμένου. Αυτό λέγεται implementation hiding που είναι μία μορφή data abstraction.

Ύστερα από τα παραπάνω μπορούμε να δούμε πώς ορίζουμε ένα αντικείμενο ή με άλλα λόγια πώς ορίζουμε την κλάση του.

2.3 Ορισμός κλάσεων στην JAVA

Μία κλάση της java μοιάζει αρκετά με ένα structure της C. Δηλαδή έχει την ακόλουθη δομή :

```
class <class_name> {
        <data_type or class_name>       <variables or objects>;

        <returned type> <method_name> (<parameter list>) {
                <method body>
        }

        <data_type or class_name>       <variables or objects>;

        <returned type> <method_name> (<parameter list>) {
                <method body>
        }
```

κ.ο.κ.
}

Π.χ.

```
class Apple {
        int     num_of_vitamins;            // πεδίο

        int countVitamins() {               // μέθοδος
                return num_of_vitamins;
        }

        Vitamins        v1, v2, v3;         // πεδία

        Vitamin getVitamin(int num) {       // μέθοδος
                return ....;
        }
}
```

Το num_of_vitamins είναι μία κοινή μεταβλητή τύπου int. Σχετικά με τους τύπους δεδομένων των κοινών μεταβλητών της java (πχ int, float, char κλπ), θα μιλήσουμε λίγο αργότερα. Αντίθετα το Vitamins είναι όνομα μιας κλάσης και συνεπώς τα v1, v2, v3 θα είναι αντικείμενα της κλάσης Vitamin ή καλύτερα αναφορές (κάτι σαν δείκτες) στα αντίστοιχα αντικείμενα. Τέλος βλέπουμε ότι οι μέθοδοι μπορούν να έχουν επιστρεφόμενο τύπο τόσο έναν απλό τύπο δεδομένων όσο και μία κλάση.

Γενικά

Μπορούμε να δηλώνουμε τα πεδία και τις μεθόδους με όποια σειρά θέλουμε. Επίσης μπορούμε να τα χρησιμοποιούμε πριν ακόμη τα δηλώσουμε. Ωστόσο συνιστάται τα πεδία να τοποθετούνται στην αρχή της κλάσης ώστε ο κώδικας να είναι πιο ευανάγνωστος. Ακόμη η τοποθέτηση κατατοπιστικών σχολίων συμβάλλει σημαντικά στην βελτίωση της αναγνωσιμότητας του προγράμματος και διευκολύνει την διαδικασία αποσφαλμάτωσης (debugging).

```
class test {
        float add2Accum(float f) {      // Add to accumulator.
                return (Accumulator += f);
        }
        float   Accumulator = 0.0;      // The accumulator.
}
```

Για τους τύπους δεδομένων (ή primitive types) της java, την βιβλιοθήκη καθώς και για τις εντολές (statements) που χρησιμοποιούμε στο σώμα των μεθόδων, θα μιλήσουμε λίγο αργότερα.

2.4 Κληρονομικότητα (inheritance)

Είναι ο μηχανισμός εκείνος ο οποίος επιτρέπει σε μια κλάση Β να κληρονομήσει πεδία και μεθόδους από μια κλάση Α. Λέμε ότι η "Β κληρονομεί από την Α". Τα αντικείμενα (instances) της κλάσης Β έχουν πρόσβαση στα δεδομένα και τις μεθόδους της κλάσης Α χωρίς να χρειάζεται να τα ξαναδηλώσουμε. Δηλαδή είναι σαν να αντιγράψαμε τα περιεχόμενα της κλάσης Α στην κλάση Β. Ακόμη, να πούμε ότι για να δηλώσουμε ότι η Β κληρονομεί από την Α χρησιμοποιούμε την δεσμευμένη λέξη **extends** :

Ένα παράδειγμα σε java :

```
class A {
        int     a;
        void add(int x) {
                a += x;}
}
```

```
class B extends A {
        // Τα 'a' και 'void add(int x) {...}' υπονοούνται.
        int     prev;
        void sub(int x) {
                prev = a;
                a -= x;
        }
}
```

Βλέπουμε δηλαδή ότι στην κλάση Β δεν ξαναγράφουμε τον κώδικα που αφορά το a και το void add(int x). Αυτό μας απαλλάσσει από σημαντικό προγραμματιστικό φόρτο. Επίσης βλέπουμε ότι μπορούμε στην Β να δηλώσουμε νέα πεδία και μεθόδους και έτσι να επεκτείνουμε τη λειτουργικότητα της κλάσης αυτής σε σχέση με την Α.

Είναι δυνατό οι μέθοδοι που ορίζονται στη Β να χρησιμοποιούν τα πεδία που δηλώθηκαν στην Α και να καλούν τις μεθόδους που ορίσθηκαν σ' αυτήν (την Α). Π.χ.

```
class B extends A {
        int square() {
                return a*a;
        }
        void increaseBy1() {
                add(1);
        }
}
```

Επιπλέον είναι δυνατό η κλάση Β να μεταβάλλει κάποια από τα ήδη υπάρχοντα πεδία ή μεθόδους. Για παράδειγμα :

```
class B extends A {
        a : float;              // Αλλάζει τον τύπο της α.
        void add(int x) {       // Αλλάζει την add(int x).
                                // Αυτό λέγεται overriding.
                a = a + ((float) x);
                print(a);
        }
}
```

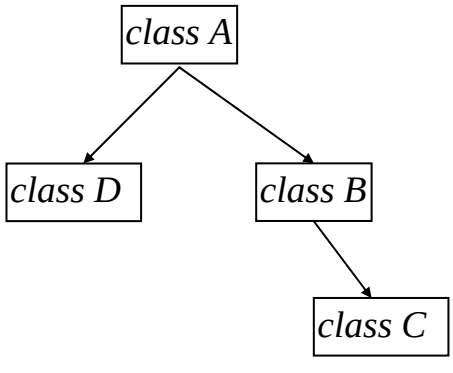
Όπως αναφέρεται και στο παράδειγμα η μεταβολή κάποιου πεδίου ή μεθόδου, που κληρονομήθηκε από την κλάση Α στην κλάση Β, λέγεται ***υπερκάλυψη*** **(overriding)** του πεδίου/μεθόδου της Α από το αντίστοιχο πεδίο/μεθόδο της Β.

Προσέξτε ότι η μέθοδος της Β που κάνει override έχει το ίδιο όνομα, λίστα παραμέτρων και επιστρεφόμενο τύπο με αυτήν της Α. Αν δεν είναι τα ίδια τότε δεν έχουμε υπερκάλυψη αλλά ορισμό μιας νέας μεθόδου.

Ορολογία

Η κλάση Β, (παραγόμενη), λέγεται υποκλάση (subclass) της Α ή παράγωγη κλάση ή παιδί της Α. Η Α τώρα λέγεται υπερκλάση (superclass) της Β ή παράγουσα κλάση ή πατέρας της Β. Η διαδικασία αυτή λέγεται subclassing.

Βέβαια από την Α μπορούν να προκύψουν και άλλες κλάσεις εκτός από την Β. Επιπλέον και από την Β μπορούν να προκύψουν νέες υποκλάσεις. Δηλαδή η κληρονομικότητα δημιουργεί μια δενδρική συσχέτιση μεταξύ των κλάσεων. Π.χ.

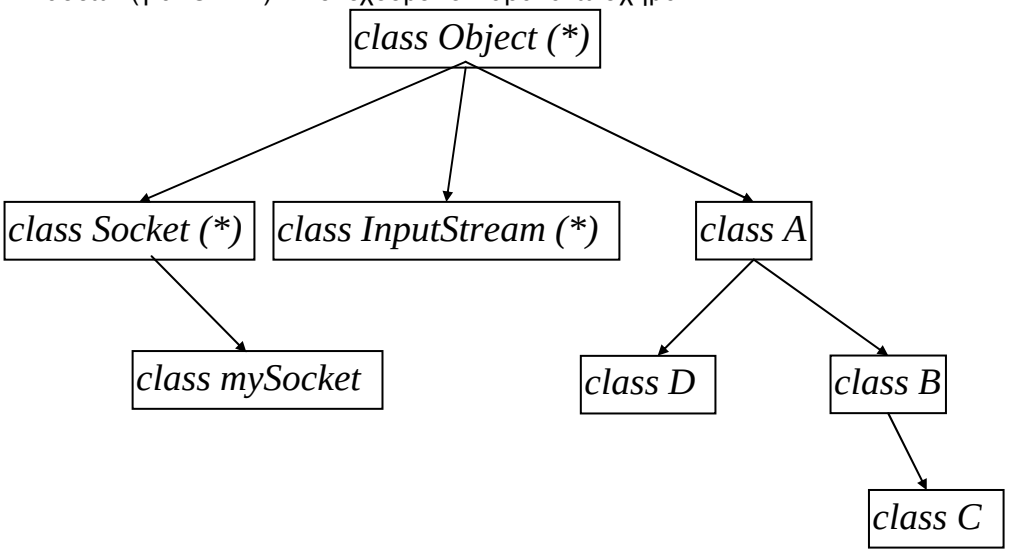


Στο παραπάνω παράδειγμα η A είναι superclass των B και D. Η B είναι superclass της C, επομένως η A είναι superclass και της C! (μεταβατική ιδιότητα). Επίσης οι B, D είναι subclasses της A, ενώ η C είναι subclass τόσο της B όσο και της A. Φυσικά η C θα περιέχει τα δεδομένα και τις μεθόδους και της A και της B.

Φυσικά όλα όσα αναφέρθηκαν σχετικά με το overriding και τη δυνατότητα πρόσβασης σε πεδία/μεθόδους των υπερκλάσεων ισχύουν τόσο για τον πατέρα όσο και για τον πατέρα του πατέρα κ.ο.κ.. Δηλαδή η C μπορεί να δει όλες τις μεθόδους της A και φυσικά μπορεί να τους κάνει override εφ' όσον δεν το έχει κάνει η B.

Σημείωση - 1

Όσες κλάσεις δεν κληρονομούν ρητά κάποια άλλη κλάση τότε υπονοείται ότι κληρονομούν την κλάση Object η οποία παρέχεται μαζί με τον compiler και με την βιβλιοθήκη έτοιμων κλάσεων (για IO κλπ). Έτσι έχουμε το παρακάτω σχήμα :



Οι κλάσεις με αστερίσκο (*) παρέχονται μαζί με τον compiler ενώ οι κλάσεις A, B, C, D, mySocket είναι ορισμένες από τον χρήστη.

Σημείωση - 2 (οι μεταβλητές this και super)

Όταν κάποια κλάση υπερκαλύπτει-μεταβάλλει (overrides) μία μέθοδο που κληρονόμησε από την υπερκλάση της τότε προφανώς ορίζει μια νέα μέθοδο. Ο ορισμός αυτής της νέας μεθόδου εμποδίζει την subclass από το να μπορεί να καλεί την μέθοδο που ορίστηκε στην υπερκλάση της. Π.χ.

```
class A {
        void doSomething() {
                ......
        }
}

class B extends A {
        void doSomething() {    // "Υπερκαλύπτει" (overrides) την
                ........                // doSomething() της κλάσης A.
        }
}

A       o1;
B       o2;
o1.doSomething();               // Καλείται η doSomething() της A.
o2.doSomething();               // Καλείται η doSomething() της B.
```

Αν η κλάση Β ή κάποια παράγωγή της (πχ η κλάση C) θέλει να χρησιμοποιήσει την doSomething() της Α τότε δεν μπορεί να το κάνει αυτό άμεσα, αλλά μόνο με έμμεσο τρόπο, εφ'όσον βέβαια η γλώσσα προβλέπει έναν.

Η java παρέχει έναν τέτοιο τρόπο ο οποίος έχει ως εξής. Κάθε φορά που δημιουργούμε μία υποκλάση Β, πχ της Α, η γλώσσα δημιουργεί αυτόματα τη μεταβλητή - δείκτη **super** η οποία δείχνει στην υπερκλάση της δηλ την Α. Έτσι όταν θέλουμε να καλέσουμε μία μέθοδο της Α στην οποία έχουμε κάνει override, (πχ την doSomething()), τότε την καλούμε ως εξής :

```
super.doSomething();
```

Επίσης η java μαζί με την super δημιουργεί και την **this** η οποία είναι μια μεταβλητή (δείκτης) που δείχνει στο ίδιο το αντικείμενο. Το this είναι χρήσιμο στις εξής δύο περιπτώσεις :

α) Όταν μία παράμετρος κάποιας μεθόδου έχει το ίδιο όνομα με κάποιο πεδίο της κλάσης.

```
class A {
        int     a;
        void set(int a) {
                this.a = a;
        }
}
```

β) Όταν ένα αντικείμενο θέλει να δώσει ένα δείκτη του σε ένα άλλο. Έτσι το δεύτερο αντικείμενο θα μπορεί να καλεί τις μεθόδους του πρώτου.

```
class A {
        void make_a_new_object() {
                B tmp;
                // create a new instance of B and store it in tmp.
                // Then give tmp a pointer to you.
                tmp.owner(this);
        }
}
```

2.5 Υπερφόρτωση (overloading)

Πριν περάσουμε στους τρόπους δημιουργίας αντικειμένων ας δούμε το παρακάτω κομμάτι κώδικα, το οποίο είναι σωστό.

```
class number {
        int     a;

        void add(int x) {
                a += x;
        }
        void add(float f) {
                a += ((int) f);
        }
        void add(number n) {
                a += n.get_a();
        }

        int get_a() {
                return a;
        }
}
```

Βλέπουμε δηλαδή ότι η συνάρτηση *void add(...)* χρησιμοποιείται πολλές φορές (3) αλλά κάθε φορά με διαφορετική λίστα παραμέτρων. Αυτή η επαναχρησιμοποίηση του ίδιου ονόματος λέγεται **υπερφόρτωση** (overloading) του ονόματος αυτού.

Στις object oriented γλώσσες επιτρέπεται ο ορισμός μίας μεθόδου με τι ίδιο όνομα δύο ή περισσότερες φορές αρκεί να έχουν διαφορετικές λίστες παραμέτρων. Παράδειγμα :

aMethod(), aMethod(type1), aMethod(type2), aMethod(type1, type2), aMethod(type2, type1).

Όλες οι παραπάνω μέθοδοι είναι διαφορετικές μεταξύ τους. Έτσι, κατά την κλήση μιας μεθόδου κατ' αρχήν εξετάζεται το όνομά της και έπειτα εξετάζεται και η λίστα των παραμέτρων της, όσον αφορά το πλήθος, τον τύπο και τη σειρά των παραμέτρων. Αυτό είναι μια μορφή πολυμορφισμού.

2.6 Κατασκευαστές *(constructors)*

Κάθε κλάση διαθέτει τουλάχιστο μία μέθοδο η οποία εκτελείται κατά τη δημιουργία ενός στιγμιότυπου της, (δηλ. κατά τη δημιουργία ενός object της κλάσης αυτής). Αυτές οι μέθοδοι λέγονται **κατασκευαστές** (constructors) της κλάσης. Σκοπός των constructors είναι η δέσμευση μνήμης για την κατασκευή του αντικειμένου καθώς και η διενέργεια κατάλληλων αρχικοποιήσεων. Ο προγραμματιστής μπορεί να ορίσει πολλούς constructors για μία κλάση αλλά για τη δημιουργία ενός αντικειμένου (στιγμιότυπου) μπορεί να καλέσει μόνο έναν από αυτούς. Φυσικά η επιλογή είναι ελεύθερη.

Αν όμως ο προγραμματιστής **δεν** ορίσει κανένα constructor τότε η γλώσσα καλεί έναν constructor της υπερκλάσης (κάποιον που δεν θέλει παραμέτρους) ή δίνει μήνυμα λάθους αν δεν βρει κάποιον τέτοιο.

Ένα θέμα που προκύπτει είναι το πώς ορίζονται οι constructors. Στην JAVA ορίζονται όπως ακριβώς και οι απλές μέθοδοι με μόνη τη διαφορά ότι έχουν το **ίδιο** όνομα με την κλάση, πχ :

```
class A {
        int     n;
        A() {           // Constructor - 1
                n=0;
        }
        A(int x) {      // Constructor - 2
                n = x;
        }
}
```

Σε αυτήν την περίπτωση οι constructors ξεχωρίζονται μεταξύ τους από τις λίστες των παραμέτρων τους. Επίσης, **οι constructors δεν έχουν επιστρεφόμενο τύπο**, ούτε void. (Βλέπε το παραπάνω παράδειγμα).

Η κατασκευή ενός νέου αντικειμένου γίνεται ως εξής:

```
A       a;
a = new A();
```

ή

```
a = new A(5);
```

ΣΗΜΕΙΩΣΕΙΣ

1. Ο constructor με κενή λίστα παραμέτρων καλείται default ή null constructor της κλάσης. Αν λοιπόν η κλάση μας καλείται Α και κάποια υποκλάση της (έστω) Β δεν έχει ορίσει δικούς της constructors τότε χρησιμοποιείται ο default constructor της Α.
2. Μια μέθοδος ενός αντικειμένου (στιγμιότυπου) **δεν** μπορεί να καλεί τους constructors. Αυτό είναι λογικό αφού το αντικείμενο δεν έχει ακόμη κατασκευαστεί.
3. Όταν ένας constructor καλείται, πριν κάνει οτιδήποτε άλλο καλεί τον default constructor της υπερκλάσης του για την δημιουργία και αρχικοποίηση των πεδίων που κληρονομούνται από αυτή. Φυσικά και ο constructor της υπερκλάσης καλεί έναν constructor της δικής του υπερκλάσης κ.ο.κ..

4. Ο προγραμματιστής μπορεί να ορίσει κάποιον constructor έτσι ώστε κατά το instanciation να μην καλέσει τον default constructor της υπερκλάσης αλλά αυτόν που ο ίδιος (ο προγραμματιστής) θέλει. Αυτό επιτυγχάνεται αν σαν πρώτη εντολή του constructor μας βάλουμε την super(....); Το super(...) εδώ δεν πρόκειται για την μεταβλητή super που είδαμε παραπάνω, (μαζί με την μεταβλητή this), αλλά αντιπροσωπεύει κάποιον από τους constructor της υπερκλάσης. Τον ποιον από αυτούς αντιπροσωπεύει θα εξαρτηθεί από τη λίστα των παραμέτρων (του super).
5. Επίσης όπως ακριβώς καλείται ο super(...) έτσι μπορεί να κληθεί και ο this(....) ο οποίος αντιπροσωπεύει κάποιον από τους άλλους constructors που ορίστηκαν στην τρέχουσα κλάση.

Η java παρέχει πρόσθετες ευκολίες στην κατασκευή και χειρισμό των strings (συμβολοσειρές χαρακτήρων) λόγω της μεγάλης σημασίας τους στις εφαρμογές. Ένα string στην java είναι ένα αντικείμενο της κλάσης String, και έχει δύο τρόπους κατασκευής :

```
String str = new String("Hello, I am a new string!");
```

ή

```
String str = "Hello, I am a new string!";
```

2.7 Καταστροφή Αντικειμένων (Finalization)

Σε ορισμένες γλώσσες όπως η C++ και η Object PASCAL υποστηρίζεται και η χρήση των **καταστροφέων** (destructors) μιας κλάσης. Προφανώς οι destructors είναι μέθοδοι που τερματίζουν την ύπαρξη ενός στιγμιότυπου της αντίστοιχης κλάσης. Ωστόσο στην JAVA το χαρακτηριστικό αυτό δεν υποστηρίζεται αφού είναι αιτία λαθών. Φαντασθείτε για παράδειγμα ότι δύο δείκτες δείχνουν στο ίδιο αντικείμενο και ότι με χρήση του ενός από αυτούς τερματίζουμε την ύπαρξη του αντικειμένου. Τότε ο άλλος δείκτη θα δείχνει σε άκυρα δεδομένα (dangling pointer).

Η JAVA όμως έχει να αντιπαρατάξει έναν άλλο ασφαλέστερο τρόπο για τερματισμό (finalization) της ύπαρξης ενός αντικειμένου. Ο τρόπος αυτός έχει ως εξής. Το runtime system της java εξετάζει κατά διαστήματα αν για κάθε αντικείμενο υπάρχει τουλάχιστο ένας δείκτης σε αυτό. Αν δεν υπάρχει τότε το αντικείμενο είναι άχρηστο (garbage) και η μνήμη που κατέχει απελευθερώνεται. Όμως προτού γίνει αυτό καλείται η μέθοδος *void finalize()* η οποία πρέπει να περιέχει κώδικα για clean-up. Για παράδειγμα αν το αντικείμενο έχει ακόμα ανοιχτές συνδέσεις στο δίκτυο τότε θα τις κλείσει. Μετά το τέλος της finalize() το αντικείμενο καταστρέφεται.

Η διαδικασία του εντοπισμού τον αντικειμένων - σκουπιδιών λέγεται garbage collection και ο μηχανισμός που το πραγματοποιεί είναι ενσωματωμένος στο JAVA runtime system.

2.8 Προσδιοριστές πρόσβασης (access specifiers)

Τυχαίνει πολλές φορές να θέλουμε να έχουμε κάποια δεδομένα ή μεθόδους μας ορατά στα αντικείμενα όλων των κλάσεων ή να θέλουμε κάποια πεδία να είναι ορατά μόνο στα αντικείμενα της τρέχουσας κλάση και όχι και στις υποκλάσεις της κλπ. Σε όλες αυτές τις περιπτώσεις χρησιμοποιούμε τους access specifiers. Η σύνταξη των δηλώσεων με τους access specifiers παρόντες γίνεται:

```
<access specifiers> <type or class>    <variables>;

<access specifiers> <returned type> <method name> (<param list>) {
        <body>
}
```

Οι access specifiers είναι δεσμευμένες λέξεις της JAVA οι οποίες καθορίζουν το ποιος θα έχει πρόσβαση, δηλαδή το ποιος θα μπορεί να χρησιμοποιεί και να επεξεργάζεται, τις μεθόδους ή/και τα πεδία που έχουν κάποιον τέτοιο προσδιοριστή. Κατ' αρχήν ας δούμε ποιοι είναι :

- **public** - Σημαίνει ότι το συγκεκριμένο πεδίο/μέθοδος μπορεί να χρησιμοποιηθεί από οποιονδήποτε, ακόμα και από ξένο αντικείμενο.
- **protected** - Σημαίνει ότι το πεδίο/μέθοδος που το έχει είναι ορατό μόνο στις μεθόδους της κλάσης αυτής καθώς και στις υποκλάσεις.
- **private** - Σημαίνει ότι το πεδίο/μέθοδος είναι ορατό μόνο στις μεθόδους αυτής της κλάσης αλλά όχι στις υποκλάσεις της.
- **τίποτα** - Είναι public αλλά μόνο για το τρέχων package.

Παραδείγματα :

```
class A {
        public int      a;      //φαίνεται από παντού
        int             b;      //φαίνεται σε όλο το package, πρακτικά παντού
        protected       c;      //φαίνεται στην A και την B
        private  d;     //φαίνεται μόνο στην A.

        public A() {...}
        public void add(int x) {....}
        void add(A a) {.....}

        protected int convert_A_to_int (A a) { return a.getValue(); }
        private void who_am_i() { System.out.println("I am class A!"); }
}

class B extends A {
        // I CAN SEE :  a, b, c, A() via super(), add(int), add(A),
        // convert_A_to_int(A),  BUT I CAN'T SEE : d, who_am_i()
}
```

Επιπλέον σε κάθε πρόγραμμα θα πρέπει να υπάρχει το πολύ μία κλάση δηλωμένη ως εξής:

```
public class A {
        .......
}
```

Αυτή η κλάση (A) θα είναι η αφετηρία του προγράμματός μας, δηλαδή αυτήν την κλάση θα πρέπει να δώσουμε από το command line στον java interpreter για να ξεκινήσει την εκτέλεση του προγράμματος. Επίσης το όνομα του αρχείου με τον πηγαίο κώδικα θα πρέπει να έχει το όνομα της κλάσης αυτής συν την επέκταση .java, (π.χ. A.java). Επιπλέον, η κλάση - αφετηρία πρέπει να περιέχει και μία μέθοδο δηλωμένη ως εξής :

```
public static void main(String args[]) {
        ..... // Το σώμα του προγράμματος γράφεται εδώ!
}
```

Αυτή η μέθοδος είναι σαν την main() της C.

2.9 Τροποποιητές (Modifiers)

Είναι δεσμευμένες λέξεις της JAVA οι οποίες προσδίδουν συγκεκριμένες ιδιότητες ή χαρακτηρίζουν τα πεδία/μεθόδους που τα έχουν. Κατ` αρχήν η σύνταξη έχει ως εξής :

<access specifiers> <modifiers> <type or class> <variables>;

<access specifiers> <modifiers> <returned type or class> <method name> (<parameter list) {
        <body of the method>
}

Ειδικά για τους modifiers **final, abstract** ισχύει επιπλέον :
final/abstract class <class name> [extends <class name>] {
        .....

}

Οι πιο σημαντικοί από τους modifiers είναι οι ακόλουθοι :

- **final** - Για τα πεδία σημαίνει ότι είναι σταθερές και όχι μεταβλητές. Συνεπώς είναι λάθος να τους αναθέσεις νέα τιμή. Ανάθεση τιμής επιτρέπεται μόνο κατά την αρχικοποίηση της σταθεράς όταν αυτή δηλώνεται, πχ final int a = 10; Αλλά όχι final int a; a=10; Για τις μεθόδους σημαίνει ότι δεν μπορείς να τους κάνεις override. Τέλος για τις κλάσεις σημαίνει ότι δεν μπορείς να κατασκευάσεις υποκλάσεις από αυτές.
- **synchronized** - Για μεθόδους σημαίνει ότι το αντικείμενο *ιδιοκτήτης* της μεθόδου αυτής είναι threadsafe. Με άλλα λόγια αν κάποιος χρησιμοποιεί αυτή την μέθοδο τότε, (με χρήση σηματοφορέα), αποκλείει την ταυτόχρονη χρήση του αντικειμένου και από τρίτον. Ο τρίτος θα περιμένει μέχρι να ελευθερωθεί το αντικείμενο και έπειτα θα το χρησιμοποιήσει και αυτός. ΠΡΟΣΟΧΗ: αν κάποιος (τρίτος) θέλει να χρησιμοποιήσει την ίδια μέθοδο ενός άλλου αντικειμένου, (της ίδιας όμως κλάσης), δεν θα έχει κανένα πρόβλημα να το κάνει αρκεί να μην χρησιμοποιείται ήδη και αυτό.
- **abstract** - Για μεθόδους σημαίνει ότι δεν θα υλοποιηθούν εδώ αλλά σε κάποια υποκλάση, έτσι **δεν** παρέχεται το σώμα της μεθόδου *{...}* αλλά στη θέση του βάζουμε ένα semicolon (;). Επίσης μία κλάση με έστω μία abstract μέθοδο θα πρέπει να δηλωθεί και αυτή ως abstarct. Είναι compile-time error να προσπαθήσουμε να κατασκευάσουμε (με new) μία abstract κλάση. Αντ' αυτού θα πρέπει να δημιουργήσουμε υποκλάσεις που θα κάνουν override σε όλες τις abstract μεθόδους.
- **static** - Το πεδίο/μέθοδος που το έχει είναι μοναδικό για όλα τα στιγμιότυπα της κλάσης. Όλα τα αντικείμενα της κλάσης αυτής έχουν ένα τέτοιο πεδίο σαν κοινή μεταβλητή. Επίσης οι static μεταβλητές και μέθοδοι μπορούν να χρησιμοποιηθούν και χωρίς να κατασκευαστεί κάποιο αντικείμενο (στιγμιότυπο) της κλάσης. Αυτό γίνεται ως εξής : <όνομα κλάσης>.<static var or method>. ΠΡΟΣΟΧΗ : οι static μέθοδοι δεν μπορούν να δουν ή να αλλάξουν τα πεδία που δεν είναι static. Ωστόσο το αντίθετο είναι επιτρεπτό. Μία static μέθοδος είναι και η public static void main(String args[]) που είδαμε παραπάνω, η οποία καλείται πριν ακόμα κατασκευαστεί κάποιο αντικείμενο.
- **native** - Όσες μέθοδοι δηλωθούν native δεν πρέπει να έχουν σώμα. Ωστόσο ο κώδικάς τους θα πρέπει να γραφεί σε C και θα συνδεθεί με το bytecode κατά την εκτέλεση του προγράμματος. Στην συγγραφή του native κώδικα βοηθά το **javah.**

2.10 Casting

Όπως συμβαίνει και στην C έτσι και στην JAVA είναι δυνατό το casting μεταξύ μεταβλητών διαφορετικού τύπου δεδομένων ή διαφορετικών κλάσεων. Για παράδειγμα :

```
int     a;
short   b = 4;

a = (int) b;        // short --> int    explicit casting.
```

ή ισοδύναμα

```
a = b;              // implicit casting

b = (short) a;      // int --> short    explicit casting ONLY!

A       a;
B       b;          // B subclass of A
a = (A) b;          // B --> A;
a = b;
b = (B) a;          // explicit casting is required!
```

Γενικός κανόνας

Για τους απλούς τύπους δεδομένων το explicit casting είναι απαραίτητο όταν υπάρχει απώλεια πληροφορίας, για παράδειγμα όταν αποθηκεύουμε έναν int (32-bit ακέραιο) σε ένα byte (8-bit ακέραιο).

Για τις κλάσεις, (ή καλύτερα για τις μεταβλητές που περιέχουν κλάσεις), το explicit casting είναι υποχρεωτικό όταν θέλουμε να αποθηκεύσουμε μία υπερκλάση σε μία υποκλάση.

2.11 Μεταβλητές - Αναφορές

Τελειώνοντας θα πρέπει να κάνουμε μερικές παρατηρήσεις που αφορούν τις μεταβλητές (πεδία) των αντικειμένων. Όσες μεταβλητές έχουν δηλωθεί σαν μεταβλητές ενός απλού τύπου δεδομένων (int, float,short κλπ) είναι ακριβώς όπως αυτές της C.

```
int     a;
short   b;
```

Οι μεταβλητές όμως που έχουν δηλωθεί σαν μεταβλητές κάποιας κλάσης είναι στην πραγματικότητα δείκτες στον αντικείμενο που τους ανατίθεται. Όμως σε αντίθεση με τη C δεν χρησιμοποιούνται οι τελεστές (* & ->) για να αναφερθούμε στα αντικείμενα. Αντίθετα οι μεταβλητές χρησιμοποιούνται σαν να μην ήταν δείκτες αλλά σαν να ήταν τα ίδια τα αντικείμενα.

Επίσης να σημειώσουμε ότι η JAVA δεν επιτρέπει την αριθμητική διευθύνσεων ή δεικτών. Για παράδειγμα

```
A       a = new A();
a++;                    // ΛΑΘΟΣ !!!
```

Ωστόσο

```
int     a;
a++;                    // ΣΩΣΤΟ διότι το a εδώ είναι τύπου int και όχι
                        // δείκτης.
```

Για να είμαστε ακόμη πιο ακριβείς οι μεταβλητές που δείχνουν σε αντικείμενα (πχ A a) δεν είναι δείκτες, με την συμβατική έννοια, στα αντικείμενα, αλλά αναφορές (references - ref) ή χειριστές (handlers) οι οποίοι δεν δείχνουν σε μία διεύθυνση της μνήμης αλλά σε κάποια δομή του runtime system. Έτσι το runtime system έχει τη δυνατότητα να ξέρει ανά πάσα στιγμή ποιο αντικείμενο αντιστοιχεί σε κάθε μεταβλητή και έτσι μπορεί να κάνει το garbage collection και την διαχείριση της μνήμης με μεγαλύτερη αποτελεσματικότητα και ασφάλεια.

Τελειώνοντας να πούμε ότι η αναφορά null μπορεί να ανατεθεί σε μεταβλητές αντικειμένων.

2.12 Πολυμορφισμός (Polymorfism)

Πολυμορφισμός είναι η δυνατότητα μίας μεθόδου να κάνει διαφορετικές ενέργειες ανάλογα με τον τύπο του αντικειμένου πάνω στο οποίο δρά. Είναι η τρίτη βασική αρχή του αντικειμενοστραφούς προγραμματισμού.

Οι τύποι πολυμορφισμού είναι:

- Η υπερκάλυψη (overriding)
- Η υπερφόρτωση (overloading)
- Η δυναμική συσχέτιση μεθόδων (dynamic method binding)

Οι δύο πρώτες έχουν αναφερθεί προηγουμένως, η τρίτη φαίνεται στο ακόλουθο παράδειγμα:

Υποθέστε ότι τρείς υποκλάσεις (Cow, Dog και Snake) έχουν δημιουργηθεί χρησιμοποιώντας την αφηρημένη (abstract) κλάση Animal, με κάθε μια από τις οποίες να υλοποιεί την δική της μέθοδο speak().

```
public class AnimalReference
{
```

```
  public static void main(String[] args)

        Animal ref; // set up a reference for an Animal

                                        // make specific objects
        Cow aCow = new Cow("Bossy");
        Dog aDog = new Dog("Rover");
        Snake aSnake = new Snake("Earnie");

                                    // now reference each as an Animal
        ref = aCow;
        ref.speak();
        ref = aDog;
        ref.speak();
        ref = aSnake;
        ref.speak();
}
```

Παρατηρήστε ότι αν και η κλήση της μεθόδου speak() έγινε σε αναφορά τύπου Animal, το πρόγραμμα μπορεί να αποφανθεί για την σωστή μέθοδο (ανάλογα με την υποκλάση) και τον χρόνο εκτέλεσης. Αυτό ονομάζεται δυναμική συσχέτιση μεθόδων (dynamic method binding)

2.13 Διασυνδέσεις (Interfaces)

Μια διασύνδεση(interface) ορίζει έναν τρόπο συμπεριφοράς που μπορεί να υλοποιηθεί από οποιαδήποτε κλάση. Δηλώνει ένα σύνολο μεθόδων αλλά δεν προσφέρει την υλοποίηση τους. Αυτές οι μέθοδοι θεωρούνται αυτόματα εικονικές και οποιαδήποτε κλάση υλοποιεί το interface πρέπει να δώσει και υλοποίηση για όλες τις μεθόδους του. Ο λόγος για τον οποίο υπάρχουν τα interfaces είναι ότι κάθε κλάση μπορεί να έχει μόνο μία υπερκλάση αλλά μπορεί να υλοποιήσει άπειρο αριθμό interfaces.

Ένα interface μοιάζει με μία abstract κλάση αλλά έχει τις εξής σημαντικές διαφορές:

- Ένα interface δεν μπορεί να παρέχει υλοποίηση για καμία μέθοδο του
- Μια κλάση μπορεί να υλοποιήσει πολλά interfaces αλλά να κληρονομήσει μία μόνο κλάση
- Ένα interface δεν είναι μέρος της ιεραρχίας κλάσεων της Java και έτσι άσχετες μεταξύ τους κλάσεις μπορούν να το υλοποιούν.

Ένα interface μπορεί επίσης να περιέχει σταθερές στον ορισμό του ή να κληρονομεί από άλλα interfaces και ορίζεται γενικά με αυτή την μορφή:

```
public interface Imyinterface extends Isuperinterface1,Isuperinterface2,…  {
        final String One  = "One";
        final String Tow = “Two”;
        public void method();
}
```

Υπάρχει η σύμβαση τα ονόματα των interfaces να ξεκινάνε με το κεφαλαίο γράμμα I αλλά αυτό δεν είναι υποχρεωτικό και υπάρχει μόνο για διευκόλυνση. Όλες οι σταθερές θεωρούνται αυτόματα public, static, και final ενώ όλες οι μέθοδοι θεωρούνται αυτόματα public και abstract. Κάποιοι access modifiers δεν είναι δυνατό να χρησιμοποιηθούν παραδείγματος χάριν οι private και protected.

Η χρησιμότητα των interfaces θα φανεί με το παρακάτω παράδειγμα. Θεωρώντας ότι θέλουμε μία κλάση η οποία είναι ταυτόχρονα applet και thread και δεδομένου ότι δεν μπορούμε να έχουμε πολλαπλή κληρονομικότητα θα είχαμε κώδικα σαν αυτόν :

```
public class MyThreadedApplet extends Applet implements Runnable {
…
```

```
}
```

Το Runnable interface μας επιτρέπει να έχουμε την λειτουργικότητα ενός thread χωρίς να είμαστε αναγκασμένοι να κληρονομούμε από την κλάση Thread. Περισσότερα στο κεφάλαιο για τα threads.

Είναι δυνατόν να ορίσουμε σαν τύπο παραμέτρου σε συνάρτηση ένα interface. Έχοντας πάλι το Runnable interface υπόψη θα μπορούσαμε να έχουμε μία συνάρτηση

```
public void Method (Runnable mythread,String S, double d) {
   ...
}
```

Οποιαδήποτε κλάση που υλοποιεί άμεσα η έμμεσα το interface μπορεί να περαστεί ως παράμετρος σε αυτή την μέθοδο.

Ένα σημαντικό σημείο κατά τον ορισμό των interfaces είναι ότι όταν θέλουμε να προσθέσουμε λειτουργικότητα προσθέτοντας και άλλες μεθόδους σε ένα interface θα πρέπει να θυμόμαστε πως όλες οι κλάσεις που το υλοποιούν δεν θα μπορούν να χρησιμοποιηθούν μέχρι να προσθέσουμε σε κάθε μία από αυτές την υλοποίηση των νέων μεθόδων του interface.

2.14 Πακέτα (packages)

Κλάσεις συναφούς λειτουργικότητας ομαδοποιούνται σε πακέτα (packages) Οι κλάσεις του ίδιου πακέτου βρίσκονται στο ίδιο directory. Ένα πακέτο έχει το ίδιο όνομα με το όνομα του directory στο οποίο αποθηκεύονται οι κλάσεις του.

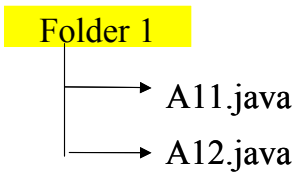
Η δήλωση του πακέτου στο οποίο ανήκει μια κλάση γίνεται στην αρχή του πηγαίου κώδικα της κλάσης:

```
package <packageName>;
```

Κλάσεις ενσωματωμένες στα packages προσαρτώνται στον κώδικα άλλων κλάσεων με τη δήλωση

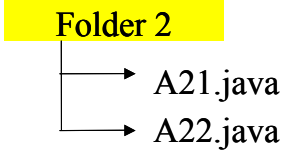
```
import <packageName>.<className>;
```

Παράδειγμα1:




```
//Δήλωση στον κώδικα της κλάσης A11.java
//ότι ανήκει στο package Folder1:
        package Folder1;
        class A11
                { Class Implementation…..}

//Δήλωση στον κώδικα της κλάσης A12.java
//ότι ανήκει στο package Folder1:
        package Folder1;
        class A12
        { class implementation……}
```

Παράδειγμα2:



```
//Δήλωση στον κώδικα της κλάσης A21.java
//ότι ανήκει στο package Folder2:
package Folder2;
class A21
{ Class implementation}

//Δήλωση στον κώδικα της κλάσης A22.java
//ότι ανήκει στο package Folder2
//και προσάρτηση σε αυτήν του κώδικα της κλάσης A11.java
package Folder2;
import Folder1.A11;
class A22
{ Class implementation}
```

3. Τύποι δεδομένων - εντολές - τελεστές

3.1 Τύποι δεδομένων στην JAVA

Οι τύποι δεδομένων στην JAVA είναι σαν αυτούς της C με μόνη διαφορά ότι το μέγεθός τους (σε bytes) είναι γνωστό και ίδιο σε όλες τις υλοποιήσεις της JAVA.

Τύπος	Όνομα	Μέγεθος
Byte	Byte	8-bit signed, -128..127
Short	Short	16-bit signed
Int	Ακέραιος	32-bit signed
Long	εκτεταμένος ακέραιος	64-bit signed
Float	πραγματικός (κινητής. υποδιαστολής)	32-bit signed
Double	διπλής ακρίβειας	64-bit signed
Char	unicode character	16-bit
Boolean	Boolean	true or false

3.2 Αναγνωριστικά, literals, σχόλια, διαχωριστές

Τα αναγνωριστικά στην JAVA είναι οποιαδήποτε ακολουθία των χαρακτήρων A..Z, a..,z, 0..9, _, $, η οποία ξεκινά με χαρακτήρα γράμμα (κεφαλαίο ή μικρό) ή _ ή $, αλλά όχι με ψηφίο (0..9). Επίσης δεν θεωρούνται αναγνωριστικά οι δεσμευμένες λέξεις της JAVA.

Τα literals στην JAVA είναι πολλών τύπων, πχ αριθμητικά, λογικά, χαρακτήρων κλπ. Μερικά παραδείγματα φαίνονται παρακάτω :

Integer literals (ακαίρεοι αριθμοί)

```
324             // decimal
0xABCD          // hexadecimal
032             // octal
2L              // long decimal integer
```

Float point literals (πραγματικοί αριθμοί)

```
3.1415  // float
3.1E12
.1e12
2E12
2.0d or 2.0D            //double
2.0f or 2.0F or 2.0     //float
```

Boolean literals (λογικά δεδομένα)

```
true, false             // boolean
```

Character literals (χαρακτήρες)

continuation	<newline>	\	// character literals
new-line	NL (LF)	\n	
horrizontal tab	HT	\t	
back space	BS	\b	
carriage return	CR	\r	
form feed	FF	\f	
backslash	\	\\	
single quote	'	\'	
double quote	"	\"	
octal bit pattern	0ddd	\ddd	
hex bit pattern	0xdd	\xdd	
unicode char	0xdddd	\udddd	

πχ

'\n' , '\23', 'a'

String literals (Συμβολοσειρές)

"" \\ the empty string
"\""
"This is a string"
"This is a \
two-line string"

Τα σχόλια στην JAVA υποδηλώνονται με τρεις τρόπους.

1. Με ζεύγος από /* και */ όπως στην C.
2. Με // όπου το κείμενο από το // έως το τέλος της τρέχουσας γραμμής θεωρείται ως σχόλιο
3. Με ζεύγος /** και */. Αυτό είναι πανομοιότυπο με το /* και */ αλλά χρησιμοποιείται από το javadoc για δημιουργία documentation.

Διαχωριστικά (separators) στη JAVA είναι οι χαρακτήρες :
+ - ! % ^ & * | ~ / > <
( ) { } [ ] ; ? : , . =
και επίσης το κενό (SPACE), ο οριζόντιος στηλογνώμονας HT ή \t, η αλλαγή γραμμής LF ή \n καθώς και τα σχόλια.

3.3 Τελεστές

Οι αριθμητικοί τελεστές είναι οι ακόλουθοι:

Τελεστής	Περιγραφή
+	Πρόσθεση
-	Αφαίρεση
*	Πολλαπλασιασμός
/	Διαίρεση
%	Υπόλοιπο
^	Ύψωση σε δύναμη

Οι συσχετιστικοί τελεστές είναι οι ακόλουθοι:

Τελεστής	Περιγραφή
<=	Μικρότερο ή ίσο
<	Μικρότερο
>=	Μεγαλύτερο ή ίσο
>	Μεγαλύτερο

Οι τελεστές ισότητας είναι οι ακόλουθοι:

Τελεστής	Περιγραφή
!=	Άνισο με
==	Ίσο με ==

Οι λογικοί τελεστές είναι οι ακόλουθοι:

Τελεστής	Περιγραφή
&&	ΚΑΙ (AND)
||	Η (OR)
!	ΟΧΙ (NOT)

Τους συσχετιστικούς τελεστές, τους τελεστές ισότητας και τους λογικούς τελεστές τους συναντάμε κυρίως στις δομές ελέγχου (βλέπε παρακάτω). Οι παραπάνω τελεστές χρησιμοποιούνται για συγκρίσεις μεταξύ αριθμών. Εάν η σύγκριση είναι αληθής τότε το αποτέλεσμα είναι η τιμή **true** διαφορετικά εάν είναι ψευδής τότε το αποτέλεσμα είναι η τιμή **false**.

Ο τελεστής αντιστοίχησης είναι ο:

Τελεστής	Περιγραφή
=	Τελεστής αντιστοίχησης

Ο τελεστής αύξησης και ο τελεστής μείωσης είναι οι ακόλουθοι:

Τελεστής	Περιγραφή
++	αύξηση κατά 1
--	μείωση κατά 1

Οι τελεστές ++ και -- χρησιμοποιούνται όταν θέλουμε να προσθέσουμε ή να αφαιρέσουμε το 1 από μία μεταβλητή. Έτσι

το ++a; ισοδυναμεί με το a=a+1;
ενώ το --a; ισοδυναμεί με το a=a-1;

Οι τελεστές ++ και -- μπορούν να χρησιμοποιηθούν είτε ως προθεματικοί τελεστές (δηλ. πριν την μεταβλητή, πχ ++a ή--a) είτε ως μεταθεματικοί (δηλ. μετά την μεταβλητή, πχ a++ ή a--).

Στην παράσταση ++a η τιμή του a αυξάνει πριν χρησιμοποιηθεί η τιμή της.
Στην παράσταση a++ η τιμή του a αυξάνει αφού χρησιμοποιηθεί η τιμή της.

Παράδειγμα:
Έτσι έστω ότι το a ισούται με 5 δηλαδή υπάρχει η δήλωση:

```
a = 5;
```

τότε η δήλωση

```
b = a++;
```

δίνει στο b την τιμή 5

ενώ η δήλωση

```
b=++a;
```

δίνει στο b την τιμή την τιμή 6. Το a και στις δύο περιπτώσεις γίνεται 6.

Οι τελεστές αντικατάστασης είναι οι ακόλουθοι

Τελεστής	Περιγραφή
+=	Τελεστής πρόσθεσης και αντιστοίχησης
-=	Τελεστής αφαίρεσης και αντιστοίχησης
*=	Τελεστής πολ/μου και αντιστοίχησης
/=	Τελεστής διαίρεσης και αντιστοίχησης
%=	Τελεστής υπολοίπου και αντιστοίχησης

Το a += b; ισοδυναμεί με το a = a+b;
Το a -= b; ισοδυναμεί με το a = a-b;
Το a *= b; ισοδυναμεί με το a = a*b;
Το a /= b; ισοδυναμεί με το a = a/b;
Το a %= b; ισοδυναμεί με το a = a %b;

Οι τελεστές πράξεων με bits είναι οι ακόλουθοι :

Τελεστής	Περιγραφή
&	AND για bit
|	OR για bit
^	XOR για bit
!	NOT για bit
<<	Ολίσθηση αριστερά
>>	Ολίσθηση δεξιά

Οι παραπάνω τελεστές αφορούν πράξεις σε επίπεδο bits.
Οι τελεστές &, |, ^ και ~ αντιστοιχούν στις απλές πράξεις της άλγεβρας Boole.
Οι τελεστές >> και << προκαλούν ολίσθηση στα δεξιά και στα αριστερά αντίστοιχα.

Έτσι για παράδειγμα εάν η μεταβλητή a είναι ο δυαδικός αριθμός 01101000 τότε
Η δήλωση

```
b = a >> 2;
```

δίνει στη μεταβλητή b την τιμή 00011010.

Προτεραιότητα των τελεστών

Προτεραιότητα τελεστών
. [ ] ( )
++ -- ! ~ instanceof
*** / %**
+ -
<< >> >>>
< > <= >=
== !=
&
^
|
&&
||
? :
= op=
,

Οι προτεραιότητα των τελεστών μικραίνει καθώς κατεβαίνουμε στον πίνακα, πχ το ***** έχει μεγαλύτερη προτεραιότητα από το **!=**. Σε περίπτωση ύπαρξης σε μια παράσταση τελεστών με την ίδια προτεραιότητα οι πράξεις γίνονται από αριστερά προς δεξιά.

Τέλος το **op=** είναι εξής τελεστές : **+= -= *= /= %= &= |= ^= <<= >>= >>>=**

3.4 Συμβολοσειρές (Strings)

Χαρακτήρες
char c='A';

Οι συμβολοσειρές (Strings) είναι ακολουθίες χαρακτήρων. Οι συμβολοσειρές στην Java είναι ένα αντικείμενα της κλάσης String.

Παράδειγα String:

```
String ntua="National Technical University of Athens";
```

Ή εναλακτικά όπως είπαμε στο δεύτερο κεφάλαιο

```
String ntua = new String("National Technical University of Athens");
```

Η εμφάνιση συμβολοσειρών γίνεται χρησιμοποιώντας την μέθοδο println() η οποία πραγματοποιεί αλλαγή γραμμής μετα την εκτύπωση

Παράδειγμα εκτύπωσης:

```
System.out.println(ntua); //Αλλαγή γραμμής στην εκτύπωση
```

Ή εναλακτικά

```
System.out.println("National Technical University of Athens"); //Αλλαγή γραμμής στην
                                                                // εκτύπωση
```

Για την εμφάνιση συμβολοσειρών χωρίς αλλάγή γραμμής μετά την εκτύπωση χρησιμοποιείται η μέθοδος print()

Παράδειγμα εκτύπωσης:

```
System.out.print("National");
System.out.print(" Technical ");
System.out.println(" University ");
System.out.println(" of ");
System.out.println(" Athens ");
```

Παράδειγμα επικόλλησης συμβολοσειρών

```
String ntuaHMMY = ntua + " HMMY Department";
```

Η συμβολοσειρά ntuaHMMY περιέχει το:
"National Technical University of Athens HMMY Department".

Απλές μέθοδοι στις συμβολοσειρές:

Έστω οι συμβολοσειρές s1 και s2.

Μέθοδος	Ενέργεια
s1.length();	Προσδιορισμός μήκους (αριθμός χαρακτήρων) συμβολοσειράς s1
s2=s1.toUpperCase();	Μετατροπή συμβολοσειράς s1 σε κεφαλαία
s2=s1.toLowerCase();	Μετατροπή συμβολοσειράς s1 σε πεζά
s2.equals(s1)	Σύγκριση συμβολοσειρών s1 και s2
int a=s2.indexOf(s1);	Το a περιέχει την θέση της s2 στην s1
>>	Ολίσθηση δεξιά

3.5 Εντολές και Δομές ελέγχου

Οι εντολές είναι σχεδόν πανομοιότυπες με αυτές της C, με μόνη εξαίρεση ότι στη JAVA μπορούμε να κάνουμε δηλώσεις μεταβλητών όχι μόνο στην αρχή ενός block αλλά και σε οποιοδήποτε σημείο του, αρκεί η δήλωση να γίνει πριν από τη χρήση της εν λόγω μεταβλητής. Να σημειώσουμε επίσης ότι ισχύουν όλοι οι κανόνες εμβέλειας, τοπικών μεταβλητών και περάσματος παραμέτρων (σε μεθόδους) της C.

Όσον, τώρα, αφορά τις δομές ροής ελέγχου (if - else, for, while, do - while και switch), οι μόνες αλλαγές που έχουν γίνει σε σχέση με τη C είναι το ότι στο if-else, while, do-while οι εκφράσεις που αποτιμώνται προκειμένου να γίνει άλμα ή επανάληψη, θα πρέπει να είναι boolean και όχι αριθμητικές. Δηλαδή :

while (1) {...} // ΕΙΝΑΙ ΛΑΘΟΣ
while (true) {....} // ΕΙΝΑΙ ΣΩΣΤΟ

Στη συνέχεια θα παρουσιάσουμε συνοπτικά τις δομές ροής ελέγχου προγράμματος της Java. Υπάρχουν γενικά δύο ήδη δομών ελέγχου ροής (control flow):

- Οι δομές επιλογής και
- Οι δομές επανάληψης

Ο ακόλουθος πίνακας συνοψίζει αυτές τις δομές για τη Java

Είδος δομής ελέγχου ροής	Δομή ελέγχου ροής
Δομές επιλογής	if-else
	switch-case
Δομές επανάληψης	for
	while
	do-while

Πέρα από τις εντολές του πιο πάνω πίνακα ο έλεγχος ροής σε ένα πρόγραμμα Java μπορεί να μεταφερθεί και σε κάποιο άλλο σημείο εξαιτίας της πρόκλησης μιας εξαίρεσης. Αλλά για τις εξαιρέσεις (exceptions) και τους χειριστές τους (exception handlers) θα μιλήσουμε σε επόμενα μαθήματα.

Επίσης κάποιες άλλες εντολές πέρα από τις ίδιες τις δομές ελέγχου ροής που είδαμε προηγουμένως είναι οι: break, continue και return που μεταφέρουν το έλεγχο ροής σε άλλα σημεία του προγράμματος. Την χρήση αυτών των εντολών θα τη δούμε στη συνέχεια.

Η Java επιτρέπει και τη χρήση ετικετών αλλά δεν υποστηρίζει την εντολή goto. Αντί αυτής μπορούν και με τις ετικέτες να χρησιμοποιηθούν οι εντολές break και continue.

Η δομή επιλογής if-else

Η εντολή if μας βοηθάει στον έλεγχο μίας λογικής έκφρασης (της συνθήκης) και αν είναι αληθής εκτελούνται μία ή περισσότερες εντολές ενώ αν είναι ψευδής δεν εκτελούνται. Αν οι εντολές είναι περισσότερες από μία τότε θα πρέπει οι εντολές να περικλειστούν ανάμεσα από άγκιστρα (τα οποία ομαδοποιούν εντολές).
Το συντακτικό της εντολής if είναι το ακόλουθο:

```
if  (συνθήκη)
   εντολές
```

π.χ.

```
if (x!=0) {
   System.out.println("Το x δεν είναι 0");
   y = 1/x;
}
```

Η εντολή if έχει επίσης και τη φράση else με την οποία επιτρέπεται η εκτέλεση μίας ομάδας εντολών εναλλακτικά αν δεν ισχύει η συνθήκη. Φυσικά μπορούμε να έχουμε και εμφωλευμένα if δηλαδή if μέσα σε άλλα if όσες φορές θέλουμε.
Το συντακτικό της if επαυξημένο με τη φράση else είναι το ακόλουθο:

```
if (συνθήκη)
 ομάδα-εντολών-1
else
 ομάδα-εντολών-2
```

```
π.χ.
if (x!=0) {
    System.out.println("Το x δεν είναι 0");
    y=1/x;
}
else {
   System.out.println("Το x είναι 0");
   y=0;
}
```

Η δομή επιλογής switch-case

Όταν οι εναλλακτικές περιπτώσεις που πρέπει να ελέγξουμε με την if είναι πάρα πολλές και αφορούν τον έλεγχο για ισότητα της τιμής μιας μεταβλητής ή μιας έκφρασης με κάποιες τιμές προτιμάται η switch-case η οποία έχει την ακόλουθη γενική μορφή:

```
switch (έκφραση) {
    case τιμή-1:
        εντολές-1; [break;]
           …
    case τιμή-v:
        εντολές-v; [break;]
    [default:
        εντολές; [break;]]
}
```

Η switch-case αποτιμά την τιμή της έκφρασης που ελέγχεται και στη συνέχεια διατρέχει με τη σειρά όλες τις περιπτώσεις που δίνονται. Αν κάποια περίπτωση βρεθεί αληθής τότε εκτελούνται οι εντολές που δίνονται μετά την άνω-κάτω τελεία γι' αυτή τη περίπτωση. Αν καμία περίπτωση δεν βρεθεί αληθής τότε εκτελείται η default περίπτωση αν υπάρχει.
Προσοχή χρειάζεται στη χρήση του break που είναι μεν προαιρετική αλλά απαιτείται τις περισσότερες φορές, μια και αν δεν υπάρχει τότε θα εκτελεστούν και οι εντολές τις επόμενης περίπτωσης (χωρίς να ελεγχθεί η τιμή της) και πιθανώς και άλλων, μέχρι να βρεθεί το επόμενο break ή να τελειώσει η switch-case.
Παρόλα αυτά ενδέχεται να υπάρχουν κάποιες περιπτώσεις που αυτό θα ήταν βολικό, όπως δείχνει το ακόλουθο τμήμα κώδικα που υπολογίζει τις μέρες ενός μήνα ανάλογα με το ποιος μήνας είναι (η τιμή της μεταβλητής month) και το αν το έτος (year) είναι δίσεκτο:

```
...
switch (month) {
   case 1:
   case 3:
   case 5:
   case 7:
   case 8:
   case 10:
   case 12:
      numDays = 31;
      break;
   case 4:
   case 6:
   case 9:
   case 11:
      numDays = 30;
```

```
        break;
    case 2:
        if ( ((year % 4 == 0) && !(year % 100 == 0))
                || (year % 400 == 0) )
            numDays = 29;
        else
            numDays = 28;
        break;
}
...
```

Η δομή επανάληψης for

Η εντολή for είναι χρήσιμη για την επανάληψη μιας σειράς εντολών όταν είναι γνωστό εκ των προτέρων πόσες φορές θέλουμε να επαναληφθούν. Έτσι συνήθως το for ελέγχεται από ένα μετρητή ο οποίος μεταβάλλεται από μία αρχική τιμή μέχρι να ξεπεράσει μία τελική τιμή. Στη for επίσης καθορίζεται το πόσο θα μεταβάλλεται αυτός ο μετρητής σε κάθε βήμα.
Η γενική μορφή της for είναι η ακόλουθη:

```
for (αρχικοποίηση; τερματισμός; αύξηση)
    εντολές;
```

Για παράδειγμα το ακόλουθο τμήμα κώδικα εμφανίζει στην οθόνη τους αριθμούς από το 1 μέχρι το 10:

```
...
for (int i=1; i<=10; i++)
    System.out.println(i);
...
```

Στη φράση της αρχικοποίησης το i δηλώνεται (η εμβέλειά του είναι το for loop) και αρχικοποιείται στο1.
Στη φράση του τερματισμού το i ελέγχεται για το αν έχει ξεπεράσει το 10. Άρα ο συγκεκριμένος βρόχος θα επαναληφθεί μέχρι το i να ξεπεράσει το 10.
Στη φράση της μεταβολής το i αυξάνεται κατά 1. Αυτό σημαίνει ότι το i θα πάρει διαδοχικά τις τιμές 1, 2, …, 10
Σε κάθε βήμα της επανάληψης ελέγχεται η τιμή του i. Αν το i ξεπεράσει το 10 ο βρόχος τερματίζεται, αν όχι αυξάνεται κατά 1 και εκτελείται ξανά η μία και μοναδική εντολή αυτού του βρόχου.

Η δομή επανάληψης while

Η εντολή for που είδαμε προηγουμένως είναι κατάλληλη όταν γνωρίζουμε πόσες επαναλήψεις θα γίνουν. Αν δεν γνωρίζουμε πόσες επαναλήψεις θα γίνουν μία καλύτερη εντολή είναι η while.
Η while έχει την ακόλουθη γενική μορφή:

```
while (συνθήκη)
    εντολές;
```

Η εντολή ή εντολές που ακολουθούν το while θα εκτελεστούν όσο η συνθήκη είναι αληθής. Ο βρόχος δηλαδή θα τερματιστεί όταν η συνθήκη - που είναι μία boolean έκφραση - γίνει ψευδής

Στο ακόλουθο τμήμα κώδικα ελέγχεται αν η τιμή x βρίσκεται μέσα στο πίνακα numbers. Κάνουμε δηλαδή μία σειριακή αναζήτηση στα στοιχεία του πίνακα. Αν το στοιχείο βρεθεί η μεταβλητή found γίνεται true και βγαίνουμε από το βρόχο. Επίσης από το βρόχο θα βγούμε αν ελεγχθούν και τα δέκα στοιχεία του πίνακα και δεν έχει βρεθεί ακόμα το x.

```
…
boolean found = false;
```

```
int i=0;
while (!found && i!=10)
    if (numbers[i++] == x)
        found = true;
…
```

Η δομή επανάληψης do-while

Υπάρχουν κάποιες περιπτώσεις στις οποίες θα θέλαμε οι εντολές μέσα στο βρόχο να εκτελεστούν τουλάχιστον μία φορά και στη συνέχεια να ελεγχθεί η συνθήκη εξόδου. Σ' αυτές τις περιπτώσεις προτιμάται η χρήση της do-while αντί της while.
Η do-while έχει την ακόλουθη γενική μορφή:

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

Στην do-while πρώτα εκτελούνται οι εντολές και στη συνέχεια ελέγχεται η συνθήκη. Ο βρόχος τερματίζεται αν η συνθήκη βρεθεί ψευδής.
Η do-while δεν χρησιμοποιείται πολύ συχνά αλλά έχει κι αυτή τις χρήσεις της.

Για παράδειγμα όταν διαβάζουμε χαρακτήρες από ένα αρχείο μέχρι να διαπιστώσουμε το τέλος του αρχείου θα πρέπει να διαβάσουμε τουλάχιστον ένα χαρακτήρα, όπως δείχνει και το ακόλουθο τμήμα κώδικα:

```
...
int c;
Reader in;
...
do {
   c = in.read();
   ...
} while (c != 1);
...
```

Εντολές διακλάδωσης

Η Java έχει τρεις εντολές διακλάδωσης οι οποίες είναι βολικές σε αρκετές περιπτώσεις και ιδιαίτερα με τις επαναληπτικές δομές που έχουμε συζητήσει.
Οι εντολές διακλάδωσης είναι οι: break, continue και return
Την εντολή break ήδη την είδαμε σε μία χρήση της με την switch-case. Μία ακόμα αρκετά συνήθη χρήση της break είναι η χρήση της για άμεση έξοδο από κάποιο βρόχο.

Για παράδειγμα η σειριακή αναζήτηση που είδαμε με το while θα μπορούσε να γίνει όπως στο For.java ως εξής:

```
int i;
boolean found = false;
for (i=0; i<10; i++)
   if (numbers[i]==x) {
      found = true;
      break;
   }
```

Η χρήση της continue μέσα σε ένα βρόχο προκαλεί την άμεση αποτίμηση και πάλι της συνθήκης εξόδου. Για να γίνει αυτό στις εντολές for και while ο έλεγχος μεταφέρεται στη πρώτη γραμμή του βρόχου ενώ στο do-while στη τελευταία. Στη συνέχεια αποτιμάται και πάλι η συνθήκη τερματισμού και η επαναληπτική διαδικασία τερματίζεται ή συνεχίζεται ανάλογα με την τιμή της συνθήκης (true ή false) ως συνήθως.

Οι εντολές break και continue έχουν και μία μορφή για ετικέτες (labeled break και labeled continue) που δεν θα συζητήσουμε εδώ μια και στη πράξη οι ετικέτες δεν χρησιμοποιούνται αφού οι υπόλοιπες δομές που είδαμε αρκούν για όλες τις πιθανές χρήσεις, χωρίς να εμφανίζουν πολλά από τα προβλήματα που πιθανώς να εμφανιστούν με την χρήση των ετικετών.

Η εντολή return διακόπτει αμέσως την εκτέλεση της μεθόδου μέσα στην οποία βρίσκετε και σε περίπτωση που η μέθοδος επιστρέφει κάποια τιμή, επιστρέφει την τιμή που βρίσκεται στα δεξιά της. Ο τύπος της επιστρεφόμενης τιμής πρέπει να είναι ίδιος με τον τύπο-αποτελέσματος που δηλώθηκε στην δήλωση της μεθόδου.

Στην ακόλουθη μέθοδο η εντολή return επιστρέφει ένα String που συγκεκριμένα την τιμή που περιέχεται στην μεταβλητή name. Η εκτέλεση της μεθόδου getName( ) τερματίζει ακόμα και αν υπάρχουν και άλλες εντολές κάτω από την εντολή: **return name;**

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

Στην παρακάτω μέθοδο η εντολή: **return interest;** τερματίζει την μέθοδο και επιστρέφει τον ακέραιο που περιέχει η μεταβλητή interest;

```
public int calculateInterest( )
{
        ...     return interest;
}
```

3.6 Πίνακες

Οι πίνακες στην JAVA δηλώνονται ως εξής:

<Τύπος ή κλάση> [] <μεταβλητή>;

ή

<Τύπος ή κλάση> <μεταβλητή>[];

Παραδείγματα

```
int[]   nums;
byte    buff[];
float   matrix[][];     //Διδιάστατος πίνακας
A       a[];
B[]     b;
```

Όπως βλέπουμε ο πίνακας δεν δημιουργείται κατά την δήλωσή του (αφού άλλωστε δεν του έχουμε προσδιορίσει το μέγεθος που θα έχει). Για την κατασκευή του πίνακα κάνουμε τα εξής :

<μεταβλητή πίνακα> = new <τύπος ή κλάση> [<μέγεθος>];

όπου το <μέγεθος> είναι τύπου byte, short, int ή long.

Παραδείγματα

```
int[]   K;
K = new int[200];
byte    buff[] = new byte[1024];
A[]     a = new A[4];
```

```
float      matrix[][];
matrix = new float[30][];
matrix = new float[0][40]; ...  matrix = new float[29][40];
```
ή μπορούμε και απευθείας να δηλώσουμε:
```
matrix = new float[30][40];
```

Αναφορά σε στοιχείο του πίνακα :
```
a[2] = 5;
matrix[0][8] = (float) (buff[19] + buff[25]);
```

Όταν ένας πίνακας είναι μεγέθους N τότε τα έγκυρα indexes θα είναι από 0 έως N-1. Κάθε άλλη τιμή θα προκαλέσει compile-time ή run-time error.

Οι μέθοδοι επιτρέπεται να παίρνουν σαν όρισμα ένα πίνακα ή να επιστρέφουν ένα πίνακα. Αυτό γίνεται ως εξής :
```
int[]  give_ints() {
        int        I[] = new int[3];
        I[0] = I[1] = I[2] = 7;
        return I;
}
```
ή
```
int give_ints() [] {....}

void get_ints(int[] i) {
        i[0] = i[1]+i[2];
}
```
ή
```
void get_ints(int i[]) { ... }
```

Σε αυτό το σημείο να πούμε ότι στην JAVA ένας πίνακας είναι ένα αντικείμενο. Η κλάση ενός τέτοιου αντικειμένου (πίνακα) δημιουργείται αυτόματα από την γλώσσα και είναι υποκλάση της κλάσης Array. Δηλαδή όταν ορίζουμε μία κλάση, η JAVA ορίζει αυτόματα μία νέα κλάση, (υποκλάση της Array), η οποία θα δίνει πίνακες με στοιχεία αντικείμενα της κλάσης που εμείς ορίσαμε.

ΠΡΟΣΟΧΗ : όταν γράφουμε int a[]; τότε το αντικείμενο - πίνακας θα είναι το a. Αντίθετα το στοιχείο a[0], ... κλπ θα είναι απλοί int που περιέχει ο πίνακας όπως ακριβώς τους ξέρουμε.

Τέλος κάθε πίνακας περιέχει μια μεταβλητή length η οποία δίνει το μέγεθος του πίνακα. Αυτή η μεταβλητή χρησιμοποιείται ως εξής :
```
a.length            //η δήλωση αυτή επιστρέφει το μέγεθος του πίνακα
if  (a.length < 1024) {...}
for (int i=0; i<a.length; i++) {
        System.out.println(a[i]);
}
```

4. Εξαιρέσεις *(Exceptions)*

Όταν σε ένα πρόγραμμα σε JAVA συμβεί κάποιο λάθος, για παράδειγμα περαστεί κάποια λάθος παράμετρος, τότε ο κώδικας που θα το ανιχνεύσει μπορεί να *εγείρει* (throw=πετάω) μία εξαίρεση! Η έγερση εξαίρεσης θα έχει ως αποτέλεσμα τον τερματισμό του thread στο οποίο συνέβη το λάθος και θα τυπωθεί κάποιο μήνυμα. Ωστόσο τα προγράμματα μπορούν να ορίσουν *χειριστές εξαιρέσεων (exception handlers)* οι οποίοι θα *πιάνουν* την εξαίρεση και θα φροντίζουν ώστε το πρόγραμμα να ανανήψει από το λάθος.

Μερικές από τις εξαιρέσεις προκαλούνται από το runtime system, όπως στην περίπτωση διαίρεσης με το μηδέν. Ωστόσο, οποιαδήποτε κλάση μπορεί να ορίσει και να εγείρει δικές της εξαιρέσεις. Αυτό γίνεται ως εξής. Πρώτα δημιουργεί (με new) ένα αντικείμενο εξαίρεσης το οποίο θα πρέπει να είναι στιγμιότυπο της κλάσης Exception ή κάποιας υποκλάσης της. Έπειτα με την εντολή **throw** και το αντικείμενο εξαίρεσης προκαλείται exception. Η εκτέλεση του κώδικα θα διακοπεί στην εντολή throw και ο υπόλοιπος κώδικας που ακολουθεί δεν θα εκτελεστεί. Επίσης η μέθοδος μέσα στην οποία συνέβη η εξαίρεση δεν θα επιστρέψει κάποια τιμή. Το αντικείμενο, τώρα, της εξαίρεσης, (η εξαίρεση ουσιαστικά), θα δοθεί στον κατάλληλο exception handler, απ' όπου και συνεχίζεται η εκτέλεση του προγράμματος. Στο παρακάτω παράδειγμα φαίνεται το πώς δημιουργείται και εγείρεται μία εξαίρεση.

```
class MyException extends Exception {
}

class MyClass {
        void oops() {
                if (/* no error occurred */) {
                        ...
                } else { /* error occurred */
                        throw new MyException();
                }
        }
}
```

Για να ορίσουμε ένα χειριστή εξαιρέσεων (exception handler) θα πρέπει να *κλείσουμε* τον κώδικα που μπορεί να προκαλέσει την εξαίρεση μέσα σε μια εντολή try. Μετά την try θα πρέπει να βάλουμε μία ή περισσότερες εντολές catch. Κάθε catch θα μπορεί να *πιάνει,* μία μόνο κλάση εξαίρεσης (πχ MyException). Επίσης σε κάθε catch θα υπάρχει ο κατάλληλος κώδικας για τον χειρισμό της εξαίρεσης. Για παράδειγμα :

```
try {
        p.a = 10;
} catch (NullPointerException e) {
        println("p was null");
} catch (Exception e) {
        println("other error occurred");
} catch (Object obj) {
        println("Who threw that object?");
}
```

Η πρώτη εντολή catch με παράμετρο που ταιριάζει στην εξαίρεση θα εκτελεστεί. Έπειτα το πρόγραμμα θα συνεχίσει μετά τις εντολές try/catch. Ωστόσο δεν είναι δυνατό η εκτέλεση του προγράμματος να συνεχιστεί από το σημείο όπου συνέβη η εξαίρεση.

Οι χειριστές εξαιρέσεων μπορούν να φωλιαστούν επιτρέποντας έτσι ο χειρισμός να γίνει σε περισσότερα του ενός σημεία. Αυτό είναι χρήσιμο όταν ο πρώτος exception handler δεν μπορεί να διορθώσει πλήρως το λάθος. Προκειμένου, τώρα, να περαστεί ο χειρισμός της εξαίρεσης στον επόμενο, (υψηλότερου επιπέδου), exception handler, θα πρέπει να χρησιμοποιήσουμε την throw και για αντικείμενο εξαίρεσης θα δώσουμε την εξαίρεση που πιάσαμε. Σημειώστε ότι έπειτα από το throw η εκτέλεση του τρέχοντος κώδικα χειρισμού διακόπτεται οριστικά.

```
try {
        f.open();
} catch(Exception e) {
        f.close();
        throw e;
}
```

Υπάρχουν περιπτώσεις όπου θέλουμε ένα κομμάτι κώδικα να εκτελείται πάντα, ανεξάρτητα από το αν θα συμβεί κάποια εξαίρεση ή όχι. Αυτό το επιτυγχάνουμε με την εντολή **finally**. Το παρακάτω παράδειγμα εξηγεί τη χρήση της.

```
try {
        // do something
} finally {
        // clean up after it
}
```

είναι παρόμοιο με :

```
try {
        // do something
} catch(Object e){
        // clean up after it
        throw e;
}
// clean up after it
```

Η εντολή finally εκτελείται ακόμα και όταν το try block περιέχει τις εντολές return, break, continue ή throw. Για παράδειγμα :

```
try {
        if (a == 10) {
                return;
        }
} finally {
        print("finally\n");
}
print("after try\n");
```

Αυτό το κομμάτι κώδικα θα τυπώνει πάντα το μήνυμα "finally" αλλά το μήνυμα "after try" μόνο όταν είναι a != 10.

5. Τα βασικά των Applets στην Java

Στην Java τα Applets εκτελούνται μέσα από κάποιον Java WWW Browser. Η αναφορά σε ένα Applet γίνεται σε μια WEB σελίδα μέσω ενός ειδικού HTML tag. Όταν ο χρήστης σηκώσει σε κάποιον Browser μια WEB σελίδα που περιέχει κάποιο Applet, ο Browser κατεύαζει το Applet από τον Web Server και το εκτελεί στον τοπικό υπολογιστή.
Επειδή τα Java Applets τρέχουν μέσα από κάποιον Java Browser, έχουν το πλεονέκτημα της δομής που παρέχει ο Browser: ένα υπάρχον παράθυρο, έννοιες γραφικών και γεγονότων, και το interface που τα περιβάλει. Επιπλέον επειδή τα Applets μπορούν να κατεβούν από οπουδήποτε και να εκτελουνται τοπικά στον υπολογιστή του χρήστη, υπάρχουν περιορισμοί που εμποδίζουν τα Applets να προκαλέσουν ζημιά στο τοπικό σύστημα όπως:

- Τα Applets δεν μπορούν να γράψουν ή να διαβάσουν στο τοπικό σύστημα αρχείων, εκτός από καταλόγους που πρέπει να έχει προκαθορίσει ο τοπικός χρήστης.
- Τα Applets μπορούν να επικοινωνήσουν μόνο με τον Server στον οποίο το Applet είχε αποθηκευτεί.
- Τα Applets δεν μπορούν να τρέξουν προγράμματα που υπάρχουν στο σύστημα του τοπικού χρήστη.

Δημιουργώντας Applets
Για να δημιουργηθεί ένα Applet πρόγραμμα, πρέπει να δημιουργηθεί μια υποκλάση της κλάσης Applet, του java.applet πακέτου:

```
Public class myClass extends java.applet.Applet{
. . . .
}
```

Η κλάση Applet παρέχει συμπεριφορά που επιτρέπει στο Applet πρόγραμμα όχι μόνο να λειτουργεί μέσα στον Browser αλλά να έχει και δυνατότητες AWT για ενσωμάτωση User Interface στοιχείων, διαχείριση γεγονότων ποντικιού και πληκτρολογίου, καθώς και ζωγραφικής στην οθόνη. Παρόλο που ένα Applet πρόγραμμα μπορεί να αποτελείται από επιπλέον βοηθητικές κλάσεις, η υποκλάση της κλάσης Applet είναι αυτή που ενεργοποιεί την εκτέλεση του Applet προγράμματος.

5.1 Βασικές λειτουργίες των Applets

Για την δημιουργία Java εφαρμογών, η κλάση της εφαρμογής πρέπει να διαθέτει την μέθοδο main(). ¨Οταν η εφαρμογή αρχίζει να τρέχει, εκτελείται η μέθοδος main() η οποία καθορίζει την συμπεριφορά του προγράμματος. Αντιθέτως στα Applets προγράμματα υπάρχουν διαφορετικές λειτουργίες που αντιστοιχούν σε γεγονότα που συμβαίνουν κατά τη διάρκεια της ζωής του Applet (πχ. γεγονότα αρχικοποίησης, ζωγραφικής, ποντικιού κλπ). Σε κάθε λειτουργία αντιστοιχεί και κάποια μέθοδος η οποία καλείται από τον Browser όταν ένα γεγονός συμβεί. Η μέθοδοι των λειτουργιών έτσι όπως ορίζονται στην Applet κλάση της Java δεν κάνουν τίποτε. Για να δώσουμε κάποια συμπεριφορά σε κάποιο γεγονός της Applet εφαρμογή μας πρέπει να ξαναορίσουμε την μέθοδο που αντιστοιχεί στο γεγονός (τεχνική method overriding) μέσα στην υποκλάση της Applet που δημιουργήσαμε. Οι πέντε πιο βασικές μέθοδοι μιας applet είναι:
1.

```
public void init(){
. . .
}
```

Η μέθοδος αυτή εκτελείται όταν η applet εφαρμογή κατεβαίνει στο τοπικό υπολογιστή για εκτέλεση. Η μέθοδος αυτή μπορεί να περιλαμβάνει την δημιουργία κάποιων αντικειμένων, τον καθορισμό παραμέτρων, το φόρτωμα εικόνων ή font κλπ.
2.

```
public void start(){
. . .
}
```

Η μέθοδος αυτή καλείται αμέσως μετά την init(). Η start() καλείται επίσης όταν η applet εφαρμογή είχε προηγουμένως σταματήσει την εκτέλεσή της. Για παράδειγμα μια applet εφαρμογή σταματά την εκτέλεσή της όταν ο χρήστης αλλάξει μέσω ενός συνδέσμου HTML σελίδα και ξαναρχίζει την εκτέλεσή της όταν ο χρήστης γυρίσει πίσω στη σελίδα της Applet εφαρμογής.

3.

```
public void stop(){
. . .
}
```

Η stop() σταματά την εκτέλεση της Applet εφαρμογής και είναι το συμπλήρωμα της start(). Ο χρήστης μπορεί επίσεις να καλέσει από μόνος του την stop για να σταματήσει την Applet εφαρμογή. Στην stop() μπορεί επίσης ο χρήστης να σταματά την εκτέλεση των Threads της Applet εφαρμογής (πράγμα που δεν γίνεται αυτόματα όταν αυτή σταματά να εκτελείται) και να τα ξανααρχίζει όταν αυτή αρχίζει πάλι την εκτέλεσή της.

4.

```
public void destroy(){
. . .
}
```

Δίνει τη δυνατότητα στη Applet εφαρμογή να ελευθερώσει τους πόρους του συστήματος που της είχαν διατεθεί (πχ αντικείμενα, threads κλπ.). Η μέθοδος εκτελείται λίγο πριν η Applet εφαρμογή πάψει οριστικά την εκτέλεσή της ή όταν ο Browser κλείσει. Συνήθως η μέθοδος αυτή δεν χρειάζεται να οριστεί στην Applet εφαρμογή εκτός από πολύ ειδικές περιπτώσεις.

5.

```
public void paint(Graphics g){
. . .
}
```

Με την μέθοδο αυτή η Applet εφαρμογή εμφανίζει κάτι στην οθόνη πχ. κείμενα, γραφικά, εικόνες. Η μέθοδος αυτή καλείται σε διάφορες περιπτώσεις όπως όταν η Applet εφαρμογή αρχικοποιήται, όταν μετακινήται ο Browser ή τοποθετήται πίσω από άλλο παράθυρο και μετά έρχεται πάλι μπροστά, όταν θέλουμε να δημιουργήσουμε animation οπότε και καλείται συνεχώς. Για να χρησιμοποιήσουμε την μέθοδο αυτή πρέπει προηγουμένως η κλάση των γραφικών να περιληφθεί στον κώδικα της Applet εφαρμογής - αυτό γίνεται μέσω της δήλωσης:

```
import java.awt.Graphics
```

5.2 Απλές Applets και η εισαγωγή τους σε σελίδες Web

Ένα παράδειγμα εφαρμογής Applet ακολουθεί:

```
import java.awt.Graphics
import java.awt.Font
import java.awt.Color

public class HelloSomeoneApplet extends java.applet.Applet{
        Font f = new Font("TimesRoman", Font.Bold, 36);
        String myname;

        public void init(){
            myname = getParameter("name");
            if (myname == null)
                    myname="Laura";
            myname = "Hello" + myname + "!";
        }

        public void paint(Graphics g){
            g.setFont(f);
            g.setColor(Color.red);
            g.drawString("Hello again!", 5,50);
```

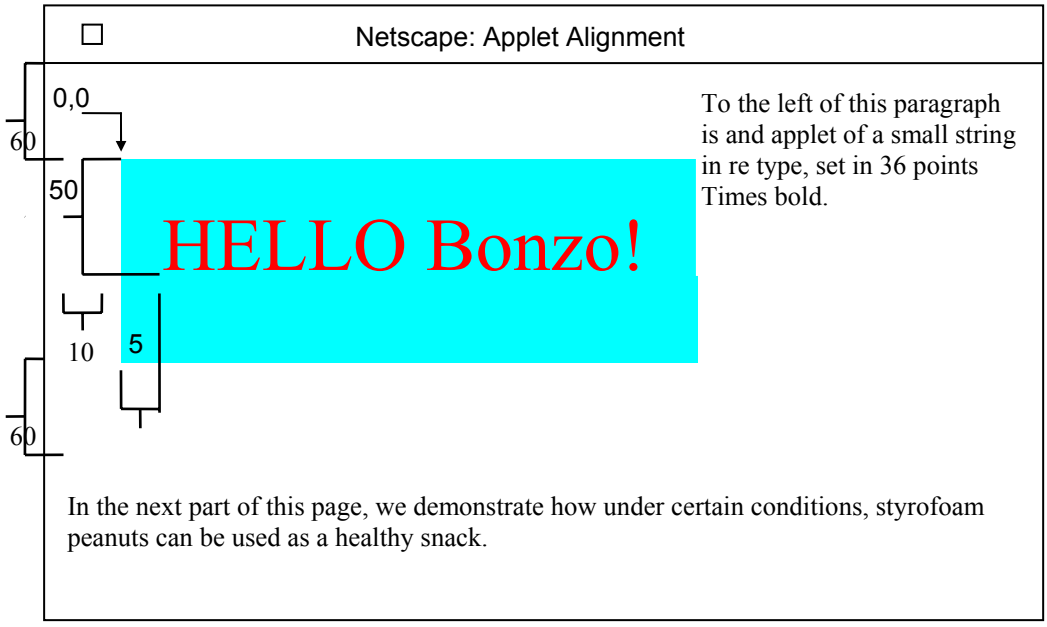
```
        }
}
```

Ο κώδικας έχει σωθεί σε ένα αρχείο με όνομα HelloAgainApplet.class στον κατάλογο /classes.

Το HTML αρχείο που περιέχει την Applet έχει τον παρακάτω κώδικα:

```
<HTML>
<HEAD>
<TITLE>Applet Alignement </TITLE>
</HEAD>
<BODY>
<P>
<APPLET CODE="HelloAgainApplet.class"  CODEBASE="classes"
 WIDTH=300 HEIGHT=70
ALIGN= LEFT VSPACE=60 HSPACE 10>
<PARAM NAME=name VALUE="Bonzo">
Hello to whoever you are
</APPLET>
To the left of this paragraph is and applet of a small string in re type, set in 36 points Times bold.
<BR CLEAR = ALL>
<P>
In the next part of this page, we demonstrate how under certain conditions, styrofoam peanuts can be used as a healthy snack.
</BODY>
</HTML>
```

Το αποτέλεσμα των παραπάνω σε έναν Web Browser θα είχε την ακόλουθη εμφάνιση:



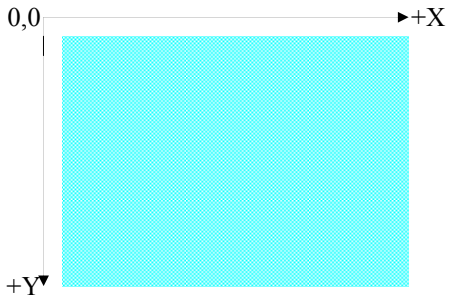
Σχόλια:

- Ο κώδικας της Applet εφαρμογής είναι κατανοητός, αξίζει ίσως να αναφερθεί ότι η εφαρμογή δέχεται ως τιμή στην string μεταβλητή myname την τιμή της παραμέτρου name που ορίστηκε μέσα στο HTML αρχείο που καλεί την εφαρμογή.
- Η ενσωμάτωση σε μιας Applet εφαρμογής σε σελίδα του Web γίνεται με το tag <APPLET>
- BR CLEAR: Το κείμενο που ακολουθεί μετά από το </APPLET> tag και το BR CLEAR εμφανίζεται στο επόμενο καθαρό αριστερό ή δεξί περιθώριο ή στην επόμενη γραμμή ανάλογα με τις τιμές που έχει δηλωθεί στο CLEAR (CLEAR=LEFT, CLEAR=RIGHT, CLEAR=ALL).

- Μέσα στα tags <APPLET> μπορούν να περιληφθούν διάφορα αναγνωριστικά που έχουν να κάνουν με τον τρόπο εμφάνισης της Applet εφαρμογής στη Web σελίδα:
  - CODE: Δηλώνει το όνομα της κλάσης που περιέχει την Applet εφαρμογή.
  - CODEBASE: Δηλώνει τον κατάλογο που βρίσκεται η κλάση που περιέχει την Applet εφαρμογή.
  - WIDTH και HEIGHT: Δηλώνουν τις διαστάσεις της περιοχής όπου το αποτέλεσμα της Applet εφαρμογής θα εμφανιστεί.
  - ALIGN: Δηλώνει την στοίχιση της Applet εφαρμογής στη σελίδα Web. Παίρνει εννέα τιμές (LEFT, RIGHT, TOP, TEXTTOP, MIDDLE, ABSMIDDLE, BASELINE, BOTOM, ABSBOTOM). Για παράδειγμα η δήλωση ALIGN=LEFT στοιχίζει την Applet εφαρμογή στα αριστερά της σελίδας και ότι κείμενο ακολουθεί μετά το </APPLET> γράφεται στο κενό που υπάρχει στα δεξιά της.
  - VSPACE και HSPACE: Η κατακόρυφη και η οριζόντια αντίστοιχα απόσταση σε pixels της Applet εφαρμογής από το κείμενο που την περιβάλλει.
  - PARAM :Δηλώνει την παράμετρο που δέχεται η Applet εφαρμογή, στο αναγνωριστικό Name ορίζεται το όνομα της παραμέτρου ενώ στο VALUE η τιμή της.

5.3 Γραφικά, Γραμματοσειρές και Χρώμα

Η σχεδίαση ενός αντικειμένου στην οθόνη, γίνεται μέσω μεθόδων της κλάσης Graphics. Όλες οι μέθοδοι σχεδίασης παίρνουν ορίσματα που αφορούν σημεία, γωνίες ή αρχικά σημεία των αντικειμένων στο σύστημα συντεταγμένων μιας applet. Το σύστημα συντεταγμένων έχει την αρχή του (0,0) στην αριστερή άνω γωνία. Οι θετικές τιμές του χ βρίσκονται στα δεξιά και οι θετικές ψ τιμές είναι προς τα κάτω.



5.3.1 Η κλάση Graphics

Για να χρησιμοποιηθούν μέθοδοι των γραφικών της κλάσης Graphics πρέπει ο κώδικάς της να περιληφθεί στον κώδικα της applet εφαρμογής. Αυτό γίνεται όπως αναφέρθηκε και παραπάνω με την δήλωση:

```
import java.awt.Graphics
```

Οι μέθοδοι που περιλλαμβάνει είναι:

Μέθοδοι για σχεδίαση
drawLine(startX, startY, width, height);
drawRect(startX, startY, width, height);
fillRect(startX, startY, width, height);
drawRoundRect(startX, startY, width, height);
fillRoundRect(startX, startY, width, height);
draw3DRect(startX, startY, width, height, boolean value);
fillRect(startX, startY, width, height, boolean value);
drawPolygon(arrayX, arrayY, array length);
fillPolygon(arrayX, arrayY, array length);
drawPolygon(polygon object);
fillPolygon(polygon object);
drawOval(TopCornerX, TopCornerY, width, height);

fillOval(TopCornerX, TopCornerY, width, height);
drawArc(TopCornerX, TopCornerY, width, height, arcStart, arcStop);
fillArc(TopCornerX, TopCornerY, width, height, arcStart, arcStop);

Μέθοδος αντιγραφής - καθαρισμού περιοχών
copyArea(fromStartX, fromStartY, fromWidth, fromHeight, toStartX, toStartY);
clearArea(fromStartX, fromStartY, fromWidth, fromHeight, toStartX, toStartY);

Μέθοδοι σχεδίασης χαρακτήρων
Για την σχεδίαση χαρακτήρων στην οθόνη πρώτα πρέπει να οριστεί η γραμματοσειρά που θα χρησιμοποιηθεί μέσω της μεθόδου:
setFont(fontObject);
Στη συνέχεια οι χαρακτήρες και strings μπορούν να σχεδιαστούν μέσω της μεθόδου:
drawString(aString, atPointX, atPointY);

Μέθοδος ανεύρεσης της γραμματοσειράς
getFont();

Μέθοδοι χρώματος αντικειμένου
setColor(colorObject);
getColor();

5.3.2 Η κλάση Fonts

Για να χρησιμοποιηθούν μέθοδοι των γραμματοσειρών της κλάσης Fonts πρέπει ο κώδικάς της να περιληφθεί στον κώδικα της applet εφαρμογής. Αυτό γίνεται με την δήλωση:

```
import java.awt.Fonts
```

Με τον κατασκευαστή της κλάσης Font δημιουργήσουμε αντικείμενα τύπου Font όπως για παράδειγμα:

```
Font f = new Font("TimesRoman", Font.BOLD, 24);
```

Οι τιμές που μπορεί να πάρει ο τύπος της γραμματοσειράς είναι Font.PLAIN, Font.BOLD, Font.ITALIC καθώς και συνδυασμοί αυτών (πχ FontBold + FontItalic).

Οι μέθοδοι που περιλαμβάνει η κλάση Font είναι:

```
getName();
getSize();
getStyle();
isPlain;
isBold;
isItalic();
```

Για περισσότερες πληροφορίες σχετικά με την ποιότητα της γραμματοσειράς που ορίζεται (πχ ύψος, μήκος των χαρακτήρων) μπορούν να χρησιμοποιηθούν μέθοδοι της κλάσης FontMetrics (βλέπε το API Java Decumentation της Sun).

5.3.3 Η κλάση Color

Για να χρησιμοποιηθούν μέθοδοι της κλάσης Color πρέπει ο κώδικάς της να περιληφθεί στον κώδικα της applet εφαρμογής. Αυτό γίνεται με την δήλωση:

```
import java.awt.Color
```

Με τον κατασκευαστή της κλάσης Color μπορούμε να δημιουργήσουμε αντικείμενα τύπου Color:

```
Color c = new Color(redValue, greenValue, blueValue);
```

Οι τιμές του κόκκινου, πράσινου, μπλε είναι ακέραιοι που κυμαίνονται από 0 έως 255. Επίσης στην κλάση αυτή έχουν οριστεί στάνταρ αντικείμενα τύπου Color αποθηκευμένα σε μεταβλητές της κλάσης, που επιτρέπουν να πάρουμε απευθείας Color αντικείμενα των πιο γνωστών χρωμάτων (Color.white, Color.black, Color.lightGray, Color.gray, Color.darkGray, Color.red, Color.green, Color.blue, Colror.yellow, Color.magenta, Color.cyan, Color.pink, Color.orange).

Τα χρώματα του Background και του Foreground μιας applet εφαρμογής καθορίζονται μέσω των μεθόδων της κλάσης Applet :

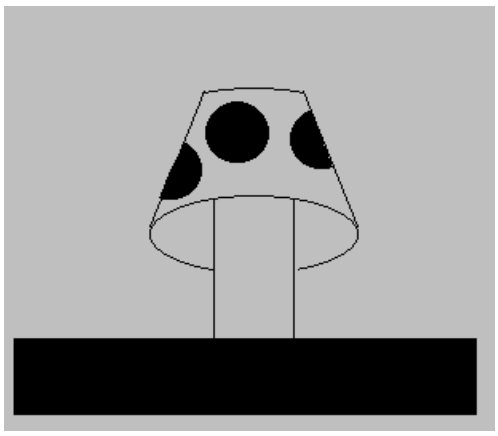
```
setBackground(colorObject);
setForeground(colorObject);
getBackground();
getForeground();
```

5.3.4 Παραδείγματα γραφικών - γραμματοσειρών - χρωμάτων

Στο σημείο αυτό παραθέτουμε τον κώδικα τριών εφαρμογών applet που σχεδιάζουν στην οθόνη α)μια λάμπα (παράδειγμα γραφικών), β)τέσσερις προτάσεις διαφορετικού τύπου γραμματοσειρών η κάθε μία (παράδειγμα γραμματοσειρών) και γ)έναν πίνακα από τετράγωνα τυχαίου χρώματος το καθένα (παράδειγμα χρωμάτων).

Α) Παράδειγμα γραφικών

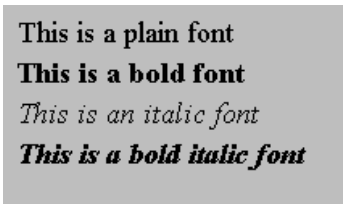
```
import java.awt.Graphics;

public class Lamp extends java.applet.Applet{

    public void paint (Graphics g){
        // the lamp platform
        g.fillRect(0,250,290,290);

        // the base of the lamp
        g.drawLine(125, 250, 125, 160);
        g.drawLine(175,250,175,160);

        // the lamp shade, top and bottom edges
        g.drawArc(85,157,130,50,-65,312);
        g.drawArc(85,87,130,50,62,58);

        // lamp shade, shides
        g.drawLine(85,177,119,89);
        g.drawLine(215, 177,181,89);

        //dots on the shade
        g.fillArc(78,120,40,40,63,-174);
        g.fillOval(120,96,40,40);
        g.fillArc(173,100,40,40,110,180);
    }
}
```



Β) Παράδειγμα γραμματοσειρών

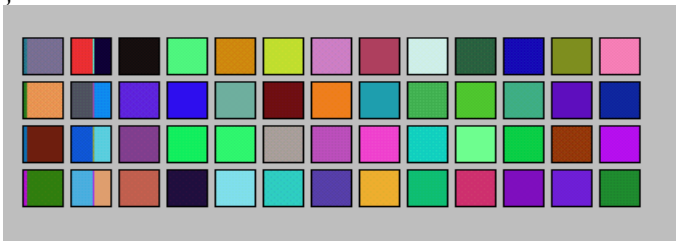
```
import java.awt.Font;
import java.awt.Graphics;
```

```
public class ManyFonts extends java.applet.Applet{

    public void paint (Graphics g){
        Font f = new Font("TimesRoman", Font.PLAIN, 18);
        Font fb = new Font("TimesRoman", Font.BOLD, 18);
        Font fi = new Font("TimesRoman", Font.ITALIC, 18);
        Font fbi = new Font("TimesRoman", Font.BOLD + Font.ITALIC, 18);

        g.setFont(f);
        g.drawString("This is a plain font", 10, 25);
        g.setFont(fb);
        g.drawString("This is a bold font", 10, 50);
        g.setFont(fi);
        g.drawString("This is an italic font", 10, 75);
        g.setFont(fbi);
        g.drawString("This is a bold and italic font", 10, 100);
    }
}
```



Γ) Παράδειγμα χρωμάτων

```
import java.awt.Graphics;
import java.awt.Color;

public class ColorBoxes extends java.applet.Applet{

    public void paint (Graphics g){
            int rval, gval, bval;

            for(int j = 30; j< (size().height -25); j += 30)  //rows
                for (int I = 5; I < (size().width -25); I += 30){  // columns
                    rval = (int)Math.floor(Math.random() * 256);
                    gval = (int)Math.floor(Math.random() * 256);
                    bval = (int)Math.floor(Math.random() * 256);

                    g.setColor(new Color(rval,gval,bval));
                    g.fillRect(I, j, 25, 25);
                    g.setColor(Color.Black);
                    g.fillRect(I-1, j-1, 25, 25);
                }
    }
}
```



6. Threads

Πολλές γλώσσες προγαμματισμού όπως και η Java διαθέτουν εργαλεία για την υλοποίηση threads στα προγράμματά τους. Αυτές οι γλώσσες καλούνται multithreading languages. Στον παραδοσιακό προγραμματισμό όταν ένα πρόγραμμα εκτελείται ονομάζεται process (διεργασία) και οι εντολές του εκτελούνται σειριακά η μία μετά την άλλη μέχρι το τέλος του. Σ' αυτή την περίπτωση μπορούμε να πούμε οτι το συγκεκριμένο process διαθέτει μόνο ένα thread που εκτελεί τις εντολές του προγράμματος. Στις multithreading γλώσσες προγραμματισμού μπορούμε να ορίσουμε σε κάποιο process να περιέχει ένα ή και περισσότερα threads (ονομάζονται και lightweight processes) οι οποίες εκτελούν κάθε μια ξεχωριστά τον κώδικα του προγράμματος. Είναι δηλαδή σαν να έχουμε πολλά processes που εκτελούνται παράλληλα μέσα στο αρχικό process.

6.1 Παράδειγμα με Threads

Το παρακάτω πρόγραμμα ορίζει δυο κλάσεις την SimpleThread και την TwoThreadsTest:

```
class SimpleThread extends Thread {
   public SimpleThread(String str) {
      super(str);
   }
   public void run() {
      for (int i = 0; i < 10; i++) {
         System.out.println(i + " " + getName());
         try {
            sleep((int)(Math.random() * 1000));
         } catch (InterruptedException e) {}
      }
      System.out.println("DONE! " + getName());
   }
}


class TwoThreadsTest {
   public static void main (String[] args) {
      new SimpleThread("Jamaica").start();
      new SimpleThread("Fiji").start();
   }
}
```

Η πρώτη μέθοδος της simpleThread κλάσης είναι ένας constructor που δέχεται ως όρισμα ένα string. Ο cunstructor υλοποιείται καλώντας έναν από τους constructors της κλάσης Thread που δέχεται ως όρισμα ένα String και ορίζει και το όνομα του Thread που θα χρησιμοποιήσουμε αργότερα.
Η επόμενη μέθοδος της κλάσης είνα η run. Η run μπορεί να θεωρείθεί ως η καρδιά ενός thread και εκεί ορίζονται οι εντολές τις οποίες πρόκειται να εκτελέσει το thread.
Η κλάση twoThreads διαθέτει την μέθοδο main στην οποία δημιουργούνται δυο threads – ένα που που ονομάζεται Jamaica και ένα που ονομάζεται Fiji. Η μέθοδος main αυτής της κλάσης δημιουργεί πρώτα το κάθε thread και κατόπιν αρχιζει την εκτέλεσή του μέσω της μεθόδου start.
Το αποτέλεσμα από την εκτέλεση του προγράμματος θα έχει την μορφή:

```
0 Jamaica
0 Fiji
1 Fiji
1 Jamaica
2 Jamaica
2 Fiji
3 Fiji
```

```
3 Jamaica
4 Jamaica
4 Fiji
5 Jamaica
5 Fiji
6 Fiji
6 Jamaica
7 Jamaica
7 Fiji
8 Fiji
9 Fiji
8 Jamaica
DONE! Fiji
9 Jamaica
DONE! Jamaica
```

Σημειωτέον ότι αυτή η έξοδος δεν είναι δεσμευτική. Τα δύο threads,με τον τρόπο που τα έχουμε ορίσει, δεν συγχρονίζονται σε καμία περίπτωση μεταξύ τους έτσι κάθε έξοδος που θα περιέχει 10 μηνύματα από το καθένα είναι δυνατή. Επειδή ο ορισμός της τάξης Thread γίνεται στο πακέτο java.lang πρέπει κάθε που πρόκειται να προσθέσουμε τη λειτουργικότητά της σε κάποιο πρόγραμμά μας να κάνουμε import το συγκεκριμένο πακέτο

6.2 Η υλοποίηση των Threads

Οι ενέργειες που λαμβάνουν χώρα κατά την εκτέλεση ενός Thread ορίζονται όπως αναφέραμε και παραπάνω στη μέθοδο run του Thread. Μετά από την δημιουργία και την αρχικοποίηση ενός Thread το σύστημα εκτελεί αυτόματα τον κώδικα που υπάρχει στη μέθοδο run. Συνήθως η μέθοδος αυτή περιέχει κάποια επαναληπτική δομή (πχ στην περίπτωση ενός animation θα έχουμε κάποιο loop που θα εμφανίζει διαδοχικά κάποιες εικόνες).
Υπάρχουν δυο τρόποι για τον ορισμό της run μεθόδου κάποιας Thread που δημιουργούμε:

- Δημιουργία υποκλάσης της κλάσης Thread και υπερκάλυψη (overriding) της run μεθόδου που υπάρχει στην Thread κλάση. Παράδειγμα η κλάση simpleThread που δώθηκε στο προηγούμενο παράδειγμα.
- Δημιουργία κλάσης που υλοποιεί το Runnable interface. Σ΄ αυτή την περίπτωση όταν δημιουργούμε την Thread πρέπει να της δώσουμε και μια αναφορά-δείκτη σε στιγμιότυπο της κλάσης που υλοποιεί το Runnable interface. Το παράδειγμα που ακόλουθεί ανήκει σε αυτόν τον τρόπο υλοποίησης Threads.

6.3 Threads σε Applets

Η Applet Clock κλάση που δίνεται στη συνέχεια εμφανίζει την ώρα του συστήματος και την ενημερώνει κάθε δευτερόλεπτο. Η applet χρησιμοποιεί το Runnable interface για να ορίσει την run μέθοδο του Thread που δημιουργεί. Ο λόγος που πρέπει η ενημέρωση και εμφάνιση της ώρας να γίνεται μέσα από κάποιο Thread είναι οτι οι ενημερώσεις είναι συχνές και ο χρήστης πρέπει να μπορεί ταυτόχρονα να κάνει και άλλες εργασίες (πχ να πηγαίνει σε άλλη σελίδα, να κάνει scroll την σελίδα της ώρας κλπ). Επίσης αφού η Java δεν υποστηρίζει πολλαπλή κληρονομικότητα (μια κλάση να είναι υποκλάση σε περισσότερες από μια κλάσεις), η κλάση Clock δεν μπορεί να κληρονομεί ιδιότητες και της κλάσης Thread και της κλάσης Applet. Ετσι η κλάση Clock είναι αναγκασμένη να χρησιμοποιήσει το Runnable interface για να αποκτήσει δυνατότητες δημιουργίας Threads.

```
import java.awt.Graphics;
import java.util.Date;

public class Clock extends java.applet.Applet implements Runnable {
```

```
    Thread clockThread = null;

    public void start() {
        if (clockThread == null) {
            clockThread = new Thread(this, "Clock");
            clockThread.start();
        }
    }



    public void run() {
        // loop terminates when clockThread is set to null in stop()
        while (Thread.currentThread() == clockThread) {
            repaint();
            try {
                clockThread.sleep(1000);
            } catch (InterruptedException e){
            }
        }
    }

    public void paint(Graphics g) {
        Date now = new Date();
        g.drawString(now.getHours() + ":" + now.getMinutes() + ":" + now.getSeconds(), 5, 10);
    }


    public void stop() {
        clockThread = null;
    }
}
```

Σχόλια

- Η start() μέθοδος εξετάζει πρώτα εάν η clockThread μεταβλητή είναι null. Εάν είναι σημαίνει οτι το Clock Applet έχει μολις αρχίσει να εκτελείται ή έχει σταματήσει την εκτέλεσή της. Και στις δυο περιπτώσει χρειάζεται η δημιουργία ενός νέου thread. Αυτό γίνεται με την δήλωση:

  clockThread = new Thread(this, "Clock");

  Η παράμετρος this είναι η αναφορά σε στιγμιότυπο της κλάσης Clock Applet, που υλοποιεί το Runnable interface, η οποία γίνεται ο στόχος της Thread που θα δημιουργηθεί.
- Οταν ο χρήστης φεύγει από την σελίδα της ώρας καλείται η μέθοδος stop() η οποία θέτει στην μεταβλητή clockThread την τιμή null. Αυτό έχει ως αποτέλεσμα να σταματήσει η μέθοδος run() και η Thread που την εκτελεί. Το ίδιο θα μπορούσε να γίνει με την δήλωση clockThread.stop() η οποία όμως δεν συστήνεται για χρήση επειδή σταματά άμεσα την run() σε όποιο σημείο εκτέλεσης και αν βρισκόταν αυτή.
- Η μέθοδος run() αποτελεί τη καρδιά της Clock Applet. Στη μέθοδο αυτή καλείται η μέθοδος repaint() η οποία καλεί αυτόματα την μέθοδο paint() που εμφανίζει στην οθόνη την ώρα.

6.4 Συγχρονισμός των Threads

Η java χρησιμοποιεί τα monitors για να παρέχει συγχρονισμό στα αντικείμενά της. Ένα αντικείμενο που περιέχει synchnonized μεθόδους θεωρείται αντικείμενο τύπου monitor. Ένα αντικείμενο τύπου monitor επιτρέπει σε ένα μόνο thread κάθε στιγμή να εκτελεί μια synchronized μέθοδο αυτού του αντικειμένου. Έτσι όταν καλείται κάποια synchronized μέθοδος ενός αντικειμένου τότε το αντικείμενο "κλειδώνεται" για αυτό το thread και δεν μπορεί να κληθεί καμία άλλη synchronized μέθοδος στο συγκεκριμένο αντικείμενο από κάποιο άλλο thread. Μη-synchronized μέθοδοι μπορούν να κληθούν κανονικά. Μετά το τέλος της εκτέλεσης μιας synchronized μεθόδου το αντικείμενο "ξεκλειδώνεται".

Μια thread που εκτελεί μια synchronized μέθοδο μπορεί να σταματήσει την εκτέλεσή της μόνο στην περίπτωση που από τον κώδικα της μεθόδου κληθεί η wait(). Σε μια τέτοια περίπτωση το thread μπαίνει σε κατάσταση wait για το συγκεκριμένο monitor αντικείμενο ενώ το αντικείμενο "ξεκλειδώνεται".

Όταν ένα thread εκτελέσει από τον κώδικα μιας synchronized μεθόδου, την μέθοδο notify(), τότε "ξυπνά" ένα thread που βρισκόταν σε κατάσταση wait για το monitor αντικείμενο. Το "ξυπνημένο" thread, δεν θα είναι σε θέση να εκτελεσθεί, έως ότου το τρέχον thread τελειώσει την εκτέλεση της synchronized μεθόδου - δηλαδή ξεκλειδώσει το monitor αντικείμενο. Εάν πολλά thread περιμένουν σε αυτό το αντικείμενο, ένα μόνο από αυτά επιλέγεται για να "ξυπνήσει" - η επιλογή γίνεται αυθαίρετα από το σύστημα. Η μέθοδος notifyAll(), "ξυπνά" όλα τα threads που βρίσκονται σε κατάσταση wait για το συγκεκριμένο monitor αντικείμενο.

Οι μέθοδοι wait(), notify(), notifyAll() ορίζονται στην κλάση java.lang.Object.

Ακολουθεί το Producer/Consumer παράδειγμα συγχρονισμού. Ο Producer παράγει έναν ακέραιο αριθμό μεταξύ 0 και 9 (συμπεριλαμβάνων), τον αποθηκεύει σε ένα αντικείμενο CubbyHole, και τυπώνει τον παραγόμενο αριθμό. Για να καταστήσει το πρόβλημα συγχρονισμού πιο ενδιαφέρον, ο Producer "κοιμάται" για ένα τυχαίο χρονικό διάστημα μεταξύ 0 και 100 χιλιοστών του δευτερολέπτου πριν την παραγωγή του επόμενου αριθμού της επανάληψης. Ο Consumer, καταναλώνει όλους τους ακέραιους αριθμούς από το CubbyHole (το ίδιο ακριβώς αντικείμενο στο οποίο ο Producer έβαλε τους ακέραιους αριθμούς αρχικά) τόσο γρήγορα όσο διατίθενται. Ο συγχρονισμός μεταξύ αυτών των δύο threads υλοποιείται σε χαμηλότερο επίπεδο, μέσα στις μεθόδους get() και put() του αντικειμένου CubbyHole.

```
//*******************************
// Class Producer.java
//*******************************
public class Producer extends Thread {
   private CubbyHole cubbyhole;
   private int number;

   public Producer(CubbyHole c, int number) {
      cubbyhole = c;
      this.number = number;
   }

   public void run() {
      for (int i = 0; i < 10; i++) {
         cubbyhole.put(i);
         System.out.println("Producer #" + this.number
                            + " put: " + i);
        try {
            sleep((int)(Math.random() * 1000));
        } catch (InterruptedException e) { }
      }
   }
}
```

```
//*
// Class Consumer.java
//*
public class Consumer extends Thread {
   private CubbyHole cubbyhole;
   private int number;

   public Consumer(CubbyHole c, int number) {
      cubbyhole = c;
      this.number = number;
   }

   public void run() {
      int value = 0;
      for (int i = 0; i < 10; i++) {
         value = cubbyhole.get();
         System.out.println("Consumer #" + this.number
                        + " got: " + value);
      }
   }
}

//*******************************
// Class CubbyHole.java
//*******************************
public class CubbyHole {
   private int contents;
   private boolean available = false;

   public synchronized int get() {
      while (available == false) {
         try {

            wait();
         } catch (InterruptedException e) { }
      }

      available = false;
      System.out.println("in get");
      notifyAll();
      return contents;
   }

   public synchronized void put(int value) {
      while (available == true) {
         try {

            wait();
         } catch (InterruptedException e) { }
      }
      contents = value;
      available = true;
      System.out.println("in gut");
      notifyAll();
   }
}

//*******************************
```

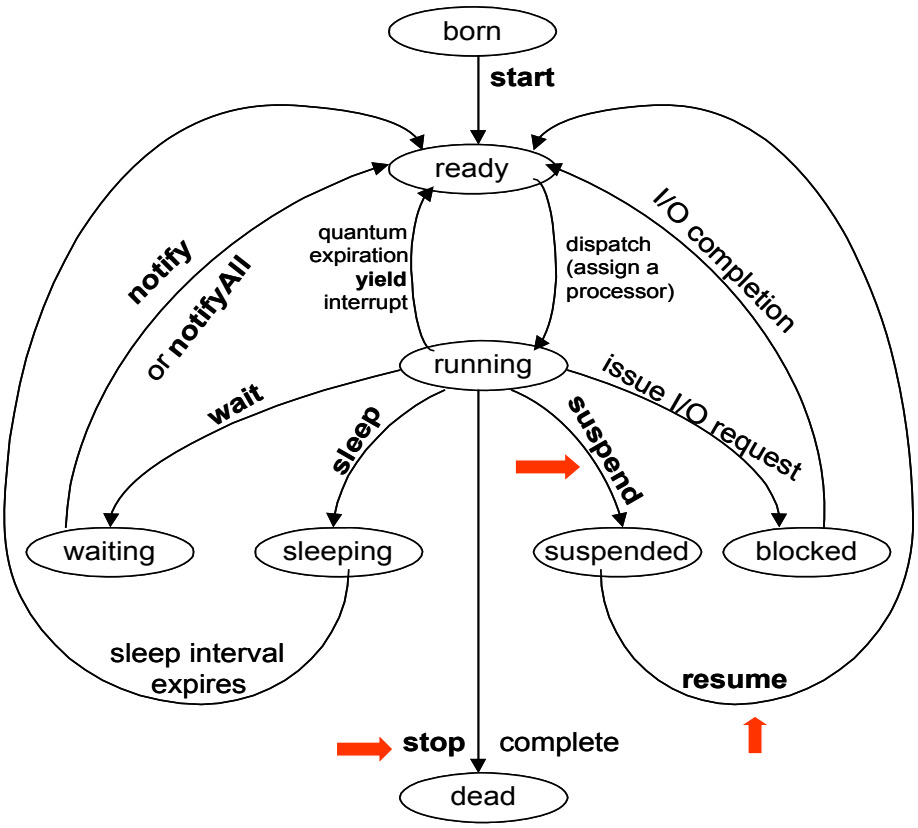
```
// Class ProducerConsumerTest.java
//*******************************
public class ProducerConsumerTest {
   public static void main(String[] args) {
      CubbyHole c = new CubbyHole();
      Producer p1 = new Producer(c, 1);
      Consumer c1 = new Consumer(c, 1);
      p1.start();
      c1.start();
   }
}
```

Το αποτέλεσμα από την εκτέλεση του προγράμματος ProducerConsumerTest θα έχει την μορφή:

```
    . . .
Consumer #1 got: 3
Producer #1 put: 4
Producer #1 put: 5
Consumer #1 got: 4
    . . .
```

6.5 Οι καταστάσεις των Threads

Το παρακάτω παράδειγμα δείχνει τις καταστάσεις στις οποίες μπορεί να βρίσκεται κάποιο thread και τις μεθόδους με τις οποίες μπορεί να πηγαίνει από μιά κατάσταση σε μια άλλη. Οι μέθοδοι stop,suspend και resume είναι μέθοδοι που υπάρχουν αλλά δεν θα πρέπει να χρησιμοποιούνται(deprecated) γιατί είναι δυνατόν να οδηγήσουν σε προβλήματα. Στη θέση αυτών των τριών μεθόδων θα πρέπει να υλοποιούμε δικές μας οι οποίες να αλλάζουν κάποια μεταβλητή σημαία(flag) που να δηλώνει την κατάσταση του thread και χειριζόμαστε κατόπιν αυτές τις καταστάσεις ανάλογα με την περίσταση.



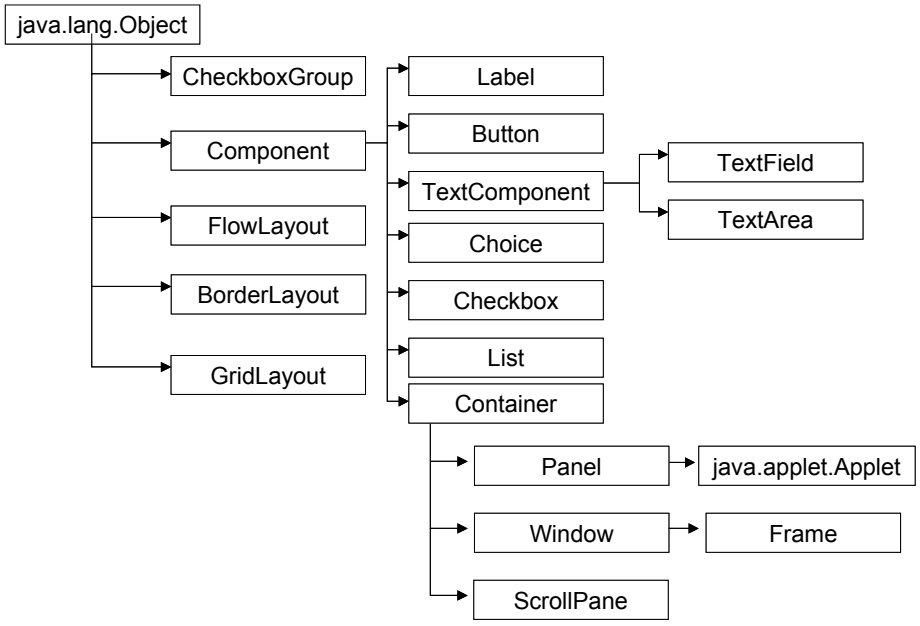
7. Abstract Windowing Toolkit (AWT)

7.1 Βασικά στοιχεία ενός GUI

Ένα Graphical User Interface-GUI (στα ελληνικά Γραφική Διεπαφή με το Χρήστη) είναι το μέρος του προγράμματος, που φροντίζει για τον τρόπο εμφάνισης και χειρισμού του προγράμματος από τον χρήστη. Ένα GUI αποτελείται από GUI-components. Οι κλάσεις που χρησιμοποιούνται για την κατασκευή αντικειμένων τύπου GUI-components ανήκουν στο java.awt (Abstract Windowing Toolkit) package. Οι βασικότερες από αυτές είναι η κλάση Component και η κλάση Container. Κάθε κλάση που κληρονομεί την κλάση Component είναι και αυτή ένα Component. Επίσης κάθε κλάση που κληρονομεί την κλάση Container είναι και αυτή ένα Container. Οι κλάσεις που συνηθέστερα χρησιμοποιούνται για την κατασκευή GUI-components περιγράφονται παρακάτω:

Label	Εμφανίζει κείμενο που δεν μπορεί να τροποποιηθεί από τον χρήστη.
Button	Περιοχή που προξενεί ένα γεγονός (event) όταν επιλέγεται με το ποντίκι.
TextField	Περιοχή όπου ο χρήστης εισάγει δεδομένα από το πληκτρολόγιο. Σε ένα TextField μπορούμε επίσης να εμφανίζουμε πληροφορίες.
TextArea	Περιοχή όπου ο χρήστης εισάγει δεδομένα από το πληκτρολόγιο. Σε ένα TextArea μπορούμε να έχουμε πολλές γραμμές κειμένου σε αντίθεση με το TextField που μπορούμε να έχουμε μόνο μία.
Choice	Μια λίστα στοιχείων από τα οποία ο χρήστης μπορεί να επιλέξει ένα από αυτά με το ποντίκι.
Checkbox	Ένα boolean component που είναι επιλεγμένο ή μή επιλεγμένο. Με αυτό υλοποιούνται και τα Radio buttons (αμοιβαίως αποκλειόμενες επιλογές).
List	Λίστα στοιχείων από τα οποία ο χρήστης μπορεί να επιλέξει ένα από αυτά με το ποντίκι. Διπλό κλικ σε ένα στοιχείο της λίστας προξενεί ένα action event.
Panel	Είναι ένα container αντικείμενο στο οποίο μπορούν να τοποθετηθούν component αντικείμενα.
ScrollPane	Είναι ένα container αντικείμενο στο οποίο μπορεί να τοποθετηθεί ένα component, συνήθως ένα Panel, το οποίο θα εμφανίζεται στον χρήστη αλλά κι όταν αυτό δεν είναι εφικτό θα εμφανίζονται Scrollbars που θα επιτρέψουν την διολίσθηση του component έτσι ώστε να γίνεται ορατό και το τμήμα του που δεν φαίνεται.

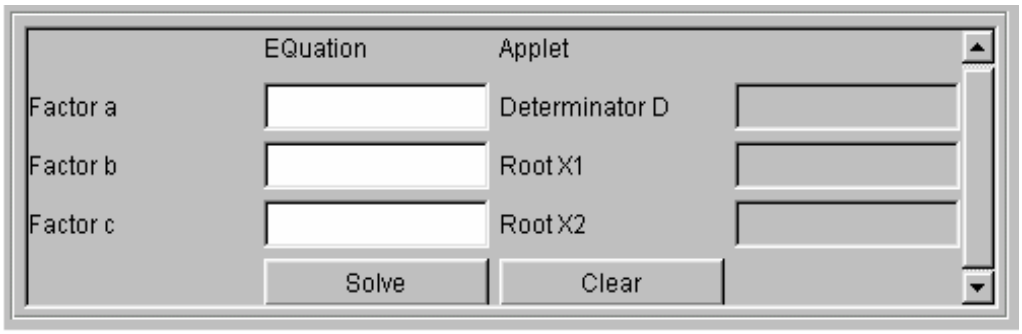
Το επόμενο σχήμα εμφανίζει την ιεραρχία των GUI-components κλάσεων μαζί με κάποιες επιπλέον συμπληρωματικές κλάσεις που διαχειρίζονται ένα GUI.



Στην κλάση Component ορίζονται οι κοινές μέθοδοι όλων των υποκλάσεών της. Αξίζει να σημειωθεί ότι στην Component ορίζονται και οι μέθοδοι paint και repaint που έχουμε ήδη δει την λειτουργία τους στα Applets.

Παράδειγμα κώδικα

Έστω ότι προσπαθούμε να φτιάξουμε ένα applet το οποίο θα λύνει την εξίσωση $\mathbf{ax^2+bx+c=0}$, και θα έχει την παρακάτω περίπου μορφή.



Κατ' αρχήν γράφουμε τον κώδικα της μεθόδου init του applet ο οποίος θα δημιουργεί τα αναγκαία GUI components (TextField, Label, Button, Panel και ScrollPane).

```
// EQuation applet
import java.applet.*;
import java.awt.*;
import java.awt.event.*;

public class eq extends java.applet.Applet {
    // Δηλώσεις των GUI components
    TextField       tfA, tfB, tfC, tfD, tfX1, tfX2;
    Button          btSolve, btClear;
    Panel       p;
    ScrollPane  sp;

    public void init() {
          // Δημιουργία των GUI components.
          // Δεν είναι ακόμη ορατά.
          p = new Panel();
          sp = new ScrollPane(ScrollPane.SCROLLBARS_AS_NEEDED);
          tfA = new TextField();
          tfB = new TextField();
          tfC = new TextField();
          tfD = new TextField();
          tfX1 = new TextField();
          tfX2 = new TextField();
          btSolve = new Button("Solve");
          btClear = new Button("Clear");

          tfD.setEditable(false);
```

```
            tfX1.setEditable(false);
            tfX2.setEditable(false);
        }
    }
```

7.2 Δομή ενός GUI

Σύνθετα GUIs χωρίζονται σε υποπεριοχές (πχ αν θεωρήσουμε το πρόγραμμα Paint των Windows - το GUI του διαιρείται στην περιοχή των μενού επιλογών, στην περιοχή εργασίας, στην περιοχή παλέτας χρωμάτων κλπ) . Για να απλοποιηθεί, λοιπόν, η ανάπτυξη ενός GUI στην Java, χωρίζεται συνήθως σε πολλά Panel components. Αξίζει να αναφερθεί οτι η κλάση Panel κληρονομεί την κλάση Container και η κλάση Applet την κλάση Panel. Έτσι τα Panels και τα Applets είναι Containers και μπορούν να έχουν Components, ακόμα και τύπου Panel. Ο κατασκευαστής της κλάσης Panel δεν παίρνει παραμέτρους. Τελικά προκύπτει μια δενδρική δόμηση του GUI που λέγεται component hierarchy ή containment hierarchy.

Σε κάθε Panel τα Components που περιέχει είναι τοποθετημένα με ένα συγκεκριμένο πλάνο ή σχέδιο (layout). Η Java παρέχει διαχειριστές πλάνων (Layout Managers) για την διευθέτηση των Component αντικειμένων μέσα σε ένα Applet ή Panel. Οι πιο συνηθισμένες κλάσεις των Layout Managers περιγράφονται παρακάτω:

FlowLayout	Είναι ο default Layout Manager για τα Applets και τα Panels. Τοποθετεί τα Component αντικείμενα από αριστερά προς τα δεξιά με την σειρά που προστίθονται
BorderLayout	Τοποθετεί τα Component αντικείμενα σε πέντε περιοχές: Βόρεια, Νότια, Ανατολική, Δυτική και Κεντρική (North, South, East, West, Center).
GridLayout	Τοποθετεί τα Component αντικείμενα σε γραμμές και στήλες με καθορισμένη σειρά (πρώτα γεμίζει η πρώτη γραμμή μετά η δεύτερη γραμμή κλπ).
CardLayout	Τοποθετεί τα Component αντικείμενα σε στοίβα. Κάθε Container αντικείμενο στη στοίβα μπορεί να χρησιμοποιήσει οποιονδήποτε Layout Manager. Μόνο το Container αντικείμενο που βρίσκεται στην κορυφή της στοίβας είναι ορατό.
GridBagLayout	Είναι παρόμοιος με τον GridLayout Manager. Διαφέρει από αυτόν στο ότι κάθε Component αντικείμενο που του προστίθεται μπορεί να έχει οποιοδήποτε μέγεθος (δηλαδή να καλύπτει περισσότερες από μια γραμμές και στήλες). Επίσης η σειρά με την οποία προστίθονται τα Component αντικείμενα στον GridBagLayout Manager μπορεί να είναι οποιαδήποτε.

Παράδειγμα κώδικα

Στην συνέχεια προσθέτουμε κώδικα ο οποίος θα δημιουργήσει το component hierarchy και θα ορίσει τους Layout Managers. Ο παρακάτω κώδικας πρέπει να προστεθεί στο τέλος της μεθόδου init που είδαμε προηγουμένως.

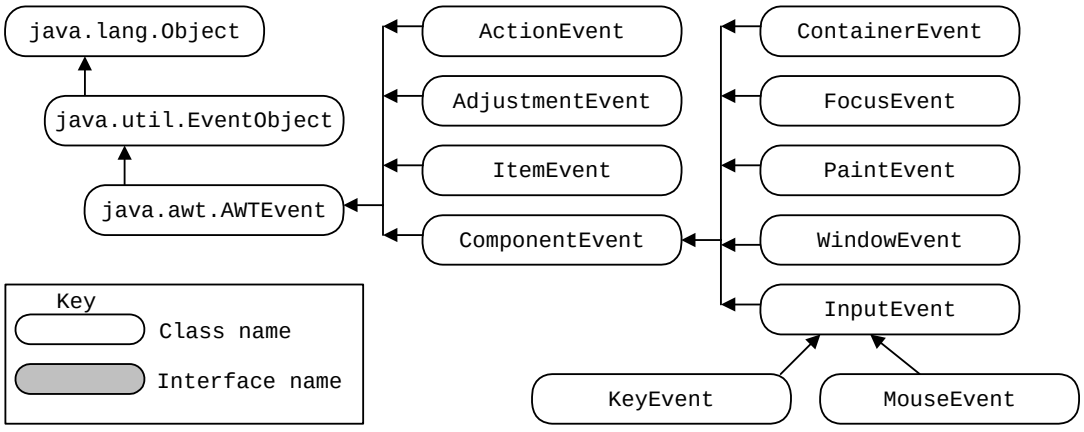
```
// Τοποθέτηση των απλών GUI components μέσα σε ένα Panel.
// Γραμμή 1
p.setLayout(new GridLayout(5,4, 5,5));
p.add(new Label(""));
p.add(new Label("EQuation"));
p.add(new Label("Applet"));
p.add(new Label(""));

// Γραμμή 2
p.add(new Label("Factor a"));
p.add(tfA);
p.add(new Label("Determinator D"));
p.add(tfD);

// Γραμμή 3
p.add(new Label("Factor b"));
```

```
p.add(tfB);
p.add(new Label("Root X1"));
p.add(tfX1);

// Γραμμή 4
p.add(new Label("Factor c"));
p.add(tfC);
p.add(new Label("Root X2"));
p.add(tfX2);

// Γραμμή 5
p.add(new Label(""));
p.add(btSolve);
p.add(btClear);
p.add(new Label(""));

// Δημιουργία του component hierarchy.
// Τοποθέτηση του Panel p μέσα στο ScrollPane sp.
// Έπειτα τοποθέτηση του sp μέσα στο applet.
sp.add(p);
sp.doLayout();
setLayout(new BorderLayout());
add(sp, BorderLayout.CENTER);
```

Ο παραπάνω κώδικας είναι ικανός να δημιουργήσει το οπτικό αποτέλεσμα που είδαμε στην εικόνα του applet (βλέπε σελίδα 29).

7.3 Γεγονότα στα GUI (GUI Events)

Γεγονότα στέλνονται σε ένα πρόγραμμα Java από τα Windows. Η πληροφορία σχετικά με ένα GUI event, αποθηκεύεται σε ένα αντικείμενο που ανήκει στην κλάση ή σε υποκλάσεις της AWTEvent. Η AWTEvent είναι ορισμένη στο πακέτο java.awt.event. Το επόμενο σχήμα εμφανίζει την ιεραρχία των κλάσεων του πακέτου java.awt.event.

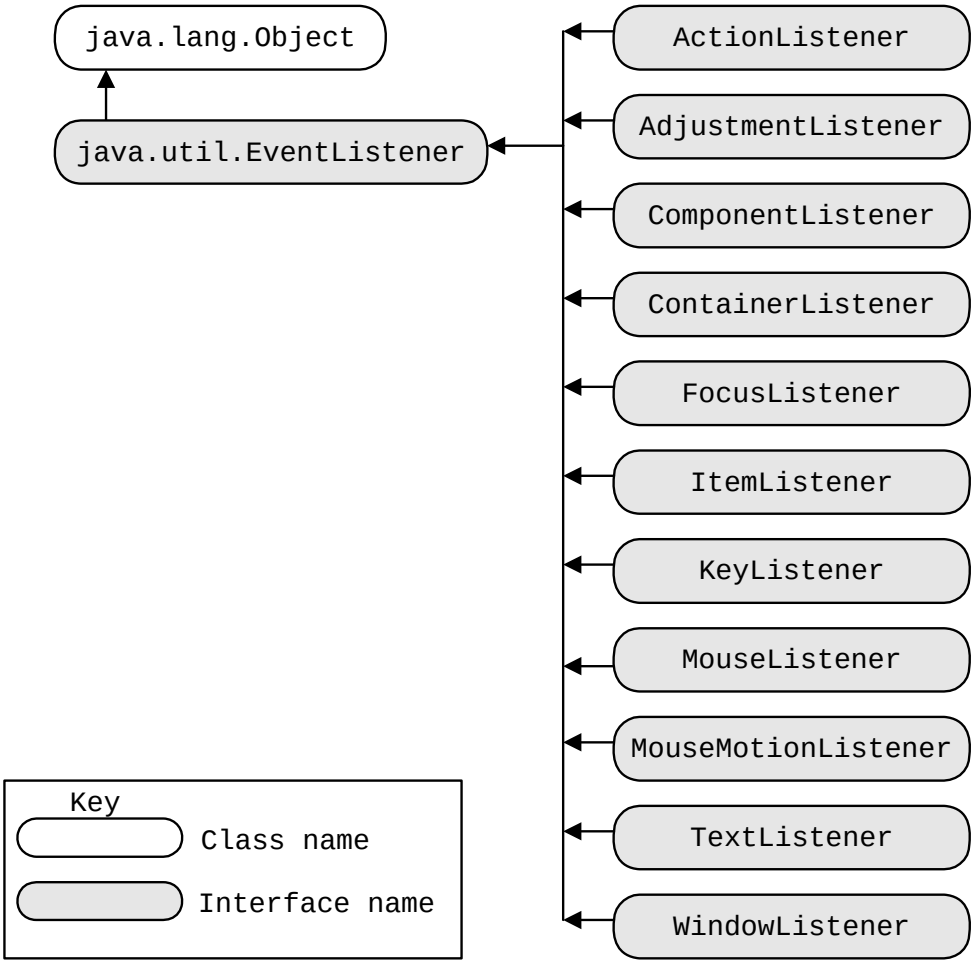


Για να μπορέσει ο προγραμματιστής να επεξεργαστεί κάποιο event το πετυχαίνει με δυο βήματα:

A) να δηλώσει έναν ακροατή γεγονότων (event listener) που θα ακούει τέτοιου τύπου events.

B) να υλοποιήσει τον χειριστή γεγονότων (event handler) που θα πιάνει events που προκύπτουν από το βήμα Α.

- Ένας event listener πρέπει να είναι αντικείμενο κάποιας κλάσης, η οποία υλοποιεί ένα ή περισσότερα event-listener interface του πακέτου java.awt.event. Το επόμενο σχήμα εμφανίζει την ιεραρχία των event-listener interfaces του πακέτου java.awt.event.



- Κάθε event-listener interface παρέχει μια ή περισσότερες event handler μεθόδους. Μια event handler μέθοδος καλείται αυτόματα όταν συμβαίνει κάποιο συγκεκριμένου τύπου γεγονός.
- Στην κλάση του event listener (είναι αυτονόητο ότι) θα πρέπει να υλοποιούνται όλες οι event handler μέθοδοι των event-listener interfaces που η κλάση αυτή υλοποιεί.
- Έτσι το event listener αντικείμενο της κλάσης αυτής, είναι σε θέση να ακούει συγκεκριμένου τύπου events που παράγονται από GUI-Components του προγράμματος - και σε περίπτωση που ένα τέτοιο event παραχθεί, εκτελείται η κατάλληλη event handler μέθοδος για αυτό το event.

Παράδειγμα κώδικα

Ο κώδικας που δίνεται στην συνέχεια ορίζει, κατασκευάζει και δηλώνει δύο ακροατές γεγονότων (event listeners) οι οποίοι ενεργοποιούνται όταν πατήσουμε κάποιο από τα Buttons “Solve” ή “Clear”. Και αυτός ο κώδικας τοποθετείται στο τέλος της μεθόδου init μετά ακριβώς από τον κώδικα για το layout και το component hierarchy.

```
class Solve implements ActionListener() {
                public void actionPerformed(ActionEvent ae) {
                   try {
                      double  a,b,c,d,x1,x2,sqrtd;
                      a=(new Double(tfA.getText())).doubleValue();
                      b=(new Double(tfB.getText())).doubleValue();
                      c=(new Double(tfC.getText())).doubleValue();

                      d = b*b-4*a*c;
                      tfD.setText(Double.toString(d));

                      if (d>0) {
                            sqrtd = Math.sqrt(d);
                            x1 = (-b-sqrtd)/(2d*a);
                            x2 = (-b+sqrtd)/(2d*a);
                            tfX1.setText(Double.toString(x1));
                            tfX2.setText(Double.toString(x2));
                      } else if (d==0) {
                            x1 = x2 = -b/(2d*a);
                            tfX1.setText(Double.toString(x1));
                            tfX2.setText(Double.toString(x2));
                      } else {
                            tfX1.setText("Error");
                            tfX2.setText("Negative D");
                      }
                   } catch (Exception e) {
                      tfD.setText(e.getMessage());
                   }
                }
}
btSolve.addActionListener(new Solve( ) );

class Clear implements ActionListener() {
                public void actionPerformed(ActionEvent ae) {
                      tfA.setText("");
                      tfB.setText("");
                      tfC.setText("");
                      tfD.setText("");
                      tfX1.setText("");
                      tfX2.setText("");
                }
}
```

```
btClear.addActionListener(new Clear( ) );
```

Εναλλακτικά οι χειριστές γεγονότων μπορούν να αρχικοποιούνται “on the fly” στο όρισμα των ακροατών γεγονότων. Σε αυτή την περίπτωση ο χειριστής γεγονότων ονομάζεται inline κλαση.

Έτσι γίνεται εύκολα κατανοητή η αντιστοίχηση των ακροατών γεγονότων στα components και διακρίνουμε ευκολότερα ποιος κώδικας εκτελείται κατά την εμφάνιση ενός event που παράγεται από κάθε component.

Η τυπική δομή καταχώρησης μιας inline κλάσης-χειριστή είναι η ακόλουθη:

```
someComponent.addSomeEventListener(   new SomeEventListener()
{                                          //Αρχικοποιημένη κλάση-ακροατής
        public void someEventMethod( )    //Overrided μέθοδος διαχείρισης
                { Event method implementation here… }
} );
```

Έτσι με χρήση inline χειριστών ο κώδικας που ορίζει, κατασκευάζει και δηλώνει τους ακροατές γεγονότων των Buttons “Solve” ή “Clear” μπορεί να γραφεί ως εξής:

```
btSolve.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent ae) {
         try {
            double  a,b,c,d,x1,x2,sqrtd;
            a=(new Double(tfA.getText())).doubleValue();
            b=(new Double(tfB.getText())).doubleValue();
            c=(new Double(tfC.getText())).doubleValue();

            d = b*b-4*a*c;
            tfD.setText(Double.toString(d));

            if (d>0) {
                  sqrtd = Math.sqrt(d);
                  x1 = (-b-sqrtd)/(2d*a);
                  x2 = (-b+sqrtd)/(2d*a);
                  tfX1.setText(Double.toString(x1));
                  tfX2.setText(Double.toString(x2));
            } else if (d==0) {
                  x1 = x2 = -b/(2d*a);
                  tfX1.setText(Double.toString(x1));
                  tfX2.setText(Double.toString(x2));
            } else {
                  tfX1.setText("Error");
                  tfX2.setText("Negative D");
            }
         } catch (Exception e) {
            tfD.setText(e.getMessage());
         }
      }
});
```

```
        btClear.addActionListener(new ActionListener() {
              public void actionPerformed(ActionEvent ae) {
                    tfA.setText("");
                    tfB.setText("");
                    tfC.setText("");
                    tfD.setText("");
                    tfX1.setText("");
                    tfX2.setText("");
              }
        });
```

7.4 Adapter Classes

Πρόκειται για προϋπάρχουσες κλάσεις που περιέχουν όλες τις συναρτήσεις ενος event listener interface με κενό σώμα. Ο προγραμματιστής, αντί να ορίσει εκ του μηδενός μια κλάση ακροατή και να ορίσει όλες τις μεθόδους του αντίστοιχου interface, απλά παρακάμπτει μόνο τις μεθόδους της adapter κλάσης που τον ενδιαφέρουν. Είναι προφανές ότι αν κάποιο event listener interface περιέχει μία μόνο μέθοδο προς υλοποίηση τότε δεν ορίζεται για αυτό αντίστοιχη adapter κλάση. Παραδείγματα adapter κλάσεων είναι: η MouseAdapter που υλοποιεί το interface MouseListener, η MouseMotionAdapter που υλοποιεί το interface MouseMotionListener, η MouseMotionAdapter που υλοποιεί το interface MouseMotionAdapterListener κλπ.

7.5 Πλήρης κώδικας του applet EQ

Τέλος δίνεται ο πλήρης κώδικας του EQ applet με ορισμένες προσθήκες που του βελτιώνουν την εμφάνιση και του επιτρέπουν να ξεκινά είτε μέσω κάποιας ιστοσελίδας είτε ως stand-alone application λόγω της ύπαρξης της μεθόδου main( ).

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

public class eq extends java.applet.Applet {
    TextField   tfA, tfB, tfC, tfD, tfX1, tfX2;
    Button      btSolve, btClear;
    Panel       p;
    ScrollPane  sp;

    public void init() {
          setBackground(SystemColor.control);

          p = new Panel();
          sp = new ScrollPane(ScrollPane.SCROLLBARS_AS_NEEDED);

          tfA = new TextField();
          tfB = new TextField();
          tfC = new TextField();
```

```
tfD = new TextField();
tfX1 = new TextField();
tfX2 = new TextField();

btSolve = new Button("Solve");
btClear = new Button("Clear");

tfD.setEditable(false);
tfX1.setEditable(false);
tfX2.setEditable(false);

p.setLayout(new GridLayout(5,4, 5,5));
p.add(new Label(""));
p.add(new Label("EQuation"));
p.add(new Label("Applet"));
p.add(new Label(""));

p.add(new Label("Factor a"));
p.add(tfA);
p.add(new Label("Determinator D"));
p.add(tfD);

p.add(new Label("Factor b"));
p.add(tfB);
p.add(new Label("Root X1"));
p.add(tfX1);

p.add(new Label("Factor c"));
p.add(tfC);
p.add(new Label("Root X2"));
p.add(tfX2);

p.add(new Label(""));
p.add(btSolve);
p.add(btClear);
p.add(new Label(""));

sp.add(p);
sp.doLayout();
setLayout(new BorderLayout());
add(sp, BorderLayout.CENTER);

btSolve.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent ae) {
         try {
            double  a,b,c,d,x1,x2,sqrtd;
            a=(new Double(tfA.getText())).doubleValue();
```

```
                        b=(new Double(tfB.getText())).doubleValue();
                        c=(new Double(tfC.getText())).doubleValue();

                        d = b*b-4*a*c;
                        tfD.setText(Double.toString(d));

                        if (d>0) {
                              sqrtd = Math.sqrt(d);
                              x1 = (-b-sqrtd)/(2d*a);
                              x2 = (-b+sqrtd)/(2d*a);
                              tfX1.setText(Double.toString(x1));
                              tfX2.setText(Double.toString(x2));
                        } else if (d==0) {
                              x1 = x2 = -b/(2d*a);
                              tfX1.setText(Double.toString(x1));
                              tfX2.setText(Double.toString(x2));
                        } else {
                              tfX1.setText("Error");
                              tfX2.setText("Negative D");
                        }
                     } catch (Exception e) {
                        tfD.setText(e.getMessage());
                     }
                  }
            });

            btClear.addActionListener(new ActionListener() {
                  public void actionPerformed(ActionEvent ae) {
                        tfA.setText("");
                        tfB.setText("");
                        tfC.setText("");
                        tfD.setText("");
                        tfX1.setText("");
                        tfX2.setText("");
                  }
            });
      } //end of init() method

      public Insets getInsets() {
            Insets      i = super.getInsets();
            return new Insets(i.top+10, i.left+10,
                              i.bottom+10, i.right+10);
      }

      public void paint(Graphics g) {
            Dimension   dim = getSize();
            g.setColor(Color.lightGray);
```

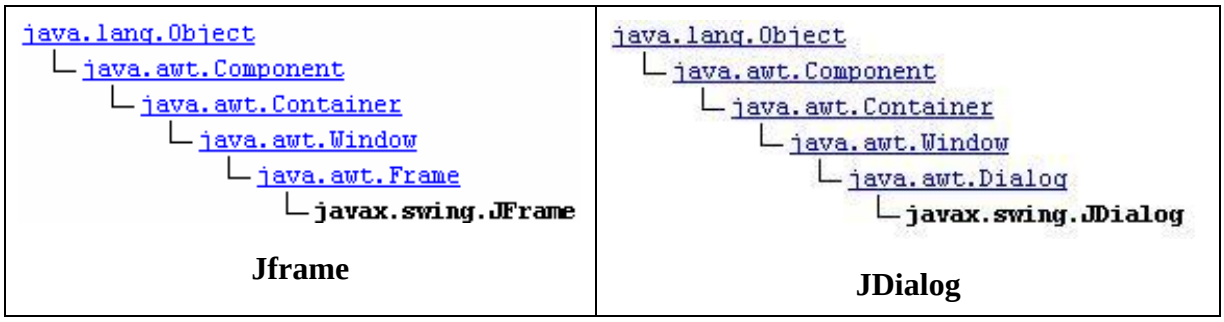
```
            g.draw3DRect(3, 3, dim.width-9, dim.height-9, false);
            g.draw3DRect(4, 4, dim.width-11, dim.height-11, true);
      }

      public static void main(String args[]) {
            Frame fr = new Frame("EQ");
            eq    eq = new eq();
            Button      btClose = new Button("Close");
            Panel p2 = new Panel();

            eq.init();
            eq.start();

            p2.setLayout(new FlowLayout(FlowLayout.CENTER));
            p2.add(btClose);

            fr.setBackground(SystemColor.control);
            fr.setLayout(new BorderLayout());
            fr.add(eq, BorderLayout.CENTER);
            fr.add(p2, BorderLayout.SOUTH);

//          fr.pack();
            fr.setSize(300, 150);

            fr.addWindowListener(new WindowAdapter() {
                  public void windowClosing(WindowEvent we) {
                        Frame src = (Frame)we.getSource();

                        Applet  appl=(Applet)src.getComponent(0);
                        appl.stop();
                        appl.destroy();

                        src.setVisible(false);
                        src.dispose();
                        System.exit(0);
                  }
            });

            btClose.addActionListener(new ActionListener() {
                  public void actionPerformed(ActionEvent ae) {
                        System.exit(0);
                  }
            });

            fr.setVisible(true);
      }
}
```

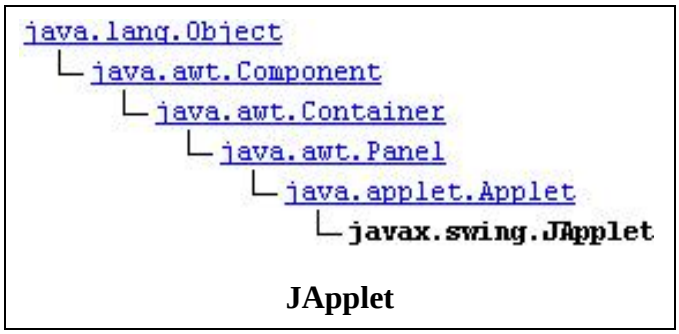
8. Γραφικές διεπαφές τύπου Swing (Swing GUIs)

Τα πακέτα που αρχίζουν με το πρόθεμα javax.swing παρέχουν ευέλικτα και ισχυρά εργαλεία ανάπτυξης GUI. Ιδιαίτερα το πακέτο javax.swing αναπτύχθηκε κυρίως λόγω των ανεπαρκειών του Abstract Windows Toolkit (AWT). Παραδείγματος χάριν, η κλάση JButton που ορίζεται σε αυτό, υπερτερεί της AWT Button κλάσης επιτρέποντας την ύπαρξη όχι μόνο απλού κείμενου, αλλά και εικόνων στα κουμπιά.

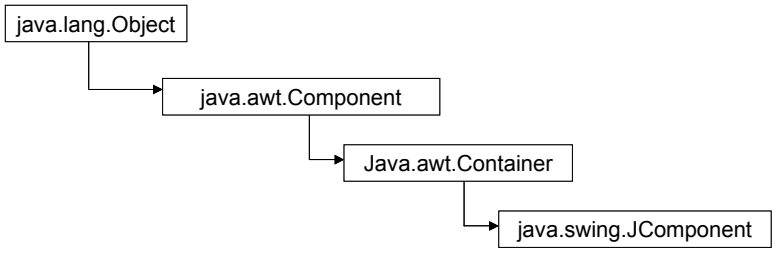
8.1 Top-Level Swing Containers και Swing Components

Κάθε γραφική διεπαφή τύπου Swing πρέπει να έχει τουλάχιστον ένα top-level Swing container. Ένα top-level Swing container παρέχει την απαραίτητη υποστήριξη που χρειάζονται τα Swing components για την εμφάνισή τους και την διαχείριση των γεγονότων που αυτά παράγουν. Υπάρχουν τρία top-level Swing containers: το JFrame, το JDialog, και (για applets) το JApplet. Κάθε JFrame αντικείμενο δημιουργεί ένα κύριο γραφικό παράθυρο, κάθε JDialog αντικείμενο δημιουργεί ένα δευτερεύον παράθυρο (δηλ παράθυρο που εξαρτάται από κάποιο άλλο παράθυρο). Κάθε JApplet αντικείμενο δημιουργεί την περιοχή εμφάνισης ενός applet στο παράθυρο του Web Browser. Η ιεραρχία των top-level Swing containers δίνεται παρακάτω:





Με εξαίρεση τα top-level containers, πχ το `JFrame`, όλα τα Swing components είναι υποκλάσεις της κλάσης `JComponent`. Η ιεραρχία της `JComponent` φαίνεται στο παρακάτω σχήμα:

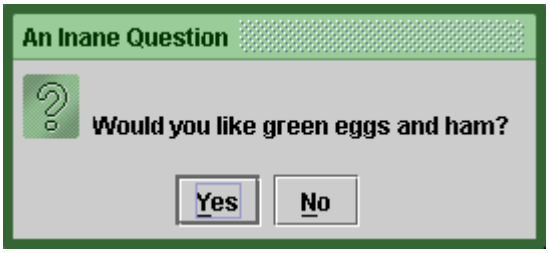


Ακολουθεί ένας πίνακας εμφάνισης των swing Containers και Components

Top-Level Containers

Τα Container στην κορυφή οποιασδήποτε Swing ιεραρχίας.

Applet

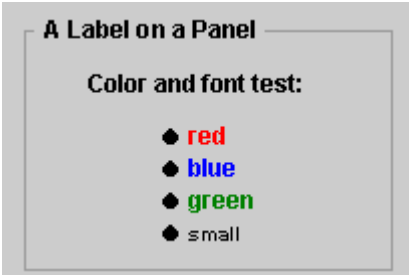


Dialog

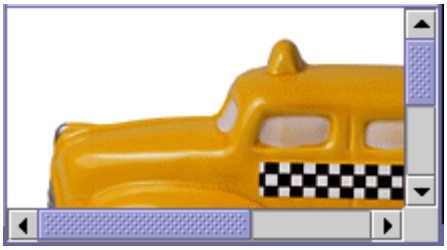
Frame

General-Purpose Containers

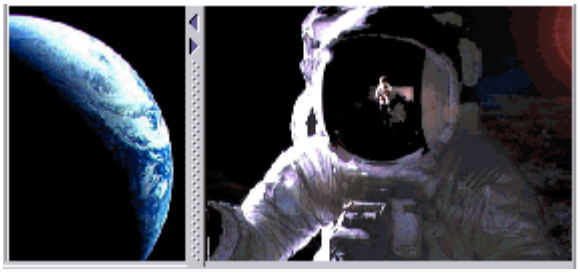
Ενδιάμεσα Container που μπορούν να χρησιμοποιηθούν κάτω από πολλές διαφορετικές περιστάσεις.



Panel



Scroll pane



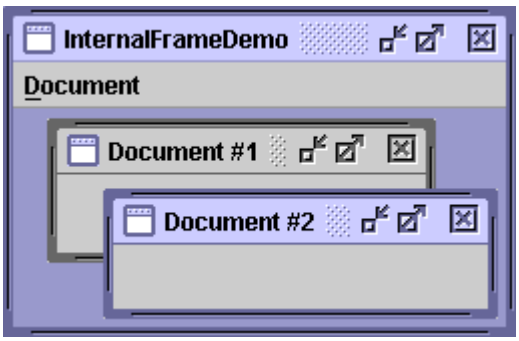
Split pane

Tabbed pane

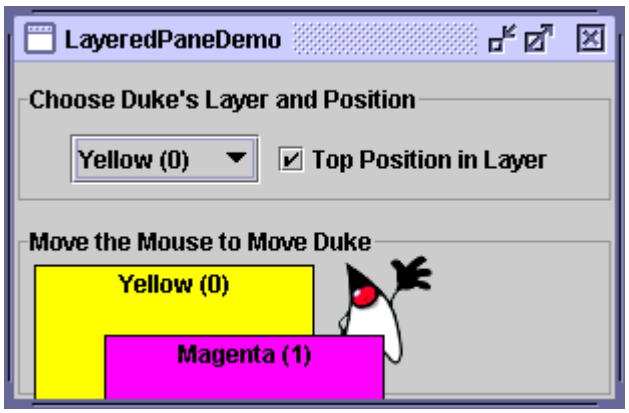
Tool bar

Special-Purpose Containers

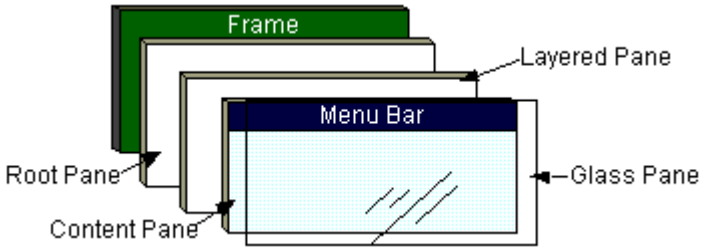
Ενδιάμεσα Container που διαδραματίζουν συγκεκριμένους ρόλους σε ένα GUI.



Internal frame



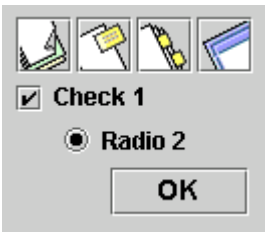
Layered pane



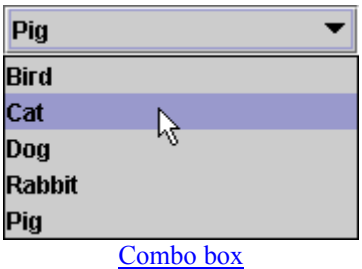
Root pane

Basic Controls

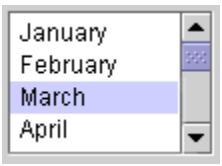
Ατομικά Component που υπάρχουν πρωτίστως για να πάρουν δεδομένα από το χρήστη, επίσης εν γένει παρουσιάζουν κάποια κατάσταση.



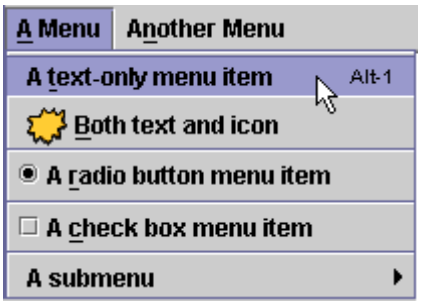
Buttons



Combo box



List



Menu

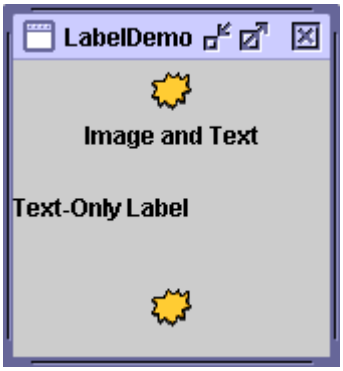
Slider

Spinner

Text field or Formatted text field

Uneditable Information Displays

Ατομικά Component που υπάρχουν απλώς για να δώσουν πληροφορίες στον χρήστη.



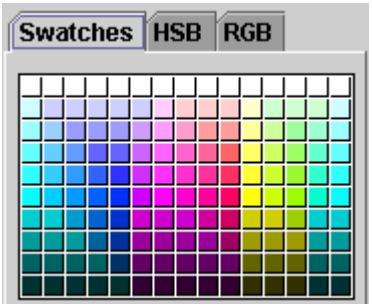
Label

Progress bar

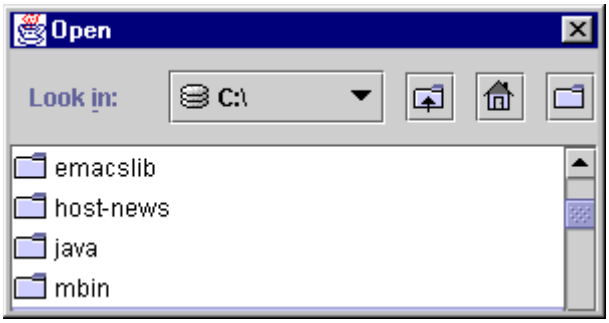
Tool tip

Interactive Displays of Highly Formatted Information

Ατομικά Component που εμφανίζουν πληροφορίες με συγκεκριμένη μορφοποίηση η οποία (εάν επιλεχθούν) μπορεί να τροποποιηθεί από το χρήστη.



Color chooser

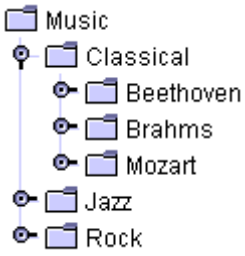


File chooser

First Name	Last Name	Favorite Food
Jeff	Dinkins	
Ewan	Dinkins	
Amy	Fowler	
Hania	Gajewska	
David	Geary	

Table

Text



Tree

Παράδειγμα Swing Εφαρμογής

Στη συνέχεια θα παρουσιαστούν οι λειτουργίες των Swing κλάσεων μέσω ενός παραδείγματος:

- JFrame
- JPanel
- JButton

Οι χρήση των υπολοίπων Swing Components είναι παρόμοια. Συνήθως, όταν υλοποιείται μια GUI εφαρμογή σε Swing, δημιουργούμε ένα JFrame αντικείμενο και επιλέγουμε συγκεκριμένο πλάνο ή σχέδιο (layout) για αυτό. Κατόπιν τοποθετούμε ένα ή περισσότερα JPanels στο Jframe. Τα JPanels διαθέτουν επίσης επιλογές layout όπως και τα JFrames. Στη συνέχεια, μπορούμε να προσθέτουμε και άλλα αντικείμενα τύπου Component. Ο κώδικας που ακολουθεί είναι ένα παράδειγμα των όσων αναφέρθηκαν.

```
import java.lang.*;
import java.util.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

class SimpleGui1 {
   public static void main(String args[]) {
      // Let's make a button first.
      JButton btn = new JButton("Click Me!");
      btn.setMnemonic(KeyEvent.VK_C); // Now you can hit the button with
                                      // Alt-C.

      // Let's make the panel with a flow layout.
      // Flow layout allows the components to be
      // their preferred size.
      JPanel pane = new JPanel(new FlowLayout());
      pane.add(btn);          // Add the button to the pane.

      // Now for the frame
      JFrame fr = new JFrame();
      fr.setContentPane(pane); // Use our pane as the default pane
      fr.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // Exit program
                                             //when frame is closed.

      fr.setLocation(200, 200); // locate Frame at (200, 200).
      fr.pack();                // Frame is ready. Pack it up for
                               // display.
      fr.setVisible(true);      // Make it visible.
   }
}
```

Το αποτέλεσμα της παραπάνω εφαρμογής είναι το ακόλουθο:

Τα JButton, JPanel, και JFrame αντικείμενα διαθέτουν πολύ περισσότερες λειτουργίες από αυτές που παρουσιάστηκαν και μπορούν να χρησιμοποιηθούν (βλέπε στο JAVA API για περισσότερα). Στο απλό παράδειγμα που παρουσιάστηκε, κάνοντας κλικ στο κουμπί δεν φέρνει κανένα αποτέλεσμα, αλλά κάνοντας κλικ στο "X" του παραθύρου τερματίζει το πρόγραμμα.

8.2 Διαχειριστές Γεγονότων - Event Handlers

Για να υπάρχει κάποιο αποτέλεσμα κάνοντας κλικ στο κουμπί του παραπάνω παραδείγματος χρειάζεται να προσθέσουμε σε αυτό το component (JButton αντικείμενο) τον κατάλληλο ακροατή γεγονότων (ButtonListener). Έτσι όταν ο χρήστης πατήσει το κουμπί, ο ακροατής προκαλείται και καλούνται οι κατάλληλες μέθοδοί του (διαχειριστές γεγονότων). Η εργασία αυτή γίνεται με τον ίδιο τρόπο που ορίζονται οι ακροατές γεγονότων στα AWT Component. Ο παραπάνω κώδικας με την προσθήκη ακροατή γεγονότων για το JButton αντικείμενο τροποποιείτε ως εξής:

```
import java.lang.*;
import java.util.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

class SimpleGui1 {
   public static void main(String args[]) {
      // Let's make a button first.
      JButton btn = new JButton("Click Me!");
      btn.setMnemonic(KeyEvent.VK_C); // Now you can hit the button with
                                      // Alt-C.

      btn.addActionListener(new ButtonListener()); // Allow the button
                                                   // to disable itself



      // Let's make the panel with a flow layout.
      // Flow layout allows the components to be
      // their preferred size.
      JPanel pane = new JPanel(new FlowLayout());
      pane.add(btn);          // Add the button to the pane.

      // Now for the frame
      JFrame fr = new JFrame();
      fr.setContentPane(pane); // Use our pane as the default pane
      fr.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // Exit program
                                             //when frame is closed.

      fr.setLocation(200, 200); // locate Frame at (200, 200).
      fr.pack();                // Frame is ready. Pack it up for
                                // display.
      fr.setVisible(true);      // Make it visible.
   }
```

```
    // Button event handler
    static class ButtonListener implements ActionListener {
       public void actionPerformed(ActionEvent e) {
          btn.setEnabled(false); // Disable the button
       }
    }

}
```

Το αποτέλεσμα του προγράμματος είναι το ίδιο. Ωστόσο, εάν γίνει κλικ στο κουμπί, τότε αυτό τίθεται εκτός λειτουργίας:

Τα Swing components παράγουν διάφορους τύπους γεγονότων (events). Ο παρακάτω πίνακας περιέχει τα πιο συνηθισμένα από αυτά.

Some Events and Their Associated Event Listeners	
Act that Results in the Event	**Listener Type**
User clicks a button, presses Enter while typing in a text field, or chooses a menu item	`ActionListener`
User closes a frame (main window)	`WindowListener`
User presses a mouse button while the cursor is over a component	`MouseListener`
User moves the mouse over a component	`MouseMotionListener`
Component becomes visible	`ComponentListener`
Component gets the keyboard focus	`FocusListener`
Table or list selection changes	`ListSelectionListener`
Any property in a component changes such as the text on a label	`PropertyChangeListener`

8.3 Swing και Threads

Η μετάβαση από τα AWT στα Swing GUI παρουσιάζει κάποιο μικρό πρόβλημα σε σχέση με τα Threads. Όλες οι AWT κλάσεις είναι συμβατές με τα threads (thread safe) ενώ οι swing κλάσεις δεν είναι.

Στο κεφάλαιο των Threads αναφέρθηκε πως μέσω των synchnonized μεθόδων η Java επιτυγχάνει τον συγχρονισμό των threads που έχουν πρόσβαση σε κάποιο κοινό αντικείμενο. Οι AWT μέθοδοι συγχρονίζονται την στιγμή της αρχικοποίησης του αντίστοιχου αντικειμένου τους στην οθόνη. Έτσι εμποδίζονται πιθανά αδιέξοδα (deadlocks) που επιφέρουν το «κρέμασμα» εφαρμογών την στιγμή της εμφάνισης γραφικών αντικείμενων (πχ. buttons, panels, κλπ) στην οθόνη.

Ωστόσο όπως αναφέρθηκε και προηγουμένως οι swing κλάσεις δεν είναι thread-safe. Ο κώδικας διαχείρισης γεγονότων στις Swing εφαρμογές εκτελείται μέσα από μια συγκεκριμένη thread η

οποία ονομάζεται *event-dispatching thread*. Με αυτό τον τρόπο επιτυγχάνεται ο κώδικας ενός event handler να τελειώνει πριν την εκτέλεση του κώδικα ενός άλλου event handler. Για την αποφυγή πιθανού αδιεξόδου (deadlock), είναι απαραίτητο και τα Swing components να δημιουργούνται, τροποποιούνται και ερωτούνται για την κατάστασή τους μέσα από την *event-dispatching thread*.

8.3.1 Η χρήση της μεθόδου invokeLater

Η κλήση της μεθόδου invokeLater μέσα από κάποιο thread ζητά από την event-dispatching thread να εκτελέσει κάποιο κώδικα. Ο συγκεκριμένος κώδικας πρέπει να γραφτεί μέσα στην μέθοδο run ενός αντικειμένου τύπου Runnable και το Runable αντικείμενο να ορισθεί ως παράμετρος της μεθόδου invokeLater. Η invokeLater μέθοδος επιστρέφει αμέσως χωρίς να περιμένει την εκτέλεση του συγκεκριμένου κώδικα από την event-dispatching thread. Ακολουθεί ένα παράδειγμα χρήσης της μεθόδου invokeLater:

```
Runnable updateAComponent = new Runnable() {
   public void run() { component.doSomething(); }
};
SwingUtilities.invokeLater(updateAComponent);
```

Στη συνέχεια το αρχικό παράδειγμα του κεφαλαίου τροποποιήθηκε ώστε να εκτελεί τον κώδικα δημιουργίας του GUI στην event-dispatching thread μέσω της μέθόδου invokeLater. Κρίνουμε σκόπιμο να αναφέρουμε ότι για να αποφύγουμε κρεμάσματα σε Swing εφαρμογές είναι σκόπιμο να εκτελείται ο κώδικας εμφάνισης των GUI στην event-dispatching thread μέσω της μεθόδου invokeLater. Παρόμοια με την invokeLater είναι και η μέθοδος invokeAndWait που μπορείται να αναφερθείτε σε αυτή

```
import java.lang.*;
import java.util.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

class SimpleGui1 {

   private static JButton btn = null;

   public static void createAndShowGUI() {
      // Let's make a button first.
      btn = new JButton("Click Me!");
      btn.setMnemonic(KeyEvent.VK_C); // Now you can hit the button with
                                      // Alt-C.

      btn.addActionListener(new ButtonListener()); // Allow the button
                                                   // to disable itself



      // Let's make the panel with a flow layout.
      // Flow layout allows the components to be
      // their preferred size.
      JPanel pane = new JPanel(new FlowLayout());
      pane.add(btn);          // Add the button to the pane.
```

```
      // Now for the frame
      JFrame fr = new JFrame();
      fr.setContentPane(pane); // Use our pane as the default pane
      fr.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // Exit program
                                                 //when frame is closed.

      fr.setLocation(200, 200); // locate Frame at (200, 200).
      fr.pack();                // Frame is ready. Pack it up for
                               // display.
      fr.setVisible(true);      // Make it visible.
   }


    public static void main(String[] args) {

        //Schedule a job for the event-dispatching thread:
       //creating and showing this application's GUI.
       javax.swing.SwingUtilities.invokeLater(new Runnable() {
           public void run() {
               createAndShowGUI();
           }
       });
    }


   // Button event handler
   static class ButtonListener implements ActionListener {
      public void actionPerformed(ActionEvent e) {
         btn.setEnabled(false); // Disable the button
      }
   }

}
```

9. Animation σε Java

9.1 Γενικά περί Animation

Ένας από τους καλύτερους τρόπους για να γίνει ένα πρόγραμμα σε Java ή μια σελίδα που τρέχει Java applet πιο ελκυστική, είναι να εκτελεί animation.

Η βασική ιδέα για να εκτελέσει ένα πρόγραμμα animation είναι να μπορεί να εμφανίζει διαφορετικά στιγμιότυπα κάποιας κίνησης το ένα μετά το άλλο με σχετικά μεγάλη ταχύτητα. Παρακάτω θα περιγράψουμε μεθοδολογίες για βέλτιστη εκτέλεση animation. Ας ξεκινήσουμε όμως παρουσιάζοντας τα βασικά είδη animation:

1. *Στατική multi-frame animation*. Πρόκειται για την περίπτωση κατά την οποία εμφανίζουμε διαδοχικά γραφικά ή εικόνες (frames) τη μία πάνω στην άλλη με σχετικά μεγάλη ταχύτητα (η animation που παράγεται από υπολογιστή μπορεί να επιτύχει 10-20 frames το δευτερόλεπτο). Η τεχνική αυτή δε διαφέρει σε τίποτα από την παραδοσιακή τεχνική πού χρησιμοποιείται στον κινηματογράφο και στα κινούμενα σχέδια (cartoons).
2. *Κινούμενες εικόνες*. Πρόκειται για την περίπτωση κατά την οποία έχουμε μια μικρή εικόνα (συνήθως ονομάζεται sprite) την οποία μετακινούμε πάνω από μιά άλλη μεγαλύτερη εικόνα (background). Αυτή η τεχνική χρησιμοποιείται κατά κόρον στα παιχνίδια. Το αποτέλεσμα μπορεί να γίνει πολύ πιο εντυπωσιακό αν το κινούμενο sprite αποτελείται από πολλά frames, αλλάζει δηλαδή μορφή κατά την κίνησή του πάνω από το background. Επίσης, κατά προτίμηση, η κινούμενη εικόνα (sprite) θα πρέπει να έχει διαφανές (transparent) background έτσι ώστε να μην είναι απαραίτητο να έχει παραλληλόγραμμη μορφή. Αυτό μπορεί να επιτευχθεί με το format GIF 89 για εικόνες της Compuserve.

9. 2 Το animation loop

Όποιο είδος animation και να θέλετε να πραγματοποιήσετε στο πρόγραμμά σας, θα χρειαστείτε έναν βρόχο (animatin loop), ο οποίος θα αναλάβει να ζωγραφίζει σε τακτά χρονικά διαστήματα τα διαδοχικά στιγμιότυπα της κίνησης που θα υλοποιεί η animation. Στην Java θα πρέπει αυτός ο βρόχος να εκτελείται σε ένα δικό του ξεχωριστό thread. Δεν θα έπρεπε για παράδειγμα να μπει σε μεθόδους όπως η paint() ή η update() της κλάσης java.awt.Component με τη δικαιολογία ότι περιέχει λειτουργίες γραφικών, γιατί αυτές οι μέθοδοι ανήκουν στο βασικό AWT thread το οποίο είναι υπεύθυνο για να το χειρισμό των βασικών λειτουργιών ζωγραφικής και χειρισμού γεγονότων (event handling) των αντικειμένων του προγράμματος. Το animation loop θα πρέπει να μπαίνει στη μέθοδο run() ενός thread που δημιουργούμε ειδικά για την εκτέλεση της animation. Στη μέθοδο paint() θα μπαίνουν οι εντολές οι οποίες ζωγραφίζουν μόνο ένα frame κάθε φορά (το τρέχον).

Παρακάτω δίνουμε τη γενική μορφή που πρέπει να έχει το animation loop. Παρατηρήστε ότι χρησιμοποιούμε μια μεταβλητή frameNumber, η οποία κρατάει τον αριθμό του frame που πρέπει να εμφανιστεί στην εκάστοτε επανάληψη του βρόχου (τρέχον frame) και μια μεταβλητή delay, στην οποία γράφουμε πόσα msec (χιλιοστά του δευτερολέπτου) θα παρεμβάλλονται ανάμεσα στην εμφάνιση των διαδοχικών frames. Προσέξτε ότι για να ζωγραφίσουμε το κάθε frame μέσα από το animation loop, καλούμε τη μέθοδο repaint() η οποία ανήκει στην κλάση java.awt.Component. Η λειτουργία της είναι να καλεί την μέθοδο paint() και επομένως (αφού στην paint() βάζουμε τις εντολές για εμφάνιση του τρέχοντος frame) να ζωγραφίζει το τρέχον frame.

```
public void run() {
        while (true) {
            //Αυξάνουμε το animation frame κατά ένα.
            frameNumber++;
```

```
            //Το ζωγραφίζουμε.
            repaint();

            //Περιμένουμε delay msec.
            try {Thread.sleep(delay); } catch (InterruptedException e) {}

    }
```

9.3. Βασικό πρόγραμμα animation

Θα παρουσιάσουμε τώρα ένα παράδειγμα μιας Applet που εκτελεί *στατική multi-frame animation*, χρησιμοποιώντας ένα animation loop όπως μας υπέδειξε η προηγούμενη παράγραφος. Το παράδειγμα αυτό είναι στοιχειώδες, με την έννοια ότι είναι το μικρότερο πρόγραμμα σε Java που μπορεί να εκτελέσει animation. Ελπίζουμε ότι τα σχόλια είναι επαρκή.

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

// Η applet υλοποιεί το Runnable interface προκειμένου να μπορέσει να
// χρησιμοποίησει δικά της threads.
public class AnimatorApplet extends Applet implements Runnable {
   int frameNumber = -1;          //Μετρητής των frames

   int delay;                     //Καθυστέρηση σε msec

  //Ορίζουμε το ξεχωριστό Thread για animation
   Thread animatorThread;

  public void init() {
      //Θα παρεμβάλλονται 100msec ανάμεσα στα frames
      delay=100;
  }

   public void start() {
         // Αρχικοποίησε το Thread για animation
         if (animatorThread == null) {
            animatorThread = new Thread(this);
         }
         // Ξεκίνα το Thread για animation
         animatorThread.start();
      }
   }

   public void stop() {
      //Σταμάτα το Thread για animation
      animatorThread = null;
   }

  //Σε αυτή τη μέθοδο μπαίνει το animation loop.
  public void run() {

      //animation loop
```

```
            while (Thread.currentThread() == animatorThread) {

                //Αύξησε τον αριθμό του frame κατά ένα
                frameNumber++;

                //Ζωγράφισε το τρέχον frame
                repaint();

                //Περίμενε μερικά msec
                try {Thread.sleep(delay) }
                catch (InterruptedException e) { break; }
            }
        }


        //Στην paint() μπαίνουν οι εντολές για εμφάνιση του τρέχοντος frame.
        public void paint(Graphics g) {
            // Ζωγράφισε ένα string που δείχνει
            // τον αριθμό του τρέχοντος frame
            g.drawString("Frame " + frameNumber, 0, 30);
            // Πράγματι, πρόκειται για πολύ χαζό frame
        }

    }
```

9.4. Τεχνικές Βελτιστοποίησης

Το παραπάνω παράδειγμα εκτελεί μια στοιχειώδη και φυσικά πολύ βαρετή animation: Εμφανίζει στην οθόνη τη συμβολοσειρά “Frame i”, όπου i ο αριθμός του τρέχοντος frame. Αν δοκιμάσετε να το τρέξετε εκτός από το ότι θα πλήξετε, μπορεί να διαπιστώσετε ότι έχει ένα μικρό πρόβλημα. Παρόλο που τυπώνεται μόλις ένα string κάθε φορά (η εκτύπωση ενός string είναι πολύ εύκολη υπολογιστικά υπόθεση - φανταστείτε να τυπώναμε εικόνες), μερικές φορές τα γράμματα μοιάζουν να αναβοσβήνουν.

Το φαινόμενο αυτό λέγεται flashing ή flickering και είναι το κυριότερο πρόβλημα που πρέπει να επιλύσουμε προκειμένου να επιτύχουμε ομαλή (smooth, flicker free) animation.

Ο λόγος που εμφανίζεται το screen flickering είναι ο εξής: κάθε φορά που καλούμε τη μέθοδο repaint() στο animation loop, προκειμένου να εμφανίσουμε ένα καινούργιο frame, σύμφωνα με την υπάρχουσα υλοποίηση της μεθόδου repaint(), καθαρίζεται το background και μετά καλείται η μέθοδος paint() για να ζωγραφίσει. Επιπλέον, ο χρόνος που απαιτείται για τον καθαρισμό του background και την εμφάνιση του νέου frame στην οθόνη είναι συνήθως μεγαλύτερος από τη συχνότητα σάρωσης (refresh rate) της οθόνης. Το αποτέλεσμα είναι ότι κάθε φορά που παρεμβάλλεται μια σάρωση ανάμεσα σε αυτές τις δύο λειτουργίες (καθαρισμός background και εμφάνιση νέου frame) βλέπουμε για λίγο χρόνο στην οθόνη μισοτελειωμένα γραφικά (μισοζωγραφισμένο το νέο frame ή, στη χειρότερη περίπτωση, εντελώς καθαρό background).

Παρακάτω θα παρουσιάσουμε δύο πολύ χρήσιμες τεχνικές που πρέπει να χρησιμοποιούνται από κάθε σοβαρό πρόγραμμα animation για την εξάλειψη του φαινομένου του flickering και για επίτευξη ομαλής (smooth) animation.

- Επανακαθορισμός της υπάρχουσας μεθόδου update().

Το Abstract Windowing Toolkit (AWT) package της Java κάθε φορά που ένα αντικείμενό του (Applet, Canvas, Frame) ζητά να ξαναζωγραφίσει τον εαυτό του καλεί τη μέθοδο update()

αυτού του αντικειμένου. Στη συνέχεια, η μέθοδος update() καλεί μια μέθοδο clearRect() για να καθαρίσει το background του αντικειμένου που την καλεί πριν καλέσει τη μέθοδο paint() για να ξαναζωγραφίσει το αντικείμενο. Η βασική ιδέα σε αυτή την τεχνική είναι ότι πρέπει να επανακαθορίσουμε τη μέθοδο update() ώστε να είναι πιο έξυπνη: να καθαρίζει το background μόνο όταν πραγματικά το χρειαζόμαστε. Έτσι μπορούμε να επιταχύνουμε αισθητά την εκτέλεση του animation loop και να ελαχιστοποιήσουμε το flickering. Για παράδειγμα στο πρόγραμμα που είδαμε παραπάνω είναι περιττό να σβήνουμε το background κάθε φορά που ζωγραφίζουμε ένα νέο frame. Μπορούμε επομένως να αντιγράψουμε στη μέθοδο update() αυτούσιο τον κώδικα που έχουμε στη μέθοδο paint() και μόνο αυτό (καμία εντολή για καθαρισμό background). Έτσι, κάθε φορά που θα καλείται η update για να εμφανιστεί ένα νέο frame, δεν θα σβήνεται το παλιό - απλά θα εμφανίζεται πάνω από το παλιό το καινούργιο frame.

- Χρησιμοποίηση buffer οθόνης

Σύμφωνα με αυτή τη μέθοδο μπορούμε να εξοικονομήσουμε πολύ χρόνο κατά την εμφάνιση γραφικών στην περίπτωση που η εμφάνισή τους περιλαμβάνει πολύπλοκους υπολογισμούς. Η ιδέα είναι ότι χρησιμοποιούμε κατά κάποιον τρόπο μια ψεύτικη οθόνη (offscreen) στην οποία ζωγραφίζουμε το frame που θέλουμε να εμφανίσουμε χωρίς να το βλέπει ο χρήστης στην πραγματική οθόνη. Την ώρα που κάνουμε τους υπολογισμούς για την εμφάνιση του frame στην ψεύτικη οθόνη, ο χρήστης βλέπει ακόμα το προηγούμενο frame και όχι ένα νέο frame να εμφανίζεται σιγά σιγά πάνω από το προηγούμενο. Μόλις τελειώσουμε τη σχεδίαση του frame στην ψεύτικη οθόνη, εμφανίζουμε την ψεύτικη οθόνη στην πραγματική. Έτσι, ο χρήστης βλέπει το νέο frame να εμφανίζεται πολύ γρήγορα στην οθόνη του. Στην Java, ο buffer οθόνης (offImage) που χρησιμοποιούμε είναι ένα πεδίο Image που δημιουργούμε χρησιμοποιώντας μια από τις μεθόδους createImage() της κλάσης java.awt.Component. Για να ζωγραφίσουμε στην οθόνη αυτή, απλά χρησιμοποιούμε το Graphics αντικείμενό της (offGraphics) το οποίο μπορούμε να προσπελάσουμε με τη μέθοδο Image getGraphics(). Όπως θα δείτε στο παράδειγμα που παρουσιάζουμε πιο κάτω, όλες οι εντολές γραφικών μέσα στη μέθοδο update() ζωγραφίζουν σε αυτό το αντικείμενο δηλαδή στην ψεύτικη οθόνη αντί στην αληθινή. Η εμφάνιση του νέου frame στην πραγματική οθόνη γίνεται μονομιάς με την κλήση της μεθόδου drawImage() στο τέλος της update().

Το παράδειγμα παρουσιάζει μια Applet που χρησιμοποιεί και τις δύο τεχνικές που αναφέραμε παραπάνω για να επιτύχει smooth animation. Και πάλι ελπίζουμε ότι τα σχόλια είναι επαρκή:

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

public class ImageSequence extends Applet implements Runnable {
   int frameNumber = -1;
   int delay;
   Thread animatorThread;
   boolean frozen = false;

   Dimension offDimension;
   Image offImage;
   Graphics offGraphics;

   Image[] images;

   public void init() {
      String str;
      int fps = 10;

      //Πόσα milliseconds ανάμεσα στα frames;
      str = getParameter("fps");
      try {
```

```
        if (str != null) {
            fps = Integer.parseInt(str);
        }
    } catch (Exception e) {}
    delay = (fps > 0) ? (1000 / fps) : 100;

    //Φόρτωσε όλες τις εικόνες.
    images = new Image[10];
    for (int i = 1; i <= 10; i++) {
        // Τα ονόματα των αρχείων είναι ενδεικτικά
        images[i-1] = getImage(getCodeBase(),
                    "myimage"+i+".gif");
    }
}

public void start() {
    if (frozen) {
      //Μην κάνεις τίποτα. Ο χρήστης ζήτησε διακοπή
    } else {
        //Ξεκίνα την animation!
        if (animatorThread == null) {
            animatorThread = new Thread(this);
        }
        animatorThread.start();
    }
}

public void stop() {
    //Σταμάτα την animation
    animatorThread = null;

    //Απελευθέρωσε τα αντικείμενα που χρειάζονταν
    // για την ψεύτικη οθόνη
    offGraphics = null;
    offImage = null;
}

public boolean mouseDown(Event e, int x, int y) {
    if (frozen) {
        frozen = false;
        start();
    } else {
        frozen = true;

        //Σταμάτα το animating thread.
        animatorThread = null;
    }
    return true;
}

public void run() {
     // Η παρακάτω μέθοδος απλά χαμηλώνει
     // την προτεραιότητα του animation thread
     // για λόγους ευγένειας προς τις άλλες διεργασίες
    // του συστήματος.
    Thread.currentThread().setPriority(Thread.MIN_PRIORITY);
```

```
    //Κράτα την τρέχουσα ώρα
    long startTime = System.currentTimeMillis();

    //animation loop.
    while (Thread.currentThread() == animatorThread) {

        frameNumber++;
        repaint();

        //Μην καθυστερείς καθόλου αν έχουμε αργήσει
        try {
            startTime += delay;
            Thread.sleep(Math.max(0,
                         startTime-System.currentTimeMillis()));
        } catch (InterruptedException e) {
            break;
        }
    }
}


public void paint(Graphics g) {
    update(g);
}

// Η update() έχει επανακαθορισετεί σύμφωνα με την
// πρώτη από τις δύο τεχνικές που παρουσιάσαμε ώστε να
// μην καθαρίζει το background της οθόνης χωρίς λόγο.
// Αντίθετα καθαρίζει το backround στη ψεύτικη οθόνη
// και σε αυτήν  ζωγραφίζει το νέο frame. Στο τέλος
// περνάει το αποτέλεσμα κατευθείαν στη πραγματική
//οθόνη σύμφωνα με τη δεύτερη τεχνική
public void update(Graphics g) {
    Dimension d = size();

    // Δημιούργησε την ψεύτικη οθόνη
    if ( (offGraphics == null)
      || (d.width != offDimension.width)
      || (d.height != offDimension.height) ) {
        offDimension = d;
        offImage = createImage(d.width, d.height);
        offGraphics = offImage.getGraphics();
    }

    //Σβήσε το παλιό frame,  αλλά στη ψεύτικη οθοόνη
    offGraphics.setColor(getBackground());
    offGraphics.fillRect(0, 0, d.width, d.height);
    offGraphics.setColor(Color.black);

    // Ζωγράφισε το νέο frame
    offGraphics.drawImage(images[frameNumber % 10], 0, 0, this);

    // Τώρα πέρνα την ψεύτικη οθόνη στην πραγματική
    g.drawImage(offImage, 0, 0, this);
}
```

```
}
```

10. Είσοδος/Έξοδος - Streams

Οι εφαρμογές που εκτελούνται σε ένα υπολογιστικό περιβάλλον, σε οποιαδήποτε γλώσσα και αν έχουν γραφεί, είναι είτε παραγωγοί δεδομένων, είτε καταναλωτές, ή, στην συνήθη περίπτωση και τα δύο. Δέχονται κάποια δεδομένα (κατανάλωση), ακολουθεί επεξεργασία τους και ύστερα τα εξάγουν (παραγωγή) για περαιτέρω χρήση.

10.1 Γενική Περιγραφή

Ο τρόπος με τον οποίο οι εφαρμογές που έχουν γραφτεί στην Java, εισάγουν και εξάγουν δεδομένα, επικοινωνώντας μεταξύ τους, με το σύστημα αρχείων (file system) ή με κάποια συσκευή (device), γίνεται με την βοήθεια των Streams. Ένα stream δηλ. δεν είναι τίποτε άλλο από ένα γενικό κανάλι, στο οποίο υπάρχει ροή δεδομένων.

Η Java διαθέτει αρκετά είδη από streams, καθένα από τα οποία χρησιμοποιείται από διαφορετικά είδη δεδομένων. Τα στάνταρτ streams εισόδου/εξόδου InputStream και OutputStream, αποτελούν την βάση για όλα τα υπόλοιπα. Ίσως τα πιο συχνά χρησιμοποιούμενα streams είναι τα DataInputStream και DataOutputStream τα οποία δίνουν την δυνατότητα, το μεν πρώτο, της ανάγνωσης - εισόδου πρωτογενών τύπων δεδομένων της Java σε μία εφαρμογή, ανεξάρτητα από το είδος της μηχανής (υπολογιστής και λειτουργικό σύστημα) πάνω στο οποίο εκτελείται η εφαρμογή. Αντίστοιχα, το δεύτερο χρησιμοποιείται για την εγγραφή - έξοδο πρωτογενών τύπων της Java. Πιο αυστηρά, τα DataInputStream και DataOutputStream αναπαριστούν Unicode αλφαριθμητικά (string), σε μορφή που είναι μια ελαφρά τροποποίηση της UTF-8.

Ως άλλα είδη από streams, θα μπορούσαμε να αναφέρουμε τους απομονωτές αρχείων (file buffers), οι οποίοι συχνά χρησιμοποιούνται για να αυξήσουν την ταχύτητα και απόδοση του συστήματος, στην περίπτωση που λαμβάνει χώρα I/O (είσοδος/έξοδος). Τα BufferedInputStream και BufferedOutputStream διάβαζουν και γράφουν τμήματα δεδομένων (το μέγεθος των οποίων μπορεί να καθοριστεί) την φορά. Όταν γίνεται ανάγνωση από ή εγγραφή σε buffered stream, ο χρήστης στην ουσία ασχολείται με τον buffer και όχι με τα πραγματικά δεδομένα του stream. Θα πρέπει όμως, σε τακτά χρονικά διαστήματα, να καλείται η μέθοδος flush των buffers, έτσι ώστε να είμαστε σίγουροι ότι όλα τα δεδομένα στον buffer έχουν διαβαστεί από ή γραφτεί στο σύστημα αρχείων (file system). Ακόμη, σε περίπτωση που θέλουμε να χειριστούμε απευθείας αρχεία από το τοπικό file system, τα FileInputStream και FileOutputStream χρησιμοποιούνται για το άνοιγμα, την ανάγνωση και την εγγραφή αρχείων.

10.2 Παράδειγμα εισόδου με την κλάση System

Μία από τις κλάσεις που η Java περιλαμβάνει σε κάθε applet ή εφαρμογή, ανεξάρτητα από το αν την έχουμε κάνει import στην αρχή ή όχι, είναι η κλάση System. Αυτή περιέχει αρκετά βασικά αντικείμενα, τα οποία μπορούν να χρησιμοποιηθούν για λειτουργίες I/O. Ένα από αυτά είναι και το αντικείμενο System.in, τύπου InputStream και βρίσκει εφαρμογή σε λειτουργίες εισόδου από το στάνταρ stream εισόδου (συνήθως το πληκτρολόγιο). Επειδή όμως το stream είναι μια ακολουθία από χαρακτήρες οι οποίοι έχουν ληφθεί από κάποια πηγή, είναι απολύτως φυσικό αυτή να μπορεί να είναι και κάποιο αρχείο, το modem, ή και οποιοδήποτε άλλο περιφερειακό.

Το παραπάνω stream εισόδου, έχει αρκετές μεθόδους με τις οποίες μπορεί να γίνει χειρισμός των δεδομένων εισόδου. Υπάρχουν μέθοδοι για λήψη συγκεκριμένων χαρακτήρων από strings, για είσοδο ακεραίων (integer) καθώς και άλλων τύπων αριθμών, ακόμη και μέθοδος για είσοδο απλών bytes. Το παρακάτω κομμάτι κώδικα διαβάζει string από το πληκτρολόγιο :

```
public class InputTest() {
        String str;
```

```
        public static void main(String args[])
        {
                str = System.in.getln();
        }

}
```

10.3 Παράδειγμα εξόδου με την κλάση System

Όπως και η είσοδος, έτσι και η έξοδος γίνεται με την βοήθεια των streams. Στην συγκεκριμένη περίπτωση, η πηγή της ακολουθίας των χαρακτήρων είναι η ίδια η εφαρμογή που θέλει λειτουργία εξόδου, ενώ ο προορισμός είναι το standard output, που χρησιμοποιεί το λειτουργικό σύστημα. Αυτό είναι συνήθως το monitor, αλλά, όπως και πιο πάνω, μπορεί να είναι και άλλες συσκευές, ο εκτυπωτής για παράδειγμα. Το αντικείμενο της κλάσης System για έξοδο είναι το System.out και είναι τύπου OutputStream. Το παρακάτω κομμάτι κώδικα, επέκταση του προηγουμένου, εκτελεί και έξοδο στο standard output.

```
public class InputTest() {
        String str;

        public static void main(String args[])
        {
                str = System.in.getln();

                System.out.println(str);
        }

}
```

Σημείωση : αν ο παραπάνω κώδικας τροποποιηθεί ελαφρά και εκτελεστεί μέσα από ένα applet, με την βοήθεια π.χ. του Netscape Navigator, τότε ως standard έξοδος θα τεθεί η Java Console του Netscape, στην οποία θα εμφανιστεί και το μήνυμα, αφού εδώ δεν έχουμε γραμμή εντολών (command line).

10.4 Κατασκευαστής και Μέθοδοι του DataInputStream

Παρακάτω παραθέτουμε τον κατασκευαστή και μερικές χρήσιμες μεθόδους της κλάσης DataInputStream.

Constructor :

public DataInputStream(InputStream in)

Δημιουργεί ένα καινούργιο αντικείμενο τύπου data input stream για ανάγνωση από το input stream που προσδιορίζεται από την παράμετρο.

Παράμετροι:

in - το input stream.

Μέθοδοι :

read(byte[])
Είσοδος byte.length bytes δεδομένων από αυτό το data input stream, σε έναν πίνακα από bytes

readBoolean()
Είσοδος μιας boolean τιμής από αυτό το data input stream.

readByte()
Είσοδος μιας προσημασμένης 8-bit τιμής από αυτό το data input stream.

readChar()
Είσοδος ενός Unicode χαρακτήρα από αυτό το data input stream.
readFloat()
Είσοδος ενός float από αυτό το data input stream.
readInt()
Είσοδος ενός προσημασμένου 32-bit ακεραίου (integer) από αυτό το data input stream.
readLine()
Είσοδος της επόμενης γραμμής κειμένου από αυτό το data input stream.
readLong()
Είσοδος ενός προσημασμένου 64-bit ακεραίου (integer) από αυτό το data input stream.

10.5 Κατασκευαστής και Μέθοδοι του DataOutputStream

Παρακάτω παραθέτουμε τον κατασκευαστή και μερικές χρήσιμες μεθόδους της κλάσης DataOutputStream.

Constructor :

public DataOutputStream(OutputStream out)
Δημιουργεί ένα καινούργιο αντικείμενο τύπου data output stream για εγγραφή στο output stream που προσδιορίζεται από την παράμετρο.

Παράμετρος :
out - το output stream.

Μέθοδοι :

flush()
"Καθαρίζει", δηλ. εκτελεί αναγκαστική μεταφορά των δεδομένων του output stream. Χρησιμοποιείται για το άδειασμα του buffer.
size()
Επιστρέφει τον αριθμό των bytes που εγράφησαν στο data output stream.
writeChar(int)
Γράφει ένα χαρακτήρα στο stream εξόδου σαν μία 2-byte τιμή, με το πιο σημαντικό byte, πρώτο.
writeDouble(double)
Μετατρέπει την double παράμετρο που δέχεται σε ένα long ακέραιο με την doubleToLongBits μέθοδο της κλάσης Double και ύστερα γράφει αυτήν την long τιμή στο stream εξόδου σαν μια 8-byte ποσότητα, με το πιο σημαντικό της byte, πρώτο.
writeInt(int)
Γράφει έναν ακέραιο (int) στο stream εξόδου με μορφή τεσσάρων bytes, το πιο σημαντικό byte, πρώτο.

11. Δικτυακός Προγραμματισμός στη Java

11.1 Uniform Resource Locator (URL)

Όποιος έχει περιηγηθεί στον Παγκόσμιο Ιστό (World Wide Web), έχει ήδη ακούσει και χρησιμοποιήσει τον όρο URL, για να προσπελάσει διάφορες HTML σελίδες στο Web. Ακολουθεί ένας απλός, αλλά περιεκτικός ορισμός :

Ορισμός : URL είναι ένα ακρωνύμιο για το Uniform Resource Locator και το οποίο αποτελεί μια αναφορά (μια διεύθυνση) στο Internet.

Είναι συχνά εύκολο (αν και όχι απόλυτα ακριβές) να σκεφτόμαστε το URL, ως το όνομα ενός αρχείου τοποθετημένου κάπου στο δίκτυο, γιατί τα περισσότερα URLs αναφέρονται σε αρχεία κάποιου υπολογιστή συνδεδεμένου στο δίκτυο. Παράδειγμα ενός URL είναι η διεύθυνση της κεντρικής σελίδας του Ε.Μ.Π. στο Internet : *http://www.ntua.gr* .

Το παραπάνω URL, όπως άλλωστε τα περισσότερα, αποτελείται από δύο κύρια μέρη :

- τον προσδιοριστή πρωτοκόλλου (protocol identifier) , π.χ. http και
- το όνομα της πηγής/πόρου (resource name), π.χ. //www.ntua.gr .

Το protocol identifier προσδιορίζει το όνομα του πρωτοκόλλου, με τους κανόνες του οποίου θα προσπελαστεί ο πόρος. Για παράδειγμα, το http είναι το πρωτόκολλο για την προσπέλαση υπερκειμένου (hypertext documents). Άλλα γνωστά πρωτόκολλα είναι το ftp για μεταφορά αρχείων, το news για ανάγνωση νέων, το file και gopher.

Το όνομα του πόρου αποτελεί την πλήρη διεύθυνσή του στο Internet. Η μορφή του ονόματος εξαρτάται αποκλειστικά από το πρωτόκολλο που χρησιμοποιείται για την προσπέλασή του, όμως τα περισσότερα ονόματα περιλαμβάνουν ένα από τα ακόλουθα στοιχεία :

- όνομα host : το όνομα του υπολογιστή στον οποίο αναζητούμε τον πόρο.
- όνομα αρχείου (filename) : η πλήρης διαδρομή στο αρχείο που ζητάμε, στον host.
- αριθμό θύρας (port number) : η θύρα στην οποία αναζητούμε τον πόρο (αυτό είναι συνήθως προαιρετικό).
- αναφορά (reference) : αναφορά σε συγκεκριμένο σημείο στο αρχείο που ζητούμε.

Στα πιο πολλά πρωτόκολλα, τα δύο πρώτα στοιχεία είναι υποχρεωτικά, ενώ τα δύο επόμενα παραλείπονται. Το πακέτο java.net περιέχει μια κλάση με όνομα URL, η οποία χρησιμοποιείται από τα προγράμματα για τον χειρισμό URL διευθύνσεων.

11.1.1 Δημιουργία URL σχετικού (relative) με ένα άλλο

Ένα σχετικό URL περιέχει μόνο τις πληροφορίες εκείνες που είναι απαραίτητες για να προσεγγίσει μία πηγή ή το περιεχόμενο ενός άλλου URL. Οι specifications των σχετικών URL χρησιμοποιούνται συχνά σε HTML αρχεία. Για παράδειγμα υποθέτουμε ότι δημιουργούμε ένα HTML αρχείο το οποίο ονομάζεται JoesHomePage.html. Μέσα στη συγκεκριμένη σελίδα υπάρχουν links σε άλλες σελίδες, όπως PicturesOfMe.html και MyKids.html,οι οποίες βρίσκονται στο ίδιο μηχάνημα και στο ίδιο directory με την JoesHomePage.html. Τα links στις σελίδες PicturesOfMe.html και MyKids.html από τη JoesHomePage.html μπορούν να προσδιοριστούν ως ονόματα αρχείων ως εξής:

```
<a href="PicturesOfMe.html">Pictures of Me</a>
<a href="MyKids.html">Pictures of My Kids</a>
```

Αυτές οι διευθύνσεις URL είναι σχετικά URL, δηλαδή τα URL είναι ορισμένα ως σχετικά προς το αρχείο στο οποίο περιέχονται (εδώ η JoesHomePage.html). Στα προγράμματα Java, μπορούμε να δημιουργήσουμε ένα αντικείμενο URL από τη specification ενός σχετικού URL.

Για παράδειγμα, έστω ότι ξέρουμε δύο URL του gamelan site:

```
http://www.gamelan.com/pages/Gamelan.game.html
http://www.gamelan.com/pages/Gamelan.net.html
```

Από αυτές τις σελίδες μπορούμε να δημιουργήσουμε αντικείμενα URL σχετικά με την κοινή τους βάση URL: http://www.gamelan.com/pages/ ως εξής:

```
URL gamelan = new URL("http://www.gamelan.com/pages/");
URL gamelanGames = new URL(gamelan, "Gamelan.game.html");
URL gamelanNetwork = new URL(gamelan, "Gamelan.net.html");
```

Αυτός ο snippet κώδικας χρησιμοποιεί τον κατασκευαστή URL που επιτρέπει τη δημιουργία ενός αντικειμένου URL από ένα άλλο αντικείμενο URL (συγκεκριμένα τη βάση) και τη specification ενός σχετικού URL. Η γενική μορφή ενός τέτοιου κατασκευαστή είναι:

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

Το πρώτο όρισμα είναι ένα URL αντικείμενο που προσδιορίζει τη βάση του καινούριου URL. Το δεύτερο όρισμα είναι ένα String που προσδιορίζει το υπόλοιπο του resource ονόματος του σχετικού με τη βάση. Στην περίπτωση που η βάση URL είναι κενή, ο κατασκευαστής αυτός χειρίζεται (αντιμετωπίζει) το σχετικό URL σαν ένα απόλυτο URL specification. Αντίστροφα, στην περίπτωση που το σχετικό URL είναι ένα απόλυτο URL specification ο κατασκευαστής αγνοεί τη βάση URL.

Πρέπει να αναφέρουμε ότι αυτός ο κατασκευαστής είναι επίσης χρήσιμος στη δημιουργία URL αντικειμένων για συγκεκριμένες αναφορές(anchors) μέσα σε ένα αρχείο. Για παράδειγμα υποθέτουμε ότι το αρχείο Gamelan.net.html έχει μια συγκεκριμένη αναφορά που ονομάζεται BOTTOM στην αρχή του αρχείου. Για τη συγκεκριμένη αναφορά μπορούμε να χρησιμοποιήσουμε το σχετικό URL, ώστε να δημιουργήσουμε ένα URL αντικείμενο ως εξής:

```
URL gamelanNetworkBottom = new URL(gamelanNetwork, "#BOTTOM");
```

11.1.2 Άλλοι κατασκευαστές URL

Η class URL διαθέτει ακόμα δύο κατασκευαστές για τη δημιουργία ενός URL αντικειμένου. Αυτοί οι κατασκευαστές είναι χρήσιμοι για κάποιον που εργάζεται με URL, όπως τα HTTP URLs, τα οποία διαθέτουν όνομα host, όνομα αρχείου, αριθμό θύρας και συστατικά αναφοράς στο resource name portion του URL. Αυτοί οι δύο κατασκευαστές είναι χρήσιμοι όταν δεν υπάρχει string που να περιλαμβάνει ολοκληρωμένο το URL specification, ενώ είναι γνωστά διάφορα συστατικά του URL.

Για παράδειγμα έστω ότι σχεδιάζουμε ένα network browsing panel παρεμφερές με ένα αρχείο browsing panel το οποίο επιτρέπει στους χρήστες να διαλέξουν πρωτόκολλο, όνομα host, αριθμό θύρας και όνομα αρχείου. Από τα συστατικά του panel έχουμε τη δυνατότητα να φτιάξουμε ένα URL. Ο πρώτος κατασκευαστής δημιουργεί ένα URL από το πρωτόκολλο, το όνομα host και το όνομα αρχείου. Το παρακάτω κομμάτι κώδικα δημιουργεί ένα URL στο αρχείο Gamelan.net.html στο site Gamelan:

```
new URL("http", "www.gamelan.com", "/pages/Gamelan.net.html");
```

Το παραπάνω είναι αντίστοιχο με το εξής:

```
new URL("http://www.gamelan.com/pages/Gamelan.net.html");
```

Το πρώτο όρισμα είναι το πρωτόκολλο, το δεύτερο το όνομα host και το τελευταίο είναι το path του αρχείου. Οφείλουμε να παρατηρήσουμε ότι το όνομα αρχείου περιέχει ένα forward slash στην αρχή. Αυτό δείχνει ότι το όνομα αρχείου προσδιορίζεται από τη ρίζα του host.

Ο τελικός κατασκευαστής URL προσθέτει τον αριθμό θύρας στη λίστα των ορισμάτων που χρησιμοποιεί ο προαναφερόμενος κατασκευαστής:

```
URL gamelan = new URL("http", "www.gamelan.com", 80,
                      "pages/Gamelan.net.html");
```

Αυτό δημιουργεί ένα αντικείμενο URL για το επόμενο URL:

```
http://www.gamelan.com:80/pages/Gamelan.net.html
```

Στην περίπτωση που δημιουργούμε ένα αντικείμενο URL χρησιμοποιώντας κάποιον από τους παραπάνω κατασκευαστές, οι μέθοδοι toString() και toExternalForm() που διαθέτει η κλάση URL, μας παράσχουν ένα String με την πλήρη διεύθυνση του πόρου.

11.1.3 MalformedURLException

Κάθε ένας από τους τέσσερις κατασκευαστές URL εγείρει μία MalformedURLException αν το όρισμα του κατασκευαστή αναφέρεται σε ένα κενό ή άγνωστο πρωτόκολλο. Τυπικά, επιθυμούμε να προλάβουμε και να χειριστούμε αυτή την εξαίρεση τοποθετώντας τις δηλώσεις του κατασκευαστή URL σε ένα try/catch pair, ως εξής:

```
try {
        URL myURL = new URL(. . .)
    } catch (MalformedURLException e) {
        . . .
        // exception handler code here
        . . .
    }
```

Τέλος πρέπει να τονίσουμε ότι τα URLs είναι "μιας εγγραφής" αντικείμενα. Από τη στιγμή που δημιουργούμε ένα URL αντικείμενο, δεν έχουμε τη δυνατότητα να μεταβάλλουμε καμία από τις ιδιότητές του(ή γνωρίσματα)(πρωτόκολλο, όνομα host, όνομα αρχείου ή αριθμό θύρας).

11.1.4 Ανάλυση ενός URL

Μερικές μέθοδοι της κλάσης του URL, με τις οποίες γίνεται (αυτόματα) ανάλυση της διεύθυνσης που περιέχει το αντικείμενό της, είναι:

getProtocol()
Επιστρέφεται ένα String με το χρησιμοποιούμενο πρωτόκολλο.
getHost()
Επιστρέφεται ένα String με το όνομα του host.
getPort()
Επιστρέφεται ένας ακέραιος (integer) με τον αριθμό της θύρας (port). Αν δεν έχει δηλωθεί port number στη διεύθυνση του URL τότε επιστρέφεται -1.
getFile()
Επιστρέφεται ένα String με το filename.
getRef()
Επιστρέφεται το συστατικό αναφοράς του URL.

Μπορούμε να χρησιμοποιήσουμε αυτές τις *getXXX* μεθόδους, ώστε να λάβουμε πληροφορίες για το URL αδιαφορώντας για τον κατασκευαστή που χρησιμοποιήσαμε για τη δημιουργία του αντικειμένου URL.

Η URL class, σε συνδυασμό με τις προηγούμενες μεθόδους, μας επιτρέπει να μην αναλύουμε ξανά τα URLs. Με δεδομένο οποιοδήποτε χαρακτηριστικό, μπορούμε να δημιουργήσουμε ένα

νέο αντικείμενο URL και απλά να καλέσουμε όποια μέθοδο θέλουμε για να πάρουμε την πληροφορία που θέλουμε. Το ακόλουθο πρόγραμμα είναι ένα παράδειγμα δημιουργίας ενός URL από ένα string χαρακτηριστικό και χρήσης των προηγούμενων μεθόδων για την ανάλυση του 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/intro.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());
   }
}
```

Ακολουθεί η έξοδος του προηγούμενου προγράμματος:

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

11.1.5 Απευθείας διάβασμα από URL

Μετά την επιτυχή δημιουργία ενός αντικειμένου URL, κλήση της μεθόδου openStream() επιστρέφει ένα stream, απ' το οποίο μπορούμε να διαβάσουμε τα περιεχόμενά του. Πιο συγκεκριμένα, η openStream() μέθοδος, επιστρέφει ένα Java.io.InputStream αντικείμενο, μέσω του οποίο μπορούμε να διαβάσουμε το URL, με τις γνωστές, από την Εισαγωγή στα streams, μεθόδους των streams.
Το πρόγραμμα που ακολουθεί, κάνει ακριβώς αυτό. Διαβάζει τα περιεχόμενα της διεύθυνσης που του υποδεικνύουμε στην αρχή και τα τυπώνει στην οθόνη, με την μορφή απλού κειμένου και εντολών HTML.

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

class OpenStreamTest {
        public static void main(String args[]) {
    try {
          DataInputStream in = new DataInputStream(System.in);
      DataInputStream dis;
      String inputLine;
          String address;

          // Get the address from the user.
          System.out.print("Give the address : ");
          address = in.readLine();

      URL adr = new URL(address);
      dis = new DataInputStream(adr.openStream());
      while ((inputLine = dis.readLine()) != null) {
        System.out.println(inputLine);
      }
```

```
        dis.close();
      } catch (MalformedURLException me) {
         System.out.println("MalformedURLException: "+ me);
      } catch (IOException ioe) {
         System.out.println("IOException: " + ioe);
      }
          }                    // End of main()
}                              // End of class
```

Χωρίς να προχωρήσουμε σε λεπτομέρειες, αναφέρουμε ότι μπορούμε, εκτός από διάβασμα, να γράψουμε σε URL, ή ακόμη και να υποβάλουμε ερωτήσεις (query), σε συνδεδεμένο με αυτό cgi-script.

11.1.6 Σύνδεση με URL

Μετά την επιτυχή δημιουργία ενός URL αντικειμένου η σύνδεση με αυτό επιτυγχάνεται με κλήση της μεθόδου openConnection. Όταν γίνεται η σύνδεση, αρχικοποιείται ένας δεσμός επικοινωνίας στο διαδίκτυο μεταξύ του δικού μας προγράμματος σε Java και του URL .Με το παράδειγμα που ακολουθεί μπορούμε να δημιουργήσουμε σύνδεση με την κεντρική σελίδα του Ε.Μ.Π. στο δίκτυο:

```
try {
   URL ntua = new URL("http://www.ntua.gr");
   ntua.openConnection();
} catch (MalformedURLException e) {     // new URL() failed
   . . .
} catch (IOException e) {               // openConnection() failed
   . . .
}
```

Η μέθοδος openConnection δημιουργεί ένα καινούριο URLConnection, αν δεν υπάρχει άλλο που να καλύπτει τις απαιτήσεις μας, το αρχικοποιεί, στη συνέχεια συνδέεται με το URL κι επιστρέφει το αντικείμενο URLConnection. Αν προκύψει κάποιο πρόβλημα, η μέθοδος openConnection εγείρει μία IOΕξαίρεση.

11.1.7 Ανάγνωση από ένα URLConnection

Αντί να πάρουμε ένα input stream απευθείας από το URL, μπορούμε να δημιουργήσουμε σύνδεση (connection) με το URL και να πάρουμε ένα input stream από τη σύνδεση αυτή, διαδικασία που εφαρμόζεται στο πρόγραμμα που ακολουθεί. Επίσης το πρόγραμμα αυτό δημιουργεί ένα BufferedReader στο input stream, από το οποίο και διαβάζει.

```
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();
        }
    }
```

Η ανάγνωση από ένα URLConnection μπορεί να είναι πιο χρήσιμη από την απευθείας ανάγνωση από το URL κι αυτό συμβαίνει γιατί μπορούμε να χρησιμοποιήσουμε το αντικείμενο URLConnection συγχρόνως και από άλλες λειτουργίες.

11.1.8 Εγγραφή σε ένα URLConnection

Πολλές σελίδες HTML περιέχουν φόρμες, δηλαδή πεδία κειμένου και άλλα αντικείμενα τα οποία μας επιτρέπουν να εισάγουμε δεδομένα στον εξυπηρετητή (server). Μόλις πληκτρολογήσουμε την απαιτούμενη πληροφορία και αρχικοποιήσουμε την έρευνα πατώντας το κατάλληλο κουμπί, ο Web browser γράφει τα δεδομένα σε ένα URL στο διαδίκτυο, ενώ ένα cgi-bin script, συνήθως, στον εξυπηρετητή (server) λαμβάνει τα δεδομένα , τα προωθεί και στη συνέχεια στέλνει μία απάντηση, συνήθως με τη μορφή μιας νέας HTML σελίδας.
Πολλά cgi-bin scripts χρησιμοποιούν την POST μέθοδο για να διαβάσουν τα δεδομένα από τον πελάτη, για αυτό άλλωστε η εγγραφή σε ένα URL συχνά αποκαλείται και posting σε ένα url. Ένα πρόγραμμα σε Java μπορεί να αλληλεπιδράσει με cgi-scripts ακόμα και στη μεριά του εξυπηρετητή, απλά πρέπει να είναι ικανό να γράψει σε ένα URL, έτσι ώστε να διοχετεύσει δεδομένα στον εξυπηρετητή. Αυτό μπορεί να γίνει με τα παρακάτω βήματα:

- Δημιουργία ενός URL.
- Δημιουργία σύνδεσης με το URL.
- Ορισμός ικανότητας εξόδου του URLConnection.
- Λήψη ενός ρεύματος (stream) εξόδου από τη σύνδεση. Αυτό το stream εξόδου συνδέεται στο σύνηθες stream εισόδου του cgi-bin script στον εξυπηρετητή.
- Εγγραφή στο stream εισόδου.
- Κλείσιμο του stream εξόδου.

Ο Hassan Schroeder, που είναι μέλος της ομάδας ανάπτυξης λογισμικού σε JAVA, δημιούργησε ένα cgi-bin script, που το ονόμασε *backwards* και που υπάρχει στο site:
http://java.sun.com/cgi-bin/backwards.
Με τη βοήθεια αυτού, δοκιμάζουμε το πρόγραμμα που δίνεται στη συνέχεια. Επίσης, μπορούμε να βάλουμε αυτό το script στο δίκτυο, να το ονομάσουμε *backwards* και να τρέξουμε το πρόγραμμα τοπικά.
Αυτό το πρόγραμμα διαβάζει ένα string από τη standard είσοδο, το αντιστρέφει και δίνει το αποτέλεσμα στη standard έξοδο. Ως είσοδο χρειάζεται ένα τύπο της μορφής: string=string_to_reverse, όπου ως string_to_reverse θεωρούμε το string του οποίου θέλουμε να αντιστρέψουμε τους χαρακτήρες.

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

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

              if (args.length != 1) {
                 System.err.println("Usage:  java Reverse string_to_reverse");
                 System.exit(1);
              }

              String stringToReverse = URLEncoder.encode(args[0]);

              URL url = new URL("http://java.sun.com/cgi-bin/backwards");
              URLConnection connection = url.openConnection();
              connection.setDoOutput(true);

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

```
        out.println("string=" + stringToReverse);
        out.close();

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

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

        in.close();
     }
  }
```

Ας δούμε πως λειτουργεί το πρόγραμμα .

```
if (args.length != 1) {
   System.err.println("Usage:  java Reverse " +
                      "string_to_reverse");
   System.exit(-1);
}
String stringToReverse = URLEncoder.encode(args[0]);
```

Με αυτές τις δηλώσεις βεβαιώνεται ότι ο χρήστης δίνει μία και μόνο γραμμή διαχείρισης και την κωδικοποιεί. Η γραμμή διαχείρισης είναι το string που θα αντιστραφεί από το cgi-bin script backwards. Μπορεί να περιέχει κενά ή άλλους μη αλφαριθμητικούς χαρακτήρες. Αυτοί οι χαρακτήρες πρέπει να κωδικοποιηθούν, καθώς το string βρίσκεται στο δρόμο για τον server. Τέτοιοι χαρακτήρες κωδικοποιούνται από τις μεθόδους της κλάσης URLEncoder.
Στη συνέχεια, το πρόγραμμα δημιουργεί το αντικείμενο URL, ανοίγει την URLConnection και θέτει την επικοινωνία ώστε να μπορεί να γράψει σε αυτό.

```
URL url = new URL("http://java.sun.com/cgi-bin/backwards");
URLConnection c = url.openConnection();
c.setDoOutput(true);
```

Τότε το πρόγραμμα δημιουργεί ένα ρεύμα (stream) εξόδου στην επικοινωνία και ανοίγει την PrintWriter σε αυτό:

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

Αν το URL δεν υποστηρίζει έξοδο, η μέθοδος getOutputStream εγείρει μια εξαίρεση τύπου: UnknownServiceException. Αν το URL υποστηρίζει έξοδο, τότε η παραπάνω μέθοδος επιστρέφει ένα ρεύμα (stream) εξόδου που συνδέεται με το standard ρεύμα εισόδου του URL στην πλευρά του εξυπηρετητή (server). Υπενθυμίζουμε ότι η έξοδος του πελάτη είναι είσοδος του εξυπηρετητή.
Ακολούθως, το πρόγραμμα γράφει τις απαραίτητες πληροφορίες στο ρεύμα (stream) εξόδου και έπειτα το κλείνει.

```
out.println("string=" + stringToReverse);
out.close();
```

Αυτός ο κώδικας γράφει στο ρεύμα (stream) εξόδου χρησιμοποιώντας τη μέθοδο println. Έτσι, βλέπουμε πως το να γράφει κανείς δεδομένα σε ένα URL είναι το ίδιο απλό με το να γράφει δεδομένα στο ρεύμα (stream). Τα δεδομένα που γράφονται στο ρεύμα (stream) εξόδου από την πλευρά του πελάτη αποτελούν την είσοδο του backward script στην πλευρά του εξυπηρετητή. Το πρόγραμμα Reverse κατασκευάζει την είσοδο στη μορφή που απαιτείται από το script για να συνενώσει το string με το κωδικοποιημένο string που θα αντιστραφεί.

Συχνά, όταν γράφουμε σε ένα URL, περνάμε πληροφορίες σε ένα cgi-bin script όπως στο παράδειγμα. Αυτό το script διαβάζει τις πληροφορίες που του γράφουμε, εκτελεί τις διαδικασίες που πρέπει και τις στέλνει πίσω, μέσω του ίδιου URL. Έτσι είναι πιθανόν, να επιθυμούμε να διαβάσουμε από ένα URL αφού πρώτα έχουμε γράψει σε αυτό. Κάτι τέτοιο κάνει το πρόγραμμα Reverse που ακολουθεί:

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

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

Όταν εκτελεστεί το πρόγραμμα Reverse χρησιμοποιώντας ως όρισμα το "Reverse Me", τότε η έξοδος θα έχει τη μορφή:

```
Reverse Me
 reversed is:
eM esreveR
```

11.2 Sockets

11.2.1 Το μοντέλο Client - Server και μερικοί ορισμοί

Το ευρύτερα διαδεδομένο μοντέλο ανάπτυξης δικτυακών εφαρμογών είναι το μοντέλο του πελάτη - εξυπηρετητή (client - server). Ο εξυπηρετητής είναι μια διεργασία, η οποία εκτελείται σε έναν υπολογιστή και αναμένει να συνδεθεί σε αυτήν κάποιο πρόγραμμα - ο πελάτης, όπως ονομάζεται -, για να του παράσχει υπηρεσίες. Ένα τυπικό σενάριο που ακολουθείται συνήθως, είναι το εξής :

- Η διεργασία - εξυπηρετητής αρχίζει να εκτελείται σε κάποιον υπολογιστή. Μετά την αρχικοποίησή της, πέφτει σε "λήθαργο", αναμένοντας μία διεργασία - πελάτη να επικοινωνήσει μαζί της και να της ζητήσει κάποια υπηρεσία.
- Μία διεργασία - πελάτης αρχίζει να εκτελείται, είτε στο ίδιο σύστημα, είτε σε κάποιο απομακρυσμένο, το οποίο συνδέεται με τον υπολογιστή στον οποίο "τρέχει" ο εξυπηρετητής μέσω δικτύου. Η διεργασία πελάτης στέλνει μια αίτηση, μέσω του δικτύου, στον εξυπηρετητή, ζητώντας του κάποιου είδους υπηρεσία (π.χ. μεταφορά αρχείου, απομακρυσμένη εκτύπωση, ανάγνωση και αποστολή mail και άλλες).
- Ταυτόχρονα με την εξυπηρέτηση κάποιου πελάτη, ο server έχει την δυνατότητα να δέχεται και αιτήσεις άλλων πελατών προς εξυπηρέτηση. Όταν ο εξυπηρετητής τελειώσει με όλους τους πελάτες, τότε ξαναπέφτει σε "λήθαργο", περιμένοντας για μια καινούργια αίτηση και η διαδικασία ξαναρχίζει από την αρχή.

Ορίζουμε ως *σύνδεση*, τον επικοινωνιακό δίαυλο μεταξύ δύο διεργασιών. Την σύνδεση μπορούμε να την θεωρήσουμε ως μία πεντάδα, που περιγράφεται ως εξής :

{ πρωτόκολλο, τοπική-διεύθυνση, τοπική-διεργασία, απομακρυσμένη-διεύθυνση,
απομακρυσμένη-διεργασία }

Το πρωτόκολλο αναφέρεται στο σύνολο των κανόνων που διέπουν την επικοινωνία. Η τοπική-διεύθυνση και απομακρυσμένη-διεύθυνση, προσδιορίζουν την ταυτότητα των υποδικτύων και των υπολογιστών, στους οποίους εκτελούνται οι επικοινωνούσες διεργασίες. Η τοπική-διεργασία και απομακρυσμένη-διεργασία, προσδιορίζουν την ταυτότητα των διεργασιών που θα επικοινωνούν, καθώς σε έναν υπολογιστή, μπορούν να εκτελούνται περισσότερες της μιας διεργασίες. Κάθε μία από αυτές τις διεργασίες που εκτελούνται στον ίδιο host και που χρειάζονται επικοινωνία μέσω δικτύου, λαμβάνει έναν 16-bit ακέραιο αριθμό, ο οποίος αναπαριστά την θύρα (port number) της διεργασίας και κατ' επέκταση, της υπηρεσίας.

Ορίζουμε, επίσης ως *μισή σύνδεση (half association),* είτε το σύνολο { πρωτόκολλο, τοπική-διεύθυνση, τοπική-διεργασία }, είτε το σύνολο { πρωτόκολλο, απομακρυσμένη-διεύθυνση, απομακρυσμένη-διεργασία }. Η μισή σύνδεση ονομάζεται αλλιώς και socket.

Ορισμός : Socket είναι το ένα άκρο, από έναν επικοινωνιακό δίαυλο διπλής κατεύθυνσης, μεταξύ δύο προγραμμάτων που εκτελούνται στο δίκτυο. Περιλαμβάνει το πρωτόκολλο, την διεύθυνση και τον αριθμό θύρας του άκρου.

Η έκδοση 4.1cBSD του Unix (1982), για τους υπολογιστές VAX, απ' το πανεπιστήμιο του Berkeley, πρωτοεισήγαγε το socket interface σαν μια μέθοδο επικοινωνίας απομακρυσμένων διεργασιών. Είχαν αρχικά υλοποιηθεί στην γλώσσα C του Unix, αλλά η απήχηση που γνώρισαν, επέβαλε την μεταφορά τους τόσο σε άλλα λειτουργικά συστήματα (π.χ. Winsock library για τα Microsoft Windows), όσο και σε άλλες γλώσσες προγραμματισμού (π.χ. Java).

Κάθε πρόγραμμα διαβάζει από και γράφει σε ένα socket, με τρόπο παρόμοιο της εγγραφής και ανάγνωσης αρχείων του file system. Υπάρχουν δύο είδη sockets :

- Το πρώτο ονομάζεται TCP (Transmission Control Protocol) socket και είναι μια υπηρεσία προσανατολισμένη στην σύνδεση (connection-oriented service). Μπορούμε να το θεωρήσουμε ανάλογο της τηλεφωνικής υπηρεσίας, στην οποία, μετά την εγκαθίδρυση

μιας σύνδεσης μεταξύ δύο συνομιλητών, αυτή χρησιμοποιείται μέχρι το πέρας της συζητήσεως τους.
- Το άλλο είδος ονομάζεται UDP (Unreliable Datagram Protocol) socket και είναι μια υπηρεσία χωρίς σύνδεση (connectionless service). Το ανάλογο, σε αυτήν την περίπτωση, είναι το ταχυδρομείο : μπορούμε να στείλουμε πολλά πακέτα στον ίδιο παραλήπτη, αλλά δεν είναι σίγουρο ότι όλα θα ακολουθήσουν την ίδια διαδρομή (… σύνδεση) για να φτάσουν στον προορισμό τους.

Μία ακόμη σημαντική διαφορά, μεταξύ των παραπάνω δύο ειδών, είναι ότι τα TCP sockets εξασφαλίζουν μια αξιόπιστη μεταφορά της πληροφορίας : ότι αποστέλλεται από το ένα άκρο είναι σίγουρο ότι θα φτάσει στο άλλο. Στο UDP socket όμως δεν συμβαίνει αυτό. Είναι στην ευθύνη του αποστολέα να ελέγξει ότι αυτό που έστειλε, το έλαβε τελικά ο παραλήπτης και δεν χάθηκε στον δρόμο. Από την άλλη, η σύνδεση με TCP socket απαιτεί την ανταλλαγή τριών "πακέτων χειραψίας" (handshake packets) και είναι πιο χρονοβόρα στην αρχικοποίησή της από την αντίστοιχη με UDP datagrams. Οι προηγούμενες δύο διαφορές καθορίζουν τελικά και την χρήση των δύο αυτών ειδών.

Για την αποφυγή σύγχυσης, να σημειώσουμε ότι ειδικά στην Java, ο όρος Socket χρησιμοποιείται για τα TCP sockets, στην ονοματολογία των κλάσεων και των μεθόδων, ενώ για την δήλωση των UDP sockets, χρησιμοποιείται ο όρος Datagram.

Στον παρακάτω πίνακα βλέπουμε τις διάφορες κλάσεις για τα sockets που περιέχονται στο πακέτο java.net καθώς και το είδος τους.

Μηχανισμός / Κλάση	***Περιγραφή***
Socket	TCP άκρο - πελάτης
ServerSocket	TCP άκρο - εξυπηρετητής
DatagramSocket	UDP άκρο (client & server)
DatagramPacket	UDP πακέτο
InetAddress	Διεύθυνση Internet Protocol (IP)
URL	Uniform Resource Locator
URLConnection	Σύνδεση με αντικείμενο του web.

11.2.2 Ανάπτυξη Client - Server εφαρμογής με TCP sockets

Μία από τις κλασικές υπηρεσίες που προσφέρονται στο δίκτυο, είναι η υπηρεσία αντήχησης (echo service). Αυτή χρησιμοποιεί για την σύνδεση TCP sockets και αριθμό θύρας 7, ενώ σκοπός της είναι να μεταδώσει πίσω στον πελάτη, που θα ζητήσει τέτοια υπηρεσία, ό,τι αυτός της στείλει. Δεν επεξεργάζεται δηλ. την είσοδο, απλά της στέλνει πάλι πίσω. Μία τέτοια εφαρμογή θα αναπτύξουμε εδώ σαν παράδειγμα, τόσο το πρόγραμμα της πλευράς του πελάτη, όσο και αυτό της πλευράς του εξυπηρετητή, μόνο που θα χρησιμοποιήσουμε έναν άλλο αριθμό θύρας, τον 8205.

Ο πελάτης (client)

Μια τυπική αλληλουχία βημάτων που συνήθως ακολουθούνται για το πρόγραμμα του client είναι η εξής :

Socket()	- αρχικοποίηση και σύνδεση στον server
getInputStream() και *getOutputStream()*	- σύνδεση του socket με τον στάνταρ μηχανισμό εισόδου/εξόδου στη Java, δηλ. τα streams
read() και write()	- ανάγνωση και εγγραφή στο - συνδεδεμένο με το

socket - stream
…….. - ακολουθούν και άλλες αναγνώσεις/εγγραφές
close() - κλείνει το socket και απελευθερώνεται ο αντίστοιχος πόρος του συστήματος.

Ας αναλύσουμε περαιτέρω τα βήματα που αναφέραμε, πριν να παραθέσουμε το πλήρες πρόγραμμα του πελάτη.

Η δημιουργία ενός αντικειμένου της κλάσης Socket, γίνεται, πολύ απλά, καλώντας έναν από τους οκτώ (!) κατασκευαστές (constructors) της κλάσης. Στο πρόγραμμά μας γίνεται με τον εξής κώδικα :

```
Socket s = new Socket(args[0], 8205);
```

δηλ. φτιάξε το αντικείμενο s, που είναι socket και σύνδεσέ το στην IP διεύθυνση που καθορίζεται από την πρώτη παράμετρο (args[0] - στην συγκεκριμένη περίπτωση, η IP διεύθυνση παρέχεται από τον χρήστη, ως το πρώτο όρισμα στο command line, με την εντολή εκτέλεσης του προγράμματος).

Εφόσον η σύνδεση είναι επιτυχής, μπορούμε να λάβουμε τα streams εισόδου/εξόδου του socket που δημιουργήσαμε, για να επιτελέσουμε στην συνέχεια I/O. Αυτό γίνεται γιατί, όπως προαναφέραμε, ο ενοποιημένος (και μοναδικός) μηχανισμός για I/O προγραμμάτων σε Java είναι τα streams. Για τον σκοπό αυτό, χρησιμοποιούμε τις μεθόδους getInputStream()και getOutputStream()που διαθέτει η κλάση Socket, ως εξής :

```
DataInputStream sin = new
                        DataInputStream(s.getInputStream());
DataOutputStream sout = new
                        DataOutputStream(s.getOutputStream());
```

Μετά την αρχικοποίηση και την λήψη των streams του socket, έχουμε είσοδο/έξοδο με τις γνωστές μεθόδους των streams. Π.χ. :

```
sout.println("Hello");        // Έξοδος στο socket
String message = sin.readLine();     // Είσοδος από το
                                                    // socket
```

Επίσης μπορούμε να λάβουμε από το ίδιο το socket s, μια σειρά από πληροφορίες σχετικές μ' αυτό, όπως την IP διεύθυνση που είναι συνδεδεμένο, καθώς και το port.

```
System.out.println("Connected to " +
                s.getInetAddress() + ":" + s.getPort());
```

Όταν τελειώσουμε με το I/O, καλό είναι να απελευθερώνουμε το socket που είχαμε ανοίξει, γιατί, όπως και οι χειριστές αρχείων (file handlers), είναι πολύτιμος πόρος του συστήματος και μπορεί να εξαντληθεί.

```
. . .
finally {
        try {
                if ( s != null) s.close();
        } catch(IOException ioe) {
                ;
        }
}
```

Ακολουθεί το ολοκληρωμένο πρόγραμμα του client μέρους της υπηρεσίας echo :

```
// The client program for the 'echo' service
// Written using JDK, ver. 1.1.5
```

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

public class EchoClient
{
	public static void echoclient(DataInputStream sin,
				DataOutputStream sout) throws IOException
	{
		DataInputStream in = new DataInputStream(System.in);
		PrintStream out = new PrintStream(sout);
		String line;

		while(true) {
			line = "";

			// read keyboard input and write to TCP socket
			try {
				line = in.readLine();
				out.println(line);
			} catch(IOException e) {
				System.out.println(e.getMessage());
			}

			// read TCP socket and write to terminal...
			try {
				line = sin.readLine();
				System.out.println(line);
			} catch(IOException e) {
				System.out.println(e.getMessage());
			}

		}

	}	// End of echoclient() function...


	public static void main(String[] args)
	{
		Socket s = null;

		try {
			// Create the socket to communicate with "echo"
			// on the specified host
			s = new Socket(args[0], 8205);

			// Create streams for reading and writing lines
			// of text from and to this socket
			DataInputStream sin =
				new DataInputStream(s.getInputStream());
			DataOutputStream sout =
				new DataOutputStream(s.getOutputStream());

			// Tell the user that we've connected
			System.out.println("Connected to " +
				s.getInetAddress() + ":" + s.getPort());

			// Use the echo feature...
			echoclient(sin, sout);
```

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

                // Always be sure to close the socket...
                finally {
                        try {
                                if (s != null)
                                        s.close();
                        } catch(IOException e) {
                                System.out.println("Closing socket...");
                        }
                }
        }               // End of main()

}               // End of class
```

Να παρατηρήσουμε σε αυτό το σημείο, ότι οι εντολές που σχετίζονται με αρχικοποίηση του socket και των streams αυτού, θα πρέπει να περιέχονται στο σώμα try εντολής κι αυτό γιατί υπάρχει περίπτωση να μην είναι εφικτή η σύνδεση, οπότε θα έχουμε και έγερση μιας εξαίρεσης IOException. Σε μια τέτοια περίπτωση, δεν θα πρέπει να πραγματοποιήσουμε λειτουργίες I/O σε μη συνδεδεμένο socket.
Επίσης, η μέθοδος close() είναι θεμιτό να καλείται σε κάθε περίπτωση, γι' αυτό και την καλούμε μέσα από την finally.

Ο εξυπηρετητής (server)

Μια τυπική TCP εφαρμογή εξυπηρετητή ανοίγει ένα "καλά - γνωστό/διαδεδομένο" port για την λήψη αιτήσεων για σύνδεση και ύστερα δημιουργεί μία διεργασία-παιδί, ή ένα ξεχωριστό νήμα εκτέλεσης για να εκτελέσει την υπηρεσία. Το port που ανοίγει ο server ονομάζεται "καλά-γνωστό", γιατί αυτό χρησιμοποιεί ο οποιοσδήποτε πελάτης για να συνδεθεί στον εξυπηρετητή. Επίσης λέμε ότι ο server "ακούει" το port στο οποίο αρχικοποιεί το socket του, για καινούργιες συνδέσεις, δηλ. καινούργιους πελάτες. Γι' αυτό το λόγο, το socket ονομάζεται "listening socket". Θα πρέπει να δοθεί προσοχή στην επιλογή του αριθμού θύρας της υπηρεσίας, ο οποίος δεν θα πρέπει να είναι ήδη σε χρήση.

Ο εξυπηρετητής, μόλις δεχθεί την σύνδεση καινούργιου πελάτη, γεννά μια καινούργια διεργασία - νήμα, για την εξυπηρέτηση των αιτήσεων του. Με αυτόν τον τρόπο, καθίσταται δυνατή η παράλληλη εξυπηρέτηση παλιών και η αποδοχή νέων πελατών.

Παρακάτω θα αναπτύξουμε τον αντίστοιχο εξυπηρετητή του EchoClient, ο οποίος αποτελείται από δύο κλάσεις : την κλάση EchoServer, για την αρχικοποίηση του "listening-socket" και την κλάση EchoThread, η οποία κληρονομεί την κλάση Thread της Java. Αντικείμενα της τελευταίας δημιουργούνται κάθε φορά που εντοπίζεται από τον EchoServer ένας νέος πελάτης, αναλαμβάνοντας την εξυπηρέτησή του.

Η αλληλουχία βημάτων για το πρόγραμμα του server είναι η εξής :

ServerS*ocket()*	- αρχικοποίηση του listening socket
accept()	- αναμονή και εντοπισμός καινούργιου πελάτη
new Thread	- δημιουργία καινούργιου thread για την εξυπηρέτηση πελάτη
getInputStream() και *getOutputStream()*	- το καινούργιο socket που επιστρέφεται από την accept συνδέεται με τον στανταρ μηχανισμό εισόδου/εξόδου στη Java, δηλ. τα streams
read() και write()	- ανάγνωση και εγγραφή στο - συνδεδεμένο με το socket - stream
……..	- ακολουθούν και άλλες αναγνώσεις/εγγραφές

close() - κλείνει το socket (όχι το listening) και
απελευθερώνεται ο αντίστοιχος πόρος του συστήματος.

Η κλάση ServerSocket περιέχεται στο πακέτο java.net και παρέχει μια ανεξάρτητη συστήματος υλοποίηση σύνδεσης TCP socket, απ' την πλευρά του εξυπηρετητή, με βάση το client/server μοντέλο. Το αντικείμενο αυτής της κλάσης συνδέει το πρόγραμμα του εξυπηρετητή με κάποια θύρα του συστήματος, για να μπορεί αυτός στην συνέχεια να "ακούει" για την σύνδεση νέων πελατών. Στον κατασκευαστή του αντικειμένου προσδιορίζεται ο αριθμός θύρας και σε περίπτωση που δεν χρησιμοποιείται ήδη από άλλη υπηρεσία, δημιουργείται το αντικείμενο. Αν όμως η θύρα είναι κατειλημμένη, τότε εγείρεται εξαίρεση. Ο κώδικας είναι ως εξής :

```
ServerSocket serversoc = new ServerSocket(8205);
```

Αν το αντικείμενό μας κατασκευαστεί επιτυχώς, τότε ο server είναι έτοιμος για την αποδοχή καινούργιου πελάτη :

```
Socket incoming = serversoc.accept();
```

Η μέθοδος accept() μπλοκάρει το νήμα απ' το οποίο έχει κληθεί, μέχρι να παρατηρηθεί δραστηριότητα στο listening-socket, δηλ. μέχρι να συνδεθεί καινούργιος πελάτης. Όταν συμβεί το τελευταίο, τότε η accept() επιστρέφει ένα καινούργιο Socket (όχι listening-socket) με το οποίο αρχίζει πλέον η επικοινωνία. Αυτό συμβαίνει για να μπορούμε να συνεχίσουμε να "ακούμε" απ' το listening socket και άλλους πελάτες, χωρίς να χρειάζεται να αλλάξουμε τον αριθμό της θύρας ("well-known"). Επίσης, κατασκευάζεται ένα νέο αντικείμενο τύπου EchoThread και αρχίζει η εκτέλεσή του :

```
EchoThread et = new EchoThread(incoming);
et.start();
```

Στην κατασκευή του νέου thread περνούμε σαν παράμετρο το socket που επέστρεψε η accept(), για να λάβουμε ύστερα, τα συνδεδεμένα με αυτό, streams εισόδου/εξόδου :

```
DataInputStream in =
                new DataInputStream(s.getInputStream());
PrintStream out =
                new PrintStream(s.getOutputStream());
```

Από εκεί και πέρα ακολουθεί I/O στο socket (μέσω των streams του), σαν να είχαμε I/O σε αρχείο. Στο τέλος, το thread εξυπηρέτησης του πελάτη κλείνει το socket που μας είχε επιστρέψει η accept (μέθοδος close()) και τερματίζει την εκτέλεσή του. Το listening socket παραμένει ανοικτό, απελευθερώνεται όμως πριν τον τερματισμό του ίδιου του EchoServer.

Πάλι σημειώνουμε την χρήση των μπλοκ try, για να αποφύγουμε την περίπτωση της χρησιμοποίησης κάποιου πόρου (στην περίπτωσή μας socket), ο οποίος πιθανόν δεν μας έχει διατεθεί από το σύστημα.

Ακολουθεί το πλήρες πρόγραμμα του εξυπηρετητή της υπηρεσίας "echo", με τις δύο κλάσεις του.

```
// Server program for the "echo" service. An alternative
// to the standard "echo" at port 7. We use TCP port 8205.
// Written using JDK 1.1.5, Petros Zerfos @1998

import java.net.*;
import java.io.*;
import java.lang.*;
import java.util.*;

public class EchoServer
```

```
{
        public static void main(String args[])
        {
                // initialize the network connection
                try {
                        ServerSocket serversoc = new ServerSocket(8205);

                        // Now sit in an infinite loop and wait for
                        // requests...

                        while (true) {
                                // accept the message
                                Socket incoming = serversoc.accept();

                                // spawn a child to serve the request
                                EchoThread et = new EchoThread(incoming);
                                et.start();
                        }
                } catch (IOException e) {
                        System.out.println("Error : " + e.getMessage());
                }

        }               // End of main

}               // End of class EchoServer()


class EchoThread extends Thread
{
        // The socket we are writing to
        Socket s;

        // Our constructor
        EchoThread(Socket s)
        {
                this.s = s;
        }

        // The run method of the thread...
        public void run()
        {
                boolean finished = false;

                try {
                        // Create streams for reading / writing lines of text to the socket
                        DataInputStream in =
                                new DataInputStream(s.getInputStream());
                        PrintStream out =
                                new PrintStream(s.getOutputStream());

                        // Print a message:
                        System.out.println("Client from : " +
                                s.getInetAddress() + " port " + s.getPort());

                        // now get the input from the socket...
                        while(!finished) {
                                String st = in.readLine();
                                // Send the same back to client
                                out.println(st);
                                // Write it to the screen as well
```

```
                                System.out.println(st);
                                // If the input was 'quit' then exit...
                        if (st.equals("quit")) {
                           finished = true;
                           System.out.println("Thread exiting...");
                        }
                                }
                           } catch (IOException e) {
                                System.out.println("Error : " + e.getMessage());
                           }

                           // Always be sure to close the socket...
                           finally {
                                try {
                                        if (s != null)
                                                s.close();
                                } catch(Exception e) {
                                        System.out.println("Error : " +
                                                                        e.getMessage());
                                }
                           }
        }                  // End of run

}                  // End of class EchoThread...
```

11.2.3 Κατασκευαστές και Μέθοδοι της κλάσης Socket

Παραθέτουμε, ενδεικτικά, μερικούς από τις μεθόδους και τους κατασκευαστές της κλάσης Socket.

Κατασκευαστές

Socket(InetAddress, int)
Δημιουργεί ένα stream (TCP) socket και το συνδέει στο port και την IP διεύθυνση που καθορίζονται ως παράμετροι.
Socket(String, int)
Δημιουργεί ένα stream (TCP) socket και το συνδέει στο port του host, του οποίου το όνομα καθορίζεται στις παραμέτρους.

Μέθοδοι

close()
Κλείσιμο του socket (απελευθέρωσή του πίσω στο σύστημα).
getInetAddress()
Επιστρέφει την διεύθυνση του απομακρυσμένου host στην οποία το socket είναι συνδεδεμένο.
getInputStream()
Επιστρέφει ένα stream εισόδου γι' αυτό το socket.
getLocalAddress()
Επιστρέφει την διεύθυνση του host στον οποίο το socket αρχικοποιήθηκε (την τοπική διεύθυνση).
getLocalPort()
Επιστρέφει τον αριθμό θύρας (τοπικό) του host στον οποίο το socket αρχικοποιήθηκε.
getOutputStream()
Επιστρέφει ένα stream εξόδου αυτού του socket
getPort()
Επιστρέφει τον αριθμό θύρας του απομακρυσμένου host στον οποίο το socket έχει

συνδεθεί.

11.2.4 Κατασκευαστές και Μέθοδοι της κλάσης ServerSocket

Παραθέτουμε, ενδεικτικά, μερικούς από τις μεθόδους και τους κατασκευαστές της κλάσης ServerSocket.

Κατασκευαστές

ServerSocket(int)
Δημιουργεί ένα server socket στη θύρα της οποίας ο αριθμός καθορίζεται απ' την παράμετρο. Αν ο αριθμός είναι το 0, τότε αφήνεται στο σύστημα η επιλογή μιας ελεύθερης (μη χρησιμοποιούμενης) θύρας.
ServerSocket(int, int)
Δημιουργεί ένα server socket στην θύρα της οποίας ο αριθμός καθορίζεται απ' την πρώτη παράμετρο, ενώ επιτρέπει τόσες συνδέσεις να περιμένουν στην ουρά του port, μέχρι να γίνουν αποδεκτές, όσες η δεύτερη παράμετρος. Και εδώ, το 0 επιλέγει μια ελεύθερη θύρα.

Μέθοδοι

accept()
Ακούει το server socket για πιθανή σύνδεση που θα γίνει και την αποδέχεται.
close()
Κλείνει το server socket.
getInetAddress()
Επιστρέφει την τοπική διεύθυνση αυτού του server socket.
getLocalPort()
Επιστρέφει το port που αυτό το server socket "ακούει".

11.2.5 Περιγραφή και ορισμός των Datagrams

Σε αντίθεση με την ανάπτυξη εφαρμογών με TCP sockets, όπου ο διαχωρισμός των ρόλων μεταξύ πελάτη και εξυπηρετητή είναι σαφής, στα UDP Datagrams η επικοινωνία που συνήθως αναπτύσσεται είναι "peer-to-peer", δηλ. "ίσος προς ίσον".
Δεν έχουμε δηλ. συχνά πλήρη διάκριση των ρόλων μεταξύ των δύο επικοινωνουσών διεργασιών και τα δύο ,κατ' όνομα, προγράμματα του πελάτη και του εξυπηρετητή λειτουργούν με παρόμοια διαδικασία βημάτων.

Πρέπει, όμως, κατά την σχεδίαση συστημάτων, να έχουμε κατά νου, ότι, ενώ τα TCP sockets διακινούν την πληροφορία αξιόπιστα και η λήψη της από το άλλο άκρο γίνεται με την ίδια σειρά με την οποία απεστάλη (με όποιο πρόσθετο χρονικό, επικοινωνιακό και υπολογιστικό κόστος συνεπάγονται οι επιβεβαιώσεις και αναμεταδόσεις που απαιτούνται για αυτό), στα datagrams, τα πακέτα (packets) που μετακινούνται μέσω του δικτύου δεν έχουν τις προηγούμενες ιδιότητες. Επίσης, ενώ στα TCP sockets η προς αποστολή πληροφορία, δεν περιείχε, αυτή καθ' αυτή, την διεύθυνση και τη θύρα προορισμού, καθώς είχε ήδη προηγηθεί ολόκληρη διαδικασία σύνδεσης και εξασφάλισης συγκεκριμένου καναλιού για την διακίνησή της, στα UDP datagrams, τα πακέτα, εκτός από την χρήσιμη πληροφορία, περιέχουν και την διεύθυνση του host-προορισμού και την θύρα του. Αυτό είναι φυσικό, λόγω της απουσίας συγκεκριμένης διαδρομής από τον έναν υπολογιστή στον άλλο. Ο ορισμός του datagram ακολουθεί :

Ορισμός : Datagram είναι ένα ανεξάρτητο, πλήρως προσδιορίσιμο (δηλ. με διεύθυνση και αριθμό θύρας) μήνυμα, του οποίου η άφιξη στον παραλήπτη, ο χρόνος της άφιξης και το περιεχόμενο δεν είναι εγγυημένα.

Όσο και αν φαίνεται περίεργο, υπάρχουν εφαρμογές που δεν απαιτούν τόσο αυστηρή και τέλεια επικοινωνία, με το συνεπαγόμενο κόστος, όπως αυτή που προσφέρουν τα TCP sockets. Σαν παράδειγμα, μπορούμε να αναφέρουμε το γνωστό πρόγραμμα PING, του οποίου σκοπός είναι ο έλεγχος της ποιότητας του επικοινωνιακού διαύλου μεταξύ δύο υπολογιστών. Στην πραγματικότητα, το PING προσπαθεί να ελέγξει και να μετρήσει τον αριθμό των πακέτων που χάθηκαν ή απορρίφθηκαν, για τον προσδιορισμό της ποιότητας της σύνδεσης. Αυτό δεν θα ήταν δυνατό με ένα πρωτόκολλο σαν το TCP, που διορθώνει αυτά ακριβώς τα λάθη.

11.2.6 Ανάπτυξη δικτυακής εφαρμογής με UDP Datagrams

Όπως αναφέραμε και προηγουμένως, οι έννοιες του πελάτη και του εξυπηρετητή δεν είναι και τόσο διακριτές, όταν έχουμε υλοποίηση με UDP datagrams. Επίσης, το βάρος τώρα δεν δίνεται τόσο στην σύνδεση (δεν υπάρχει άλλωστε με την τυπική έννοια), αλλά στη σωστή δημιουργία του πακέτου που πρόκειται να σταλεί, έτσι ώστε να περιέχει όλες τις απαραίτητες πληροφορίες, μαζί με τα προς αποστολή δεδομένα.
Ως παράδειγμα εφαρμογής δικτύου με UDP πρωτόκολλο, παραθέτουμε δύο προγράμματα : το πρώτο, ο TimeClient, είναι ένα πρόγραμμα με το οποίο μπορεί κανείς να κάνει ερώτηση σε απομακρυσμένο host, ζητώντας του μάθει την εκεί ώρα. Αντίστοιχα ο TimeServer παρέχει αυτήν την υπηρεσία, καθώς εκτελείται συνέχεια στον απομακρυσμένο host. Για τον αριθμό θύρας της υπηρεσίας διαλέξαμε τον 8505 κατά τυχαίο τρόπο, αλλά μπορεί να χρησιμοποιηθεί και οποιοσδήποτε άλλος διαθέσιμος.

Θα χρησιμοποιήσουμε τα αντικείμενα δύο κλάσεων : την κλάση DatagramSocket, που υλοποιεί την επικοινωνία με UDP datagrams και την κλάση DatagramPacket, που αποτελεί το καλούπι με το οποίο θα φτιάχνουμε τα προς αποστολή πακέτα.

Ο πελάτης

Πρώτα πρέπει να κατασκευάσουμε το αντικείμενο της διεύθυνσης του host που τρέχει ο TimeServer. Αυτή παρέχεται από τον χρήστη με την εντολή εκτέλεσης του προγράμματος από την γραμμή εντολών (π.χ. #java TimeClient 127.0.0.1 – Αξίζει να αναφέρουμε εδώ οτι η διεύθυνση 127.0.0.1 είναι η default διεύθυνση του τοπικού μηχανήματος την οποία μπορούμε και να πάρουμε μέσω της μεταβλητής localhost). Το αντικείμενο της διεύθυνσης ανήκει στην κλάση InetAddress, της οποίας χρησιμοποιούμε την μέθοδο getByName(String), για να το μετατρέψουμε από την αλφαριθμητική μορφή που το λαμβάνουμε στον τύπο InetAddress :

```
InetAddress hostAddress = InetAddress.getByName(args[0]);
```

Κατόπιν δημιουργούμε το datagram socket, μέσω του οποίου θα στείλουμε το πακέτο με την αίτηση και θα λάβουμε αυτό με την απάντηση. Ο παρακάτω κώδικας εκτελεί ακριβώς αυτό :

```
DatagramSocket serversoc = new DatagramSocket();
```

Ο κατασκευαστής που χρησιμοποιήσαμε, διαλέγει από μόνος του κάποιο ελεύθερο port στο μηχάνημα που “τρέχει” ο πελάτης, για να το συνδέσει με το datagram socket. Να σημειώσουμε εδώ, ότι δεν χρειάζεται να συνδέσουμε το datagram socket με streams εισόδου/εξόδου, αφού δεν έχουμε I/O σε κάποιο κανάλι, αλλά αποστολή και λήψη μεμονωμένων πακέτων.

Ακολουθεί κατασκευή του πακέτου της αίτησης. Ανήκει στην κλάση DatagramPacket, ο κατασκευαστής της οποίας δέχεται 4 ορίσματα : έναν buffer, ο οποίος αποτελεί τα δεδομένα σε μορφή πίνακα από bytes (δεν είναι String ή array από χαρακτήρες), το μέγεθος του προηγούμενου πίνακα, την InetAddress διεύθυνση προορισμού, καθώς και τον αριθμό θύρας του προορισμού. Στην περίπτωση της αίτησης στον TimeServer, ο buffer δεν περιέχει δεδομένα, γιατί δεν θέλουμε να στείλουμε κανενός είδους χρήσιμη πληροφορία.

```
DatagramPacket packet;
```

```
byte[] message = new byte[256];
packet = new DatagramPacket(message, 256, hostaddress, port);
```

Η αποστολή γίνεται, πολύ απλά, καλώντας την μέθοδο send(DatagramPacket) του datagram socket :

```
serversoc.send(packet);
```

Στη συνέχεια έχουμε αναμονή για λήψη της απάντησης με την μέθοδο receive(DatagramPacket) του datagram socket.

```
serversoc.receive(packet);
```

Η receive() “μπλοκάρει” το νήμα απ’ το οποίο έχει κληθεί, μέχρι να λάβει κάποιο datagram πακέτο στο port που έχει ανοίξει το datagram socket. Χρειάζεται όμως προσοχή, καθώς το UDP πρωτόκολλο δεν είναι αξιόπιστο και το πακέτο με την απάντηση μπορεί να χαθεί. Γι’ αυτό το λόγο συνίσταται η χρησιμοποίηση ενός χρονομέτρου, το οποίο θα ξεμπλοκάρει το νήμα, για να τερματιστεί ομαλά η εφαρμογή.

Τέλος, με την μέθοδο getData() της κλάσης DatagramPacket, λαμβάνουμε τον πίνακα από bytes με την χρήσιμη πληροφορία, τον οποίο μετατρέπουμε σε String για την εκτύπωση. Οι μέθοδοι getInetAddress() και getPort() επιστρέφουν την διεύθυνση και τον αριθμό θύρας αντίστοιχα, του αποστολέα του πακέτου. Πριν το πρόγραμμα τερματιστεί, δεν ξεχνάμε να απελευθερώσουμε τους πόρους που είχαμε δεσμεύσει, δηλ. το DatagramSocket().

Ακολουθεί το πλήρες πρόγραμμα του TimeClient :

```
// the TimeClient program in Java,
// using UDP Datagrams... We will use UDP port 8505
// Petros Zerfos @1998

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

public class TimeClient {
      public static void timeclient(DatagramSocket serversoc,
                  InetAddress hostaddress, int port) {
            // Declare a datagram packet...
            DatagramPacket packet;

            // Declare a byte array to send as a request and
            // receive the answer
            byte[] message = new byte[256];

            try {
              // construct the packet with all the information
                              // needed
              packet = new DatagramPacket(message, 256, hostaddress, port);

              System.out.println("Sending the request for the time...");
              // since the socket is already open, send the
                              // request
              serversoc.send(packet);

              // Now, get the reply
              System.out.println("Waiting for reply...");
              serversoc.receive(packet);
              String mesg = new String(packet.getData(), 0);
```

```
            // Print the server's time
            System.out.println("Time at server's location  " +
                packet.getAddress() + ":" + packet.getPort() +  is " + mesg );
        } catch(Exception exc) {
            System.out.println("Error : " + exc.toString());
        }
    }           // End of timeclient()

    public static void main(String args[]) {
        // We use the command line to get the address of the
        // time server
        InetAddress hostAddress;
        int portnum;

        // Declare a datagram socket
        DatagramSocket serversoc;

        // initialize the network
        try {
            // Get network info
            hostAddress = InetAddress.getByName(args[0]);
            portnum = 8505;

            serversoc = new DatagramSocket();

            timeclient(serversoc, hostAddress, portnum);

            // Don't forget to release the resource...
            // by closing the datagram socket.
            if (serversoc != null) serversoc.close();

        } catch (UnknownHostException uhe) {
            System.out.println("Unknown host : " +
                                uhe.toString());
        } catch (SocketException exc) {
            System.out.println("Error : " + exc);
        }

    }                   // End of main()

}                   // End of class
```

Να σημειώσουμε και εδώ την χρήση μπλοκ try, για την αντιμετώπιση πιθανών προβλημάτων στην εκτέλεση κατασκευαστών και των μεθόδων.

Ο εξυπηρετητής

Ο server της υπηρεσίας ώρας δεν διαφέρει και πολύ από τον client. Η μεθοδολογία ανάπτυξής του μοιάζει πάρα πολύ με αυτή του πελάτη και μάλλον κουραστική θα ήταν μια εκτενέστερη ανάλυση. Εκείνο που αξίζει να πούμε, είναι ότι, εξαιτίας του ότι η παρεχόμενη υπηρεσία απαιτεί μικρής διάρκειας εξυπηρέτηση, δεν είναι απαραίτητο ο εξυπηρετητής να δημιουργεί για κάθε πελάτη και ένα καινούργιο νήμα, ώστε να μπορεί να δεχθεί και νέες συνδέσεις. Όλη η χρήσιμη πληροφορία της απάντησης μπορεί να χωρέσει μέσα σε ένα datagram packet, το οποίο έχει μέγιστο μέγεθος 8KBytes. Στα UDP datagrams, το βάρος της σχεδίασης μετατοπίζεται, από την εξυπηρέτηση του πελάτη, στην αυτόνομη εξυπηρέτηση των πακέτων.

Μπορούμε, επίσης, να παρατηρήσουμε τον τρόπο με τον οποίο ο εξυπηρετητής “ανοίγει” το datagram socket, για την λήψη των πακέτων : ο κατασκευαστής του αντικειμένου DatagramSocket δέχεται πλέον το port (δηλ. το 8505), στο οποίο “ακούει” το socket, για νέες συνδέσεις. Ακολουθεί ο πλήρης κώδικας για το πρόγραμμα του εξυπηρετητή :

```
// The TimeServer for the time service...
// Using UDP Datagrams and port number 8505
// Petros Zerfos @1998

import java.io.*;
import java.net.*;
import java.util.*;     // For the date functions...

public class TimeServer {
      public static void main(String args[]) {
            // Our datagram socket for I/O
            DatagramSocket s = null;

            // Our packet for getting the request and sending the
                        // reply
            DatagramPacket packet = null;

            // Create a receive buffer
            byte[] buf = new byte[256];

            // Create the packet to receive the request
            packet = new DatagramPacket(buf, buf.length);

            // now create a socket to listen in
            try {
                  s = new DatagramSocket(8505);
            } catch(Exception e) {
                  System.out.println("Error : " + e.toString());
            }

            // Now sit in an infinite loop and reply to the
                        // queries...
            while (true) {
                  // sit around and wait for a request
                  try {
                        s.receive(packet);
                  } catch (Exception e) {
                        System.out.println("Error : " + e.toString());
                  }

                  // Get the client's info
                  InetAddress cl = packet.getAddress();
                  int port = packet.getPort();
                  System.out.println("Client from " +
                        cl.getHostAddress() + ":" + port +
                        " requested the time.");

                  // Construct the response and sent it back...
                  String localtime = new Date().toString();
                  // Convert the localtime string to an array
                                    // of bytes
                  localtime.getBytes(0, localtime.length(), buf, 0);
                  // Construct the packet
                  try {
```

```
                    packet = new DatagramPacket(buf, buf.length, cl, port);

                    // Send it!
                    s.send(packet);
                } catch (Exception e) {
                    System.out.println("Error : " + e.toString());
                } //End of while

                System.out.println("Client served...");
            }
        }           // End of main()

}               // End of TimeServer class.
```

11.2.7 Κατασκευαστές και Μέθοδοι της κλάσης DatagramSocket

Κατασκευαστές

DatagramSocket()
Κατασκευάζει ένα datagram socket και το συνδέει σε κάποιο διαθέσιμο port, στο μηχάνημα του τοπικού host.
DatagramSocket(int)
Κατασκευάζει ένα datagram socket και το συνδέει στη θύρα που προσδιορίζεται στην παράμετρο, στον τοπικό host.
DatagramSocket(int, InetAddress)
Κατασκευάζει ένα datagram socket και το συνδέει στην τοπική διεύθυνση και θύρα.

Και οι τρεις κατασκευαστές εγείρουν μια SocketException εξαίρεση, σε περίπτωση αποτυχίας στη δημιουργία του αντικειμένου.

Μέθοδοι

close()
Κλείνει το datagram socket.
getLocalAddress()
Gets the local address to which the socket is bound.
Επιστρέφει την τοπική διεύθυνση, στην οποία το socket έχει συνδεθεί.
getLocalPort()
Επιστρέφει τον αριθμό θύρας στο τοπικό μηχάνημα, στον οποίο το socket έχει συνδεθεί.
receive(DatagramPacket)
Δέχεται ένα datagram πακέτο από αυτό το socket.
send(DatagramPacket)
Στέλνει ένα datagram πακέτο από αυτό το socket.

11.2.8 Κατασκευστές και Μέθοδοι της κλάσης DatagramPacket

Κατασκευαστές

DatagramPacket(byte[], int)
Κατασκευάζει ένα DatagramPacket, για λήψη πακέτων με μήκος αυτό που προσδιορίζεται από την δεύτερη παράμετρο. Η πρώτη παράμετρος είναι ο buffer στον οποίο θα τοποθετηθούν τα δεδομένα.
DatagramPacket(byte[], int, InetAddress, int)
Κατασκευάζει ένα datagram packet για αποστολή πακέτων. Η πρώτη παράμετρος είναι το δεδομένα σε μορφή πίνακα από bytes, η δεύτερη το μέγεθος του πίνακα, ενώ η τρίτη και τέταρτη παράμετρος είναι η διεύθυνση και ο αριθμός θύρας

προορισμού αντίστοιχα.

Μέθοδοι

getAddress()
Επιστρέφει την IP διεύθυνση του υπολογιστή, στον οποίο το datagram θα σταλθεί, ή απ' τον οποίο έχει παραληφθεί.
getData()
Επιστρέφει τα δεδομένα του πακέτου που έλαβε, ή που θα στείλει.
getLength()
Επιστρέφει το μήκος των δεδομένων του πακέτου που έλαβε, ή που θα στείλει.
getPort()
Επιστρέφει τον αριθμό θύρας του απομακρυσμένου host, απ' τον οποίο το πακέτο ελήφθει, ή τον αριθμό θύρας του host, στον οποίο το πακέτο θα σταλεί.