# ΔΙΑΔΙΚΤΥΑ :

## Υπηρεσίες, Τεχνολογία, Εφαρμογές

*Τμήμα Πληροφορικής Α.Π.Θ.*

## Υποστηρικτικό υλικό – Σημειώσεις

**Διδάσκοντες : Α. Βακάλη – Γ. Παπαδημητρίου**

*Εαρινό εξάμηνο 2000 – 2001*

# ΜΕΡΟΣ II

# J A V A

## ΔΙΑΔΙΚΤΥΑ : Υπηρεσίες, Τεχνολογία, Εφαρμογές

## Προγραμματισμός στο Διαδίκτυο & JAVA

*Τμήμα Πληροφορικής Α.Π.Θ.*

**ΣΧΗΜΑΤΑ**

# 1. ΕΙΣΑΓΩΓΗ

## 1.1. Τι είναι η JAVA ?

Η Java είναι μία νέα γλώσσα αντικειμενοστραφούς προγραμματισμού που αναπτύχθηκε από την εταιρία Sun Microsystems και μοντελοποιήθηκε με βάση τη γλώσσα C++. Η Java σχεδιάστηκε έτσι ώστε να είναι απλή, σύντομη και προσαρμόσιμη σε διάφορες πλατφόρμες και Λειτουργικά Συστήματα. Το χαρακτηριστικό της Java είναι ότι υποστηρίζει την ανάπτυξη μικρο-εφαρμογών (apllets) οι οποίες μπορούν να εκτελεστούν από οποιοδήποτε από τα γνωστά προγράμματα πλοήγησης (browsers) του Παγκόσμιου Ιστού (World Wide Web) του δια-δικτύου Internet.

Η διαδικασία για τη δημιουργία μίας μικρο-εφαρμογής στην Java περιλαμβάνει :

- σύνταξη του προγράμματος σε Java
- μετάφραση του προγράμματος με χρήση μεταφραστή της Java
- αναφορά της εφαρμογής στην HTML σελίδα

Η html σελίδα εγκαθίστανται σε κάποιον Web κόμβο. Οταν κάποιος χρήστης αναφέρεται στη συγκεκριμένη html σελίδα μέσω κάποιου προγράμματος πλοήγησης, βλέπει την εκτέλεση της αναφερόμενης Java εφαρμογής. Αυτό γίνεται διότι το πρόγραμμα πλοήγησης "κατεβάζει" (downloads) την εφαρμογή στο τοπικό σύστημα και την εκτελεί τοπικά ώστε ο χρήστης να δει το αποτέλεσμα της εκτέλεσης της.

Είναι σημαντικό να τονιστεί ότι η Java δε σχεδιάστηκε απλώς για τη σύνταξη Web εφαρμογών, αλλά για να αποτελέσει μία ισχυρή πλήρη γλώσσα αντικειμενοστραφούς προγραμματισμού.

## 1.2. Ιστορικό της JAVA

Οπως αναφέρθηκε, η Java ξεκίνησε από την εταιρία Sun το 1991 στα πλαίσια ενός ερευνητικού προγράμματος που είχε θέμα την ανάπτυξη λογισμικού για τις κοινές ηλεκτρονικές συσκευές (τηλεοράσεις, video, τοστιέρες κλπ). Ο αρχικός στόχος της Java ήταν να δημιουργηθεί μία απλή, γρήγορη κι ευφυής γλώσσα που να μπορεί να προσαρμοστεί σε ένα ευρύ φάσμα ηλεκτρονικών συσκευών. Αυτός ο αρχικός στόχος οδήγησε την Java στη δυναμική της δυνατότητα να κατανέμει εκτελέσιμα προγράμματα μέσω του Web σε ένα σύνολο διαφορετικών υπολογιστικών συστημάτων.

Η Java χρησιμοποιήθηκε από την Sun σε διάφορα έργα της εσωτερικά χωρίς να αποκτήσει ιδιαίτερο εμπορικό ενδιαφέρον ως το 1994 οπότε και αναπτύχθηκε σε διάστημα λίγων μηνών το πρόγραμμα πλοήγησης HotJava που έδωσε τη δυνατότητα της εκτέλεσης και μεταφοράς των Java εφαρμογών.

Παράλληλα αναπτύχθηκε υποστηρικτικό υλικό που αποτέλεσε το πακέτο ανάπτυξης Java εφαρμογών (Java Developer's Kit - JDK), το οποίο περιλαμβάνει ένα σύνολο εργαλείων για την ανάπτυξη των Java εφαρμογών. Σήμερα κυκλοφορεί η έκδοση JDK 1.1 η οποία περιλαμβάνει και το πρόγραμμα επισκόπησης των μικρο-εφαρμογών (appletviewer) που δίνει τη δυνατότητα ελέγχου των Java εφαρμογών κατά τη σύνταξη τους.

Ποιό είναι το μέλλον της Java ? Ηδη άλλες εταιρίες (εκτός της Sun) υποστηρίζουν την Java στα προγράμματα πλοήγησης τους (Netscape, Microsoft κλπ) και συνεχίζεται η κυκλοφορία νέων εκδόσεων της Java με αυξημένες δυνατότητες και εργαλεία προγραμματισμού. Η γλώσσα Java περιλαμβάνεται ήδη στα Προγράμματα Σπουδών πολλών Πανεπιστημίων στις Ηνωμένες Πολιτείες ως βασική γλώσσα προγραμματισμού (μάθημα CS1) και χρησιμοποιείται για την υλοποίηση εργασιών και ερευνητικών έργων. Επίσης, έχει ήδη επεκταθεί και σε πολλές εφαρμογές απαιτήσεων (π.χ. στην NASA για επεξεργασία των δεδομένων και στοιχείων που λαμβάνονται από δορυφόρους) λόγω της δυναμικότητας της και της δυνατότητας της να συνδυάζει τον κλασσικό προγραμματισμό με τις νεές γλώσσες υπερκειμένου που αναπτύχθηκαν μέσω του Internet.

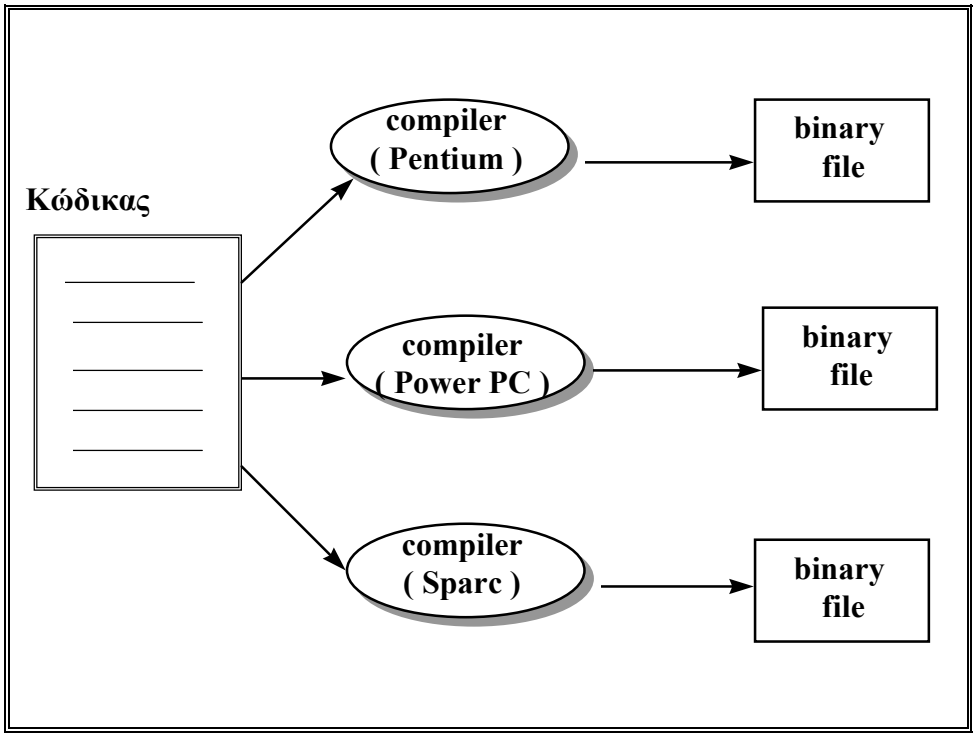
## 1.3. Πλεονεκτήματα της JAVA

Η Java παρουσιάζει τα παρακάτω βασικά πλεονεκτήματα :

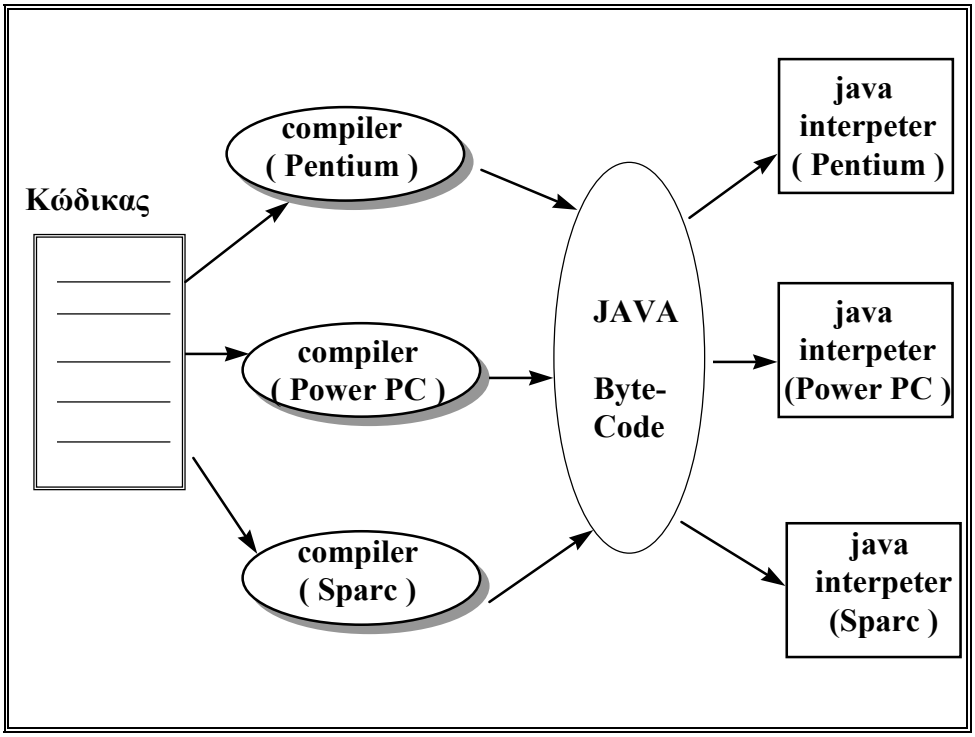
- Ανεξαρτησία από την Πλατφόρμα του Υπολογιστικού Συστήματος. Η ανεξαρτησία αυτή εκφράζεται με τη δυνατότητα μετακίνησης ενός προγράμματος εύκολα από ένα υπολογιστικό σύστημα σε κάποιο άλλο. Οι βασικές κλάσεις της Java δίνουν τη δυνατότητα να γραφεί κώδικας που μπορεί να μεταφερθεί από μία υπολογιστική πλατφόρμα σε μία άλλη χωρίς να απαιτείται επανασύνταξη του προγράμματος. Η δια-πλατφορμική ανεξαρτησία δεν αφορά μόνο στο επίπεδο κώδικα της Java. Τα δυαδικά αρχεία (binary files) της Java είναι επίσης δια-πλατφορμικά και μπορούν να εκτελούνται σε πολλαπλές πλατφόρμες χωρίς να χρειάζεται επανα-μετάφραση του κώδικα. Αυτό γίνεται κατορθωτό λόγω του ότι τα δυαδικά αρχεία αποθηκεύονται σε μία μορφή κωδικολέξης (bytecode).

- Τα bytecodes είναι ένα σύνολο από εντολές που μοιάζουν με κώδικα μηχανής αλλά δεν αφορούν σε συγκεκριμένο τύπο επεξεργαστή. Οταν μεταφράζεται ένα πρόγραμμα σε μία συμβατική γλώσσα προγραμματισμού (π.χ. στη C) ο μεταφραστής μετατρέπει τον κώδικα σε γλώσσα μηχανής ή σε εντολές του επεξεργαστή. Αυτές οι εντολές είναι προσαρμοσμένες στον συγκεκριμένο επεξεργαστή του υπολογιστή στον οποίο μεταφράζεται το πρόγραμμα. Αν χρειάζεται να χρησιμοποιηθεί το ίδιο πρόγραμμα σε κάποιο άλλο μηχάνημα θα πρέπει να μεταφραστεί ξανά με χρήση του τοπικού μεταφραστή ώστε να προκύψει νέο εκτελέσιμο πρόγραμμα για τη συγκεκριμένη μηχανή. (Σχήμα 1.1)

Η Java ενεργεί με διαφορετικό τρόπο. Το περιβάλλον ανάπτυξης της Java περιλαμβάνει δύο μέρη : τον Java μεταφραστή(compiler) και τον Java μεταγλωτιστή(interpreter). Ο Java μεταφραστής παίρνει το Java πρόγραμμα και δημιουργεί τα αρχεία που περιέχουν bytecodes. Το επιπρόσθετο στάδιο δημιουργίας των bytecodes ανεξαρτητοποιεί το εκτελέσιμο πρόγραμμα κι έτσι τα Java προγράμματα εκτελούνται από οποιοδήποτε μηχάνημα διαθέτει το Java μεταγλωτιστή (Σχήμα 1.2).




**Σχήμα 1-1*: Μετάφραση κώδικα στο συμβατικό προγραμματισμό***




*Σχήμα 1.2*

**Σχήμα 1-2*: Μετάφραση κώδικα στην Java***

Με τον ίδιο τρόπο που τα html αρχεία είναι αναγνώσιμα από οποιοδήποτε πρόγραμμα πλοήγησης του Web, έτσι και οι εφαρμογές της Java εκτελούνται από οποιοδήποτε πρόγραμμα πλοήγησης που υποστηρίζει την Java.

Το μόνο μειονέκτημα της χρήσης των bytecodes είναι η ταχύτητα εκτέλεσης. Τα συμβατικά προγράμματα που μεταφράζονται για κάθε μηχανή εκτελούνται γρηγορότερα λόγω του ότι τα Java bytecodes πρέπει να περάσουν από τον τοπικό μεταγλωτιστή. Η ταχύτητα εκτέλεσης δεν είναι το ζητούμενο σε κάθε περίπτωση αλλά ακόμα και για τις περιπτώσεις που απαιτείται η γρηγορότερη δυνατή εκτέλεση υπάρχει η δυνατότητα να συνδεθούν κομμάτια κώδικα συμβατικής γλώσσας προγραμματισμού με τον Java κώδικα και με ειδικά εργαλεία να μετατραπούν τα bytecodes σε απλό κώδικα.

- Η Java είναι γλώσσα αντικειμενοστραφούς προγραμματισμού. Οι περισσότερες αντικειμενοστραφείς ιδιότητες της Java κληρονομήθηκαν από την C++ αλλά έχουν συμπεριληφθεί και επιπλέον χαρακτηριστικά από άλλες γλώσσες προγραμματισμού. Οπως και στις περισσότερες γλώσσες αντικειμενοστραφούς προγραμματισμού, η Java περιλαμβάνει ένα σύνολο από βασικές βιβλιοθήκες κλάσεων (class libraries) που υποστηρίζουν τους βασικούς τύπους δεδομένων, τις δυνατότητες εισόδου εξόδου του συστήματος καθώς και άλλες συναρτήσεις εργαλείων. Αυτές οι βασικές κλάσεις αποτελούν μέρος του βασικού "πακέτου" ανάπτυξης (JDK) της

Java. Το JDK περιλαμβάνει ακόμα κλάσεις που υποστηρίζουν τη δικτύωση, τα βασικά πρωτόκολλα του Internet καθώς και συναρτήσεις για υποστήριξη των χρηστών. Οι κλάσεις αυτές έχουν συνταχθεί επίσης σε Java κι επομένως, είναι μεταφερόμενες μεταξύ των διαφορετικών υπολογιστικών συστημάτων.

- Η Java είναι εύκολη στην εκμάθηση διότι σχεδιάστηκε για να είναι μικρή κι απλή στη σύνταξη, τη μετάφραση, τη μεταφορά της από μηχάνημα σε μηχάνημα. Παρά την απλότητα της είναι ιδιαίτερα δυναμική κι ευέλικτη γλώσσα.

## 1.4. Το πρώτο πρόγραμμα σε JAVA

Είναι προφανές ότι για να εκτελεστούν τα Java προγράμματα χρειάζεται να υπάρχει το κατάλληλο Java περιβάλλον ανάπτυξης εφαρμογών. Οπως ήδη αναφέρθηκε υπάρχει διαθέσιμο το Java Development Kit (JDK) που παρέχει τα απαραίτητα εργαλεία για τη σύνταξη κι εκτέλεση των Java προγραμμάτων. Το JDK είναι διαθέσιμο από διάφορους κόμβους στο Internet και βέβαια ο πλέον έγκυρος κόμβος είναι της εταιρίας Sun (*http://java.sun.com*).

Τα προγράμματα της Java διακρίνονται σε : *μικρο-εφαρμογές(applets)* και *εφαρμογές (applications*). Οι μικροεφαρμογές είναι Java προγράμματα που φορτώνονται μέσω του Παγκόσμιου Ιστού (Web) και εκτελούνται από ένα Web πρόγραμμα πλοήγησης (π.χ. Netscape) στη μηχανή του χρήστη. Οι εφαρμογές είναι Java προγράμματα που δεν απαιτούν κάποιο πρόγραμμα πλοήγησης για να εκτελεστούν αλλά αφορούν στην ανάπτυξη προγραμμάτων για την επίλυση ποικίλων προβλημάτων. Στη συνέχεια παρατίθεται μία μικρή Java εφαρμογή, που μπορεί να αποτελέσει και μικρο-εφαρμογή εάν δηλωθεί σε ένα html αρχείο.

- Το πρόγραμμα Γειά σου, κόσμε

Για τη σύνταξη οποιοδήποτε προγράμματος στην Java χρειάζεται ένας απλός διορθωτής κειμένου (editor) που να δίνει τη δυνατότητα δημιουργίας ASCII αρχείων. Στον editor λοιπόν καταγράφεται το παρακάτω πρόγραμμα :

```
class HelloWorld {
    Public static void main (String args[]) {
        System.out.println("Γεια σου, κόσμε !");
    }
}
```

Το πρόγραμμα αυτό περιλαμβάνει δύο μέρη : τον ορισμό της κλάσης HelloWorld και το κυρίως σώμα του προγράμματος που περιέχεται σε μία μέθοδο που ονομάζεται main.

Αφού γραφεί το πρόγραμμα χρειάζεται να σωθεί σε ένα αρχείο το οποίο πρέπει να έχει το όνομα της κλάσης του προγράμματος και κατάληξη .java δηλαδή θα ονομάζεται HelloWorld.java. Το πρόγραμμα αυτό χρειάζεται να

μεταφραστεί με χρήση του μεταφραστή που περιέχεται στο πακέτο JDK και ονομάζεται javac. Αρα με την εντολή

javac HelloWorld.java

και με την προϋπόθεση ότι δεν υπάρχουν λάθη στο πρόγραμμα δημιουργείται ένα αρχείο με όνομα HelloWorld.class στον κατάλογο όπου βρίσκεται το πρόγραμμα με τον πηγαίο κώδικα. Αυτό το αρχείο περιλαμβάνει τα bytecodes της Java που περιγράφηκαν προηγουμένως. Στη συνέχεια το πρόγραμμα αυτό μπορεί να εκτελεστεί με χρήση του μεταγλωτιστή της Java .Ο μεταγλωτιστής ονομάζεται java και με την εντολή

java HelloWorld

εκτελείται το αρχείο HelloWorld.class και στην οθόνη εμφανίζεται η συμβολοσειρά "Γειά σου, κόσμε !"

Χρειάζεται να τονιστεί ότι ο μεταφραστής(compiler) και ο μεταγλωτιστής (interpreter) στην Java αποτελούν δύο διαφορετικά εργαλεία. Ο μεταφραστής (javac) παίρνει τα .java αρχεία με τον πηγαίο κώδικα και δημιουργεί τα .class αρχεία. Στη συνέχεια, ο μεταγλωτιστής(java) εκτελεί τα .class αρχεία και παράγει τα αποτελέσματα των προγραμμάτων.

## 1.5. Η πρώτη JAVA μικρο-εφαρμογή

Οπως περιγράφηκε, η δημιουργία και ο τρόπος χρήσης των εφαρμογών και των μικρο-εφαρμογών διαφέρει στην Java. Η πρόσβαση στις μικρο-εφαρμογές μέσω του Web πορϋποθέτει κάποιες επιπλέον απαιτήσεις και οδηγεί πολλές φορές σε δυσκολότερη ανάπτυξη μικρο-εφαρμογών. Η διαδικασία ανάπτυξης των μικρο-εφαρμογών περιλαμβάνει τα παρακάτω στάδια :

1. σύνταξη του προγράμματος, δημιουργία .java αρχείου
2. μετάφρασή του με τον javac, δημιουργία .class αρχείου
3. Αναφορά της εφαρμογής σε όλα τα html αρχεία των Web σελίδων μέσω των οποίων θα εκτελείται από κάθε αναγνώστη στο τοπικό του μηχάνημα.

Αντίστοιχα, για τη δημιουργία μικρο-εφαρμογής για το παράδειγμα "Γειά σου, κόσμε !" θα είχαμε :

1. Δημιουργία του αρχείου HelloWorld Applet java με περιεχόμενο :

```
import java.awt.Graphics;
public class HelloWorldApplet  extends java.applet.Applet {
    Public void paint (Graphics g) {
        g.drawString ("Γειά σου, κόσμε !", 5, 25);
    }
}
```

Η import εντολή (ανάλογη της #include της C) επιτρέπει στην μικρο-εφαρμογή την πρόσβαση στις κλάσεις του JDK πακέτου που αφορούν στη δημιουργία μικρο-εφαρμογών και στη χρήση γραφικών στην οθόνη.

2. μετάφραση με την εντολή : javac HelloWorld Applet.java

η οποία δημιουργεί το αρχείο HelloWorldApplet.class

3. Δήλωση της εφαρμογής στη Web σελίδα που την αφορά. Ακολουθεί παράδειγμα μίας τέτοιας html σελίδας :

```
<HTML>
<HEAD><TITLE> Hello to the World </ TITLE></HEAD>
<BODY>
<p> Η Java εφαρμογή μας παρουσιάζει το μήνυμα :
<APPLET CODE="HelloWorldApplet.class" WIDTH=150 HEIGHT=25>
</ APPLET> </BODY></HTML>
```

Για να εκτελεστεί η μικρο-εφαρμογή και να δούμε τα αποτελέσματα χρειάζεται το κατάλληλο πρόγραμμα πλοήγησης που να υποστηρίζει την Java. Επίσης υπάρχει η δυνατότητα να δούμε τα αποτελέσματα μίας μικρο-εφαρμογής με χρήση του εργαλείου appletviewer που περιέχεται στο πακέτο JDK. Η εντολή για να χρησιμοποιήσουμε αυτό το εργαλείο είναι :

appletviewer HelloWorldApplet .html

# 2. ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΑΦΗΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ & JAVA

## 2.1. Αντικείμενα και Κλάσεις

Στον αντικειμενοστραφή προγραμματισμό το πρόγραμμα αποτελείται από ένα σύνολο από επιμέρους ανεξάρτητα "αντικείμενα" καθένα από τα οποία έχει ένα συγκεκριμένο ρόλο στο πρόγραμμα. Επιπλέον, τα αντικείμενα μπορούν να επικοινωνούν μεταξύ τους με μία σειρά από προκαθορισμένους τρόπους. Ο αντικειμενοστραφής προγραμματισμός διευρύνει τις δυνατότητες υποστήριξης μεθόδων και χαρακτηριστικών που καθιστούν εύκολη κι ευέλικτη τη δημιουργία και χρήση αντικειμένων. Το πλέον ιδιαίτερο χαρακτηριστικό είναι η έννοια της κλάσης.

Η κλάση είναι ένα πρότυπο δήλωσης των χαρακτηριστικών που μπορεί να αφορούν πολλαπλά αντικείμενα. Οι κλάσεις ενσωματώνουν όλα τα χαρακτηριστικά ενός συγκεκριμένου συνόλου αντικειμένων. Στον αντικειμενοστραφή προγραμματισμό δεν ορίζουμε αντικείμενα αλλά κλάσεις αντικειμένων. Η στιγμή ύπαρξης μίας κλάσης είναι το πραγματικό αντικείμενο και αποτελεί την συγκροτημένη αναπαράσταση μίας κλάσης. Η προγραμματιστική διαδικασία περιλαμβάνει αρχικά τη δήλωση της κλάσης και στη συνέχεια τη δημιουργία στιγμών αναφοράς προς την κλάση. Κάθε στιγμή αναφοράς της κλάσης υϊοθετεί όλα τα βασικά χαρακτηριστικά της κλάσης και επιπλέον κάθε στιγμή αναφοράς μπορεί να αποκτήσει διαφορετική συμπεριφορά, τρόπο λειτουργίας και παρουσίασης. Σε αναλογία με τη C, η κλάση είναι ένα είδος δημιουργίας ενός συμπαγούς τύπου δεδομένων με χρήση της *struct* και *typedef,* με τη διαφορά ότι οι κλάσεις παρέχουν περισσότερες δυνατότητες από την απλή συλλογή δεδομένων.

Ο κώδικας των Java προγραμμάτων περιλαμβάνει το σχεδιασμό και την κατασκευή του κατάλληλου συνόλου κλάσεων. Ενα σύνολο κλάσεων συγκροτεί τη βιβλιοθήκη των κλάσεων. Το περιβάλλον της Java παρέχει και βιβλιοθήκη κλάσεων που περιλαμβάνει ένα σύνολο διεργασιών που είναι απαραίτητες τόσο για βασικές προγραμματιστικές ανάγκες όσο και για την υποστήριξη γραφικών και δικτυακών εφαρμογών..

## 2.2. Καταστάσεις και Χαρακτηριστικά

Κάθε κλάση στην Java αποτελείται από δύο βασικά στοιχεία : τα χαρακτηριστικά και την κατάσταση στην οποία βρίσκεται. Τα χαρακτηριστικά περιλαμβάνουν τις ιδιότητες που διαφοροποιούν το ένα αντικείμενο από το άλλο. Εστω ότι χρειάζεται να δημιουργηθεί μία κλάση που λέγεται *computer* και περιλαμβάνει τα παρακάτω χαρακτηριστικά :

*Model* με ενδεικτικές τιμές τιμές : SparcStation1, Pentium Pro 2, DECStation

*Company* με ενδεικτικές τιμές τιμές : IBM, SUN, Digital

Τα χαρακτηριστικά δηλώνονται με χρήση μεταβλητών και αποτελούν τις καθολικές μεταβλητές που αφορούν συνολικά στο αντικείμενο. Κάθε ύπαρξη της κλάσης μπορεί να έχει διαφορετικές τιμές για τις μεταβλητές αυτές και κάθε μεταβλητή ορίζεται ως στιγμιαία μεταβλητή(instance variable). Κάθε αντικείμενο έχει μία αντίστοιχη στιγμιαία μεταβλητή, κάθε μεταβολή στην μεταβλητή αυτή τροποποιεί το χαρακτηριστικό του αντικειμένου. Οι στιγμιαίες μεταβλητές μπορούν να καθοριστούν κατά τη δήλωση-δημιουργία του αντικειμένου και παραμένουν σταθερές κατά τη διάρκεια ύπαρξης του αντικειμένου ή μεταβάλλονται κατά τη διάρκεια εκτέλεσης του προγράμματος. Εκτός από τις στιγμιαίες μεταβλητές, υπάρχουν και οι μεταβλητές κλάσης οι οποίες αφορούν στην κλάση και σε όλες τις στιγμές ύπαρξης της.

Η κατάσταση μίας κλάσης καθορίζει τη μέθοδο που χρησιμοποιούν οι στιγμές ύπαρξης μίας κλάσης για να ενεργούν προς τον "εαυτό" τους καθώς και για να επικοινωνούν με άλλες κλάσεις. Για παράδειγμα στην κλάση *computer* μπορεί να υπάρχουν δύο καταστάσεις :

*start* the computer *stop* the computer

Για τον καθορισμό της κατάστασης ενός αντικειμένου δημιουργούνται οι κατάλληλες μέθοδοι που συμπεριφέρονται όπως και οι συναρτήσεις (functions) στις άλλες γλώσσες προγραμματισμού, και ορίζονται κατά τη δήλωση της κλάσης. Οι μέθοδοι αποτελούν συναρτήσεις που δηλώνονται μέσα στις κλάσεις και επενεργούν στις στιγμές ύπαρξης της κλάσης. Οι μέθοδοι μπορεί να αφορούν σε περισσότερα από ένα αντικείμενα. Μία κλάση ή ένα αντικείμενο μπορεί να καλεί μεθόδους μίας άλλης κλάσης ή ενός άλλου αντικειμένου έτσι ώστε να τροποποιείται το περιβάλλον ή η κατάσταση του αντικειμένου. Αντίστοιχα με τις στιγμιαίες μεταβλητές και τις μεταβλητές κλάσης υπάρχουν στιγμιαίες μέθοδοι καθώς και μέθοδοι κλάσης.

## 2.3. Δημιουργία Κλάσης

Το πρώτο βήμα κατά τη δημιουργία μίας κλάσης είναι η δήλωση της. Στο παράδειγμα της κλάσης *computer* η αρχική δήλωση γίνεται με την εντολή :

```
class Computer {
        String model;
        String company;
        boolean computerState;
}
```

Το αποτέλεσμα είναι η δημιουργία της κλάσης Computer και των στιγμιαίων μεταβλητών model και company που είναι τύπου συμβολοσειράς(String)

καθώς και της computerState που είναι "λογικού" τύπου(boolean) και αναφέρεται στο αν ο ηλεκτρονικός υπολογιστής είναι ανοιχτός ή κλειστός.

Το δεύτερο βήμα αφορά στην προσθήκη μεθόδων στην κλάση. Ενδεικτικά προστίθεται η μέθοδος που αφορά στο ξεκίνημα του ηλεκτρονικού υπολογιστή. Οι παρακάτω εντολές προστίθενται στο σώμα δήλωσης της κλάσης *computer* :

```
void startComputer() {
        if (computerState = = true)
                System.out.println("Ο Υπολογιστής είναι ήδη ανοικτός!");
        else {
                computerState = true;
                System.out.println("Ο Υπολογιστής είναι τώρα ανοικτός");
        }
}
```

Επίσης μπορεί να συνπεριληφθεί και μία μέθοδος (prnAttrib) που θα εμφανίζει τις τρέχουσες τιμές των στιγμιαίων μεταβλητών της κλάσης :

```
void prnAttrib() {
        System.out.println("Ο Η/Υ είναι" + model +","+ company);
        if (computerState = = true)
                System.out.println("Ο Υπολογιστής είναι ανοικτός.");
        else
                System.out.println("Ο Υπολογιστής είναι κλειστός.");
}
```

Μετά τη δήλωση των μεθόδων και των μεταβλητών της κλάσης το πρόγραμμα αποθηκεύεται σε ένα αρχείο *Computer.java* (όπως αναφέρθηκε, πάντοτε τα Java αρχεία έχουν το ίδιο όνομα με το όνομα της κλάσης που περιλαμβάνουν). Με την εντολή *javac Computer.java* δημιουργείται η κλάση, αλλά για να μεταγλωτισθεί το πρόγραμμα χρειάζεται να υπάρχει κάποιο κυρίως πρόγραμμα *main*. Γενικά, απαιτείται η ύπαρξη ενός προγράμματος εφαρμογής που να χρησιμοποιεί την κλάση *Computer* ή να υπάρχει κάποιο *main* πρόγραμμα. Ενδεικτικά παρατίθεται ένα υπόδειγμα κυρίως προγράμματος *main :*

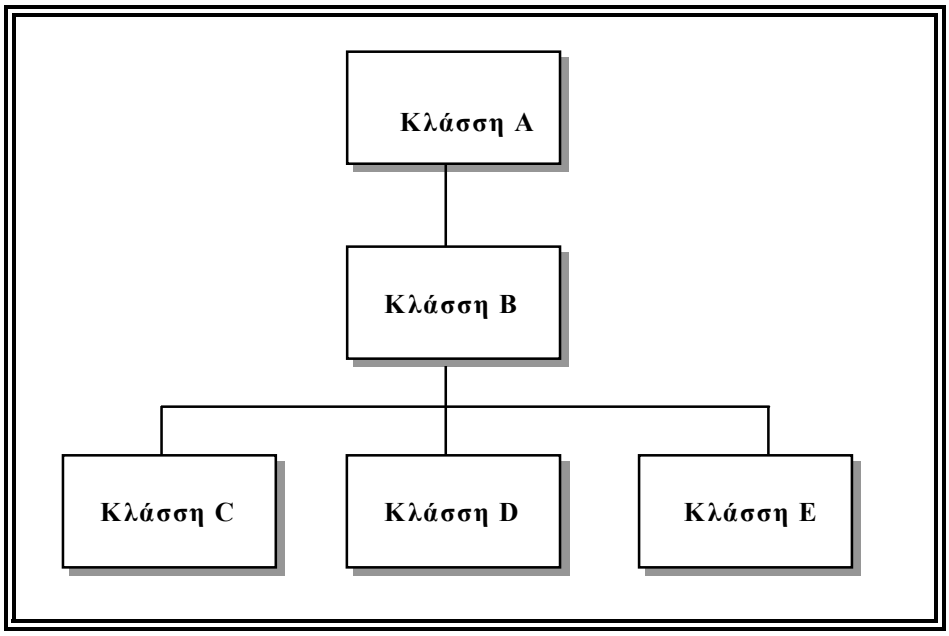
```
public static void main (String args[]) {
        Computer pc = new Computer();
        pc.model = "SparcStation1";
        pc.company = "SUN ";

        System.out.println("Κλήση μεθόδου εμφάνισης χαρακτηριστικών΄1");
        Computer.PrnAttrib();
        System.out.println("**********************************");
        System.out.println("ΕΚΚΙΝΗΣΗ ΥΠΟΛΟΓΙΣΤΗ");
```

```
        Computer.StartComputer()
        System.out.println("**********************************");
        System.out.println("Κλήση μεθόδου εμφάνισης χαρακτηριστικών 2");
        Computer.PrnAttrib();
}
```

## 2.4. Κληρονομικότητα και Ιεραρχία Κλάσεων

Κάθε βιβλιοθήκη κλάσης περιλαμβάνει ένα σύνολο από μηχανισμούς και τεχνικές που βοηθούν στην οργάνωση των κλάσεων και των μεθόδων τους. Η κληρονομικότητα είναι ένας από τους βασικότερους παράγοντες στον αντικειμενοστραφή προγραμματισμό και έχει άμεση επίδραση στον τρόπο με τον οποίο σχεδιάζονται οι κλάσεις στην Java Η κληρονομικότητα είναι ένας δυναμικός μηχανισμός που διαφοροποιεί τη μία κλάση από την άλλη. Μέσω της κληρονομικότητας όλες οι κλάσεις (αυτές που συντάσσει ο προγραμματιστής καθώς και κλάσεις από άλλες βιβλιοθήκες κλάσεων) κατατάσσονται με αυστηρή ιεραρχία.

Κάθε κλάση έχει μία υπερ-κλάση (η κλάση που προηγείται στα επίπεδα της ιεραρχίας) που είναι ο "γονέας" της και κάθε κλάση μπορεί να έχει μία ή περισσότερες υπο-κλάσεις (οι κλάσεις που έπονται στα επίπεδα της ιεραρχίας) που είναι οι απόγοοι της. Οι κλάσεις που βρίσκονται στο βάθος των επιπέδων ιεραρχίας υιοθετούν από τις γονικές κλάσεις τους που προηγούνται στο "δέντρο" της ιεραρχίας. Στο Σχήμα 2.1 η κλάση Β "κληρονομεί" από την κλάση Α, ενώ οι κλάσεις C, D, E κληρονομούν από τις κλάσεις Α και Β.

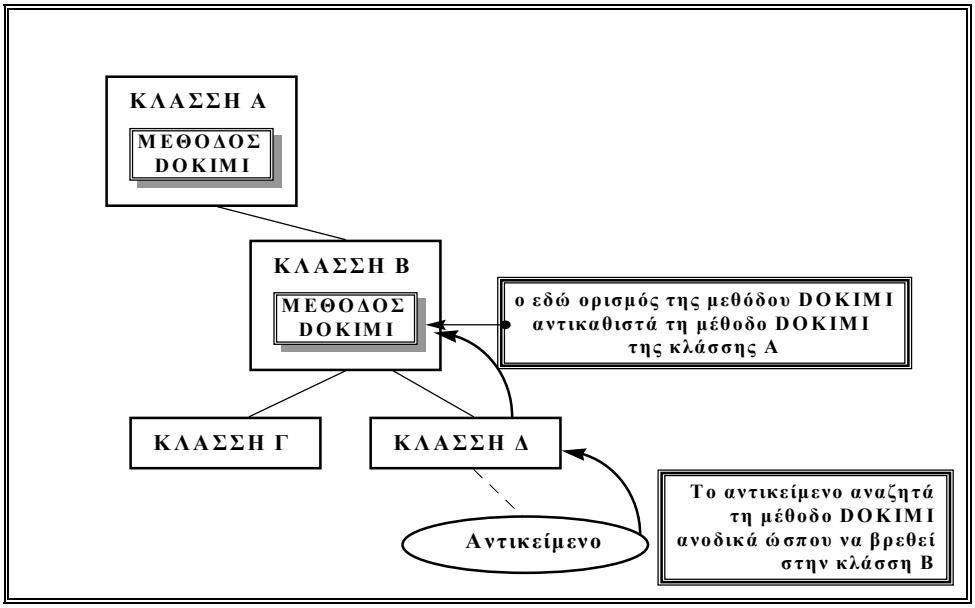



**Σχήμα 2-1:** ***Η ιεραρχία μίας κλάσης***

Οι υπο-κλάσεις κληρονομούν όλες τις μεθόδους και τις μεταβλητές από τις υπερ-κλάσεις τους. Ετσι η συμπεριφορά μίας υπερ-κλάσης μεταδίδεται προς όλες τις υπο-κλάσεις στο βάθος των επιπέδων ιεραρχίας. Επομένως, κάθε κλάση περιλαμβάνει ένα συνδυασμό όλων των χαρακτηριστικών των προγονικών της κλάσεων.

Στην κορυφή των κλάσεων ιεραρχίας στην Java βρίσκεται η κλάση *Object.* Η κλάση *Object* είναι η γενικότερη κλάση της Java και όλες οι κλάσεις κληρονομούν τα χαρακτηριστικά της. Κάθε κλάση του δέντρου ιεραρχίας προσθέτει επιπλέον χαρακτηριστικά στις κλάσεις προγόνους της με βάση αυτό που την αφορά. Οταν μία κλάση χρειάζεται να συμπεριλάβει τα χαρακτηριστικά κάποιας άλλης κλάσης δημιουργείται μία υπο-κλάση της η οποία κληρονομεί τα χαρακτηριστικά της και επιπλέον μπορεί να συμπληρωθεί με άλλες μεθόδους ή μεταβλητές. Είναι χρήσιμο κατά την ανάπτυξη μίας Java εφαρμογής, να προηγείται ο σχεδιασμός της ιεραρχίας των κλάσεων ώστε να οργανωθεί σωστά ο Java κώδικας. Κατά την ανάπτυξη των κλάσεων η πληροφορία (μεταβλητές, μέθοδοι) που αφορά σε περισσότερες από μία κλάσεις μπορεί να περιληφθεί σε μία υπερ-κλάση που την κληρονομούν οι υπο-κλάσεις της. Προγραμματιστικά αυτό είναι ιδιαίτερα χρήσιμο, διότι δε θα χρειαστεί να επαναληφθούν κοινά μέρη κώδικα και δε θα δαπανηθεί επιπλέον χρόνος για τη μετάφραση των υποκλάσεων.

Κάθε δημιουργία μίας νέας ύπαρξης της κλάσης συνεπάγεται την αυτόματη δημιουργία νέων στιγμών τόσο κάθε μεταβλητής που ορίζεται σε αυτήν την κλάση όσο και κάθε μεταβλητής που ορίζεται σε όλες τις υπερ-κλάσσεις της. Οι μέθοδοι παρουσιάζουν παρόμοια συμπεριφορά : τα νέα αντικείμενα έχουν πρόσβαση σε όλες τις μεθόδους της κλάσης τους και όλων των υπερ-κλάσεων αλλά οι ορισμοί των μεθόδων βρίσκονται δυναμικά όταν κληθεί η συγκεκριμένη μέθοδος. Ετσι, όταν καλείται η μέθοδος ενός συγκεκριμένου αντικειμένου, η Java πρώτα ελέγχει την κλάση του αντικειμένου για να βρει τον ορισμό της μεθόδου. Εάν δε βρεθεί η μέθοδος στο συγκεκριμένο αντικείμενο, αναζητά τη μέθοδο στην αμέσως παραπάνω υπερκλάση και συνεχίζει να ανεβαίνει στο δέντρο της ιεραρχίας των κλάσεων έως ότου βρει τη μέθοδο που αναζητά.




*Σχήμα 2.2*

**Σχήμα 2-2:** ***Η κληρονομικότητα μίας μεθόδου***

Οταν στην υπο-κλάση ορίζεται μία μέθοδος που έχει ακριβώς τα ίδια χαρακτηριστικά (όνομα, αριθμό και τύπο μεταβλητών) με μία μέθοδο που υπάρχει σε κάποια υπερ-κλάση της, τότε χρησιμοποιείται ο ορισμός της μεθόδου που δίνεται στην υπο-κλάση. Γενικά, χρησιμοποιείται η μέθοδος από την πρώτη κλάση στην ιεραρχία των κλάσεων στην οποία θα βρεθεί η δήλωση της. (Σχήμα 2.2).

## 2.5. Δημιουργία υπο-κλάσεων

Η δημιουργία μίας υπο-κλάσης είναι παρόμοια με τη δήλωση μίας κλάσης, με τη διαφορά ότι πρέπει να δηλωθεί η κλάση που είναι "γονέας" της συγκεκριμένης υποκλάσης. Η σύνδεση της υπο-κλάσης με την κλάση που είναι "γονέας" της γίνεται με χρήση της υπο-εντολής *extends*, όπως περιγράφεται παρακάτω. Η έννοια της υποκλάσης χρησιμοποιείται εκτενώς στην ανάπτυξη των μικρο-εφαρμογών (applets). Ολες οι μικρο-εφαρμογές αποτελούν υπο-κλάσεις της κλάσης *Applet* που περιλαμβάνεται στο πακέτο *java.applet* της Java. Ετσι η δήλωση μίας νέας κλάσης γίνεται ως εξής :

```
public class ComputerMessage extends java.applet.Applet {
……
}
```

Η δήλωση public δηλώνει ότι η συγκεκριμένη κλάση είναι διαθέσιμη και ορατή από όλες τις άλλες κλάσεις του Java προγράμματος που εκτελείται. Η κλάση ComputerMessage είναι υποκλάση της κλάσης java.applet.Applet της Java και αφορά την εμφάνιση του μηνύματος "ΑΓΟΡΕΣ ΠΩΛΗΣΕΙΣ COMPUTER" στην οθόνη μέσα από κάποιο html αρχείο Για να γίνει το

παράδειγμα περισσότερο αντιπροσωπευτικό η δήλωση της υπο-κλάσης συμπληρώνεται με τη δήλωση μίας μεθόδου paint() και με την εισαγωγή συγκεκριμένων κλάσεων της Java (Graphics, Font, Color) που βελτιώνουν την εμφάνιση του κειμένου. Ετσι προκύπτει :

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

public class ComputerMessage extends java.applet.Applet {

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

        public void paint(Graphics g) {
                g.setFont(f);
                g.setColor(Color.red);
                g.drawString("ΑΓΟΡΕΣ ΠΩΛΗΣΕΙΣ COMPUTER",5,50);
        }
}
```

Η δήλωση της μικρο-εφαρμογής στη Web σελίδα που αφορά την παραπάνω κλάση γίνεται (όπως περιγράφηκε και στο 1ο Κεφάλαιο) :

```
<HTML>
<HEAD><TITLE>Computer Sales </ TITLE></HEAD>
<BODY>
<p> Η Java εφαρμογή μας παρουσιάζει το μήνυμα :
<APPLET CODE=" ComputerMessage.class" WIDTH=150 HEIGHT=25>
</ APPLET> </BODY></HTML>
```

# 3. ΤΑ ΒΑΣΙΚΑ ΣΤΟΙΧΕΙΑ ΤΗΣ JAVA

## 3.1. Οι μεταβλητές

Οι μεταβλητές αποτελούν θέσεις μνήμης στις οποίες αποθηκεύονται τιμές αντίστοιχες με τον τύπο που έχει δηλωθεί για καθεμία. Οι μεταβλητές έχουν ένα όνομα, ένα τύπο και μία τιμή. Μία μεταβλητή πρέπει πρώτα να δηλωθεί πριν να χρησιμοποιηθεί. Μετά τη δήλωση της μπορούν να της ανατεθούν τιμές. Η Java διαθέτει τρία είδη μεταβλητών :

- στιγμιαίες μεταβλητές (instance variables) : χρησιμοποιούνται για να δηλώσουν τα χαρακτηριστικά ή την κατάσταση ενός συγκεκριμένου αντικειμένου και μπορεί να είναι διαφόρων τύπων. Η εξ'ορισμού αρχική τιμή μίας αριθμητικής μεταβλητής είναι *0*, μίας μεταβλητής χαρακτήρων είναι *'\0'* ενώ μίας λογικής (boolean) μεταβλητής είναι *false*.
- μεταβλητές κλάσεων (class variables) : παρόμοιες με τις στιγμιαίες μεταβλητές, με τη διαφορά ότι οι τιμές τους αφορούν σε όλες τις υπάρξεις της κλάσης τους χωρίς να χρειάζεται να δίνονται διαφορετικές τιμές για κάθε αντικείμενο. Η εξ'ορισμού αρχική τιμή μίας μεταβλητής κλάσης είναι *null*.
- τοπικές μεταβλητές (local variables) : δηλώνονται και χρησιμοποιούνται στο εσωτερικό της δήλωσης των μεθόδων και αφορούν συνήθως μεταβλητές προσωρινής χρήσης, όπως για παράδειγμα μετρητές, αθροιστές κλπ. Οι τοπικές μεταβλητές πρέπει να έχουν πάντοτε κάποια τιμή πριν χρησιμοποιηθούν.

Η Java δε διαθέτει ολικές μεταβλητές (global variables) όπως άλλες συμβατικές γλώσσες προγραμματισμού.Οι στιγμιαίες και οι μεταβλητές κλάσεων χρησιμοποιούνται για την ολική επικοινωνία των μεταβλητών μεταξύ των διαφόρων αντικειμένων.

### 3.1.1. Ορισμός μεταβλητών

Ο ορισμός μίας μεταβλητής περιλαμβάνει τη δήλωση του τύπου της και του ονόματος της, όπως για παράδειγμα :

```
int myComputer;
String myAddress;
Boolean isCompleted;
```

Οι ορισμοί των μεταβλητών μπορούν να τεθούν σε οποιοδήποτε σημείο δήλωσης μίας μεθόδου αλλά συνήθως τοποθετούνται στην αρχή της δήλωσης της μεθόδου. Μεταβλητές του ίδιου τύπου μπορούν να δηλωθούν μαζί και

επίσης υπάρχει δυνατότητα ανάθεσης αρχικών τιμών. Ακολουθούν ενδεικτικά παραδείγματα :

int myComputer, myJob, numTerminals=44; Boolean isCompleted = false;
String myAddress = "22 Egnatia Str"; int x=2, y=8, z=10;

Τα ονόματα των μεταβλητών στη Java αρχίζουν με κάποιο γράμμα, το σύμβολο υπογράμμισης (underscore) (_) ή το σύμβολο του δολαρίου ($). Δεν μπορούν να ξεκινούν με αριθμό. Μετά τον πρώτο χαρακτήρα, οι υπόλοιποι χαρακτήρες μπορεί να περιλαμβάνουν οποιονδήποτε αριθμό ή γράμμα ή σύμβολο. Τα σύμβολα %, *, @ αφορούν τελεστές στην Java και θα πρέπει να χρησιμοποιούνται με προσοχή στα ονόματα των μεταβλητών. Η Java κάνει διάκριση μεταξύ των κεφαλαίων και των μικρών γραμμάτων, δηλαδή η μεταβλητή X είναι διαφορετική από τη μεταβλητή x. Γενικά στον προγραμματισμό της Java ισχύει ο κανόνας ότι τα ονόματα των μεταβλητών είναι ενδεικτικά του περιεχομένου τους, μπορεί να αποτελούνται από συνδυασμό διαφόρων λέξεων με την πρώτη λέξη σε μικρά γράμματα και στις υπόλοιπες λέξεις το πρώτο τους γράμμα να είναι κεφαλαίο ενώ τα υπόλοιπα να είναι μικρά :

long realBigNumber";
Boolean computerSalesManagerOfMcompany = false;

### 3.1.2. Τύποι μεταβλητών

Ο τύπος κάθε μεταβλητής καθορίζει το είδος των τιμών που μπορεί να έχει η μεταβλητή και μπορεί είναι ένα από τα παρακάτω είδη :

- ένας από τους 8 βασικούς πρωταρχικούς τύπους της Java :

| *ΤΥΠΟΣ* | *ΜΕΓΕΘΟΣ* | *ΕΥΡΟΣ ΤΙΜΩΝ* |
|---|---|---|
| boolean | 8 bits | TRUE ή FALSE |
| byte | 8 bits | -128 έως 127 |
| char | 16 bits | |
| short | 16 bits | -32.768 έως 32.767 |
| int | 32 bits | -2.147.483.648 έως 2.147.483.647 |
| long | 64 bits | -9223372036854775808 έως 9223372036854775807 |
| float | 32 bits | απλής ακρίβειας, σύμφωνα με προείπω IEEE 754 |
| double | 64 bits | διπλής ακρίβειας |

Οι παραπάνω τύποι δεδομένων είναι ανεξάρτητοι της μηχανής και είναι οι πρωταρχικοί τύποι μεταβλητών που περιλαμβάνονται στην Java. .

- το όνομα μίας κλάσης

- ένας πίνακας
  Οι πίνακες χρησιμοποιούνται για την αποθήκευση μία ακολουθίας αντικειμένων. Κάθε στοιχείο του πίνακα κρατά ένα αντικείμενο και όλα τα στοιχεία του πίνακα είναι του ιδίου τύπου αντικειμένων. Υπάρχουν πίνακες όλων των πρωταρχικών τύπων καθώς και πίνακες πινάκων. Για να δημιουργηθεί ένας πίνακας ακολουθούνται τα παρακάτω βήματα :
  1. Δήλωση της μεταβλητής που θα περιέχει τον πίνακα, ακολουθούμενη από [ ] για μονοδιάστατο ή [ ] [ ] για πολυ-διάστατο πίνακα. Οι αγκύλες μπορούν να ακολουθούν το όνομα του τύπου και όχι το όνομα της μεταβλητής :
     String keimenoGrafos[ ]; ή String [ ] keimenoGrafos;

  2. Δημιουργία ενός νέου αντικειμένου πίνακα και ανάθεση του στην μεταβλητή που δηλώθηκε στο Βήμα 1.

  3. Αποθήκευση στοιχείων στον πίνακα.

Είναι χαρακτηριστικό ότι η Java δε διαθέτει αντίστοιχο του ορισμού *typedef* (της C ή της C++). Η δήλωση νέων τύπων μεταβλητών στην Java γίνεται με δήλωση μίας νέας κλάσης και με δήλωση μεταβλητών τύπου αυτής της κλάσης.

## 3.2. Εκφράσεις και τελεστές

Οι εκφράσεις αποτελούνται από εντολές που επιστρέφουν τιμές αποτελεσμάτων, ενώ οι τελεστές είναι τα ειδικά σύμβολα που χρησιμοποιούνται συνήθως στις εκφράσεις. Οι τελεστές στην Java μπορεί να είναι αριθμητικοί, να αφορούν σε διάφορες μορφές ανάθεσης τιμών, να εκτελούν αύξηση ή μείωση τιμών ή να περιλαμβάνουν λογικές πράξεις.

### 3.2.1. Αριθμητικοί Τελεστές

Η Java περιλαμβάνει πέντε τελεστές για τις βασικές πράξεις :

| | | | |
|---|---|---|---|
| + | ΠΡΟΣΘΕΣΗ | * | ΠΟΛΛΑΠΛΑΣΙΑΣΜΟ |
| - | ΑΦΑΙΡΕΣΗ | / | ΔΙΑΙΡΕΣΗ |
| % | ΥΠΟΛΟΙΠΟ | | |

Ακολουθεί ένα παράδειγμα απλής αριθμητικής :

| *ΠΡΟΓΡΑΜΜΑ* | *ΑΠΟΤΕΛΕΣΜΑΤΑ* |
|---|---|
| class operationArithmetic {<br>public static void main (String args[]) {<br>short a = 14; | a+b=16 |

```
        int b = 2;                                          a-b= 12
        float a1 = 10.5f;
        float b1 = 8;                                       a/b= 7

        System.out.println("a+b=" + (a+b));                 a%b=0
        System.out.println("a-b="+(a-b));                   a1/b1=1.3125
        System.out.println("a/b=" + (a/b));
        System.out.println("a%b=" + (a%b));
        System.out.println("a1/b1=" + (a1/b1));
    }
}
```

### 3.2.2. Ανάθεση τιμών σε μεταβλητές

Η ανάθεση τιμών στις μεταβλητές αποτελεί μία μορφή έκφρασης, διότι κάθε ανάθεση τιμών καταλήγει σε μία τιμή η οποία στη συνέχεια μπορεί να δοθεί σε άλλες μεταβλητές, όπως για παράδειγμα :

```
a = b = c = 1;
```

Υπολογίζεται η έκφραση μετά το σύμβολο του ίσον και το αποτέλεσμα ανατίθεται στη μεταβλητή αριστερά του ίσον. Υπάρχει συντομογραφία των συχνότερα χρησιμοποιούμενων αναθέσεων τιμών :

| | |
|---|---|
| a += b (που σημαίνει a=a+b) | a -= b (που σημαίνει a=a-b) |
| a *= b (που σημαίνει a=a*b) | a /= b (που σημαίνει a=a/b) |

### 3.2.3. Αυξο-μείωση τιμών

Οπως στην C και στην C++, οι τελεστές ++ και -- χρησιμοποιούνται για την αύξηση ή αντίστοιχα μείωση μίας τιμής κατά 1. Η Java επιτρέπει στην έκφραση a++ η μεταβλητή a να είναι και τύπου float. Οι τελεστές ++ και – μπορεί να προηγούνται ή να έπονται του ονόματος μίας μεταβλητής :

```
b = a++;      η b παίρνει την τιμή της a πριν την αύξηση της κατά 1
b = ++a;      η b παίρνει την τιμή της a μετά την αύξηση της κατά 1
```

Ακολουθεί ένα παράδειγμα με αυξο-μειώσεις τιμών :

| *ΠΡΟΓΡΑΜΜΑ* | *ΑΠΟΤΕΛΕΣΜΑΤΑ* |
|---|---|
| class incDecr { | αύξηση μετά |
| public static void main (String args[]) { | |
| int a = 0; | a= 1 b=0 |
| int b = 0; | |
| | αύξηση πριν |

```
            b = a++;                                    a= 2  b=2
            System.out.println("αύξηση μετά ");
            System.out.println("a=" + a+"b="+b);
            b = ++a;
            System.out.println("αύξηση πριν ");
            System.out.println("a=" + a+"b="+b);
        }
}
```

### 3.2.4. Συγκρίσεις

Η Java διαθέτει ένα σύνολο από εκφράσεις που χρησιμοποιούνται για έλεγχο και συγκρίσεις. Ολες οι εκφράσεις αυτές επιστρέφουν boolean τιμή (δηλαδή true ή false). Στον παρακάτω πίνακα παρουσιάζονται οι τελεστές σύγκρισης καθώς και οι λογικοί τελεστές της Java :

| *ΤΕΛΕΣΤΗΣ* | *ΕΝΕΡΓΕΙΑ* | *ΤΕΛΕΣΤΗΣ* | *ΕΝΕΡΓΕΙΑ* |
|---|---|---|---|
| = = | ισότητα | != | ανισότητα |
| < (<=) | μικρότερο (ίσον) | > (>=) | μεγαλύτερο (ίσον) |
| & ή && | λογικό AND | \| ή \|\| | λογικό OR |
| ^ | λογικό XOR | ! | λογικό NOT |

Με τους τελεστές &, |, ^ υπολογίζονται πάντοτε οι τιμές και των δύορισμάτων μίας έκφρασης ανεξάρτητα από το αποτέλεσμα ενώ αντίστοιχα με τους && και | | εάν το αριστερό μέρος μίας έκφρασης είναι false δεν υπολογίζεται το δεξιό μέρος της έκφρασης και επιστρέφεται η τιμή false ως τελική απάντηση.

Η Java υποστηρίζει και δυαδικούς τελεστές, "κληρονομώντας" την C και την C++. Οι δυαδικοί τελεστές περιλαμβάνουν :

| *ΤΕΛΕΣΤΗΣ* | *ΕΝΕΡΓΕΙΑ* | *ΤΕΛΕΣΤΗΣ* | *ΕΝΕΡΓΕΙΑ* |
|---|---|---|---|
| << | αριστερά μετατόπιση | >> | δεξιά μετατόπιση |
| <<= | ανάθεση αριστερά μετατόπισης(a=a<<b) | >>= | ανάθεση δεξιά μετατόπισης(a=a>>b) |
| >>> | μηδενισμός δεξιά μετατόπισης | >>>= | ανάθεση από μηδενισμό δεξιά μετατόπισης |

| | | | |
|---|---|---|---|
| & | δυαδικό AND | \| | δυαδικό OR |
| ^ | δυαδικό XOR | ~ | δυαδικό συμπλήρωμα |
| a &= b | AND ανάθεση (a =a&b) | a \|= b | OR ανάθεση (a =a \| b) |
| a ^= b | XOR ανάθεση (a =a ^ b) | | |

Ολοι οι τελεστές της Java ιεραρχούνται με βάση τη σειρά προτεραιότητας τους. Στη συνέχεια παρουσιάζεται η προτεραιότητα των τελεστών και πράξεων της Java, από τον ισχυρότερο προς τον ασθενέστερο :

| *ΤΕΛΕΣΤΗΣ* | *ΠΑΡΑΤΗΡΗΣΕΙΣ* |
|---|---|
| . [ ] ( ) | Η τελεία (.) χρησιμοποιείται για την προσπέλαση σε μεθόδους και μεταβλητές που περιέχονται στη δήλωση των αντικειμένων και των κλάσεων.<br>Οι [ ] χρησιμοποιούνται στη δήλωση των πινάκων. |
| ++ - - ! ~<br>instanceof | Η instanceof επιστρέφει true ή false αντίστοιχα εάν ένα αντικείμενο υπάρχει στην επονομαζόμενη κλάση ή εάν υπάρχει σε κάποια από τις υπερκλάσεις της. |
| new (type)expression | Ο τελεστής new δημιουργεί νέες στιγμές μίας κλάσης. |
| * / % | |
| + - | |
| >> << >>> | |
| < > <= >= | |
| == != | |
| & | AND |
| ^ | XOR |
| \| | OR |
| && | Λογικό AND |
| \|\| | Λογικό OR |
| ? : | συντομογραφία της if … then … else |
| = += -= *= /= %= ^=<br>&= \|= <<= >>= >>>= | Διάφορες αναθέσεις τιμών |

### 3.2.5. Αριθμητική Συμβολοσειρών

Η Java χρησιμοποιεί τον τελεστή της πρόσθεσης (+) και για να δημιουργήσει ή να "αθροίσει" συμβολοσειρές. Οπως παρουσιάστηκε σε προηγούμενα παραδείγματα Java προγραμμάτων η εντολή :

System.out.println(“a=” + a+”b=”+b);

παράγει την εκτύπωση μίας συμβολοσειράς που αποτελείται από τις εκφράσεις που βρίσκονται μέσα στα “ ” ακολουθούμενες από τις τιμές των μεταβλητών στις θέσεις που δίνονται με χρήση του τελεστή +. Ενα αντικείμενο ή ένας τύπος αντικειμένων μετατρέπεται σε συμβολοσειρά (τύπος string) με χρήση της μεθόδου *toString*().

Ο τελεστής += χρησιμοποιείται και για μεταβλητές τύπου string. Ετσι η έκφραση *myComputer += “Danae”;* ισοδυναμεί με την έκφραση : *myComputer = myComputer+“Danae”;* το αποτέλεσμα της οποίας είναι η ανάθεση στην μεταβλητή *myComputer* της προηγούμενης τιμής της ακολουθούμενη από τη συμβολοσειρά *Danae.*

### 3.2.6. Τύποι Κυριολεκτικών

Ο όρος “κυριολεκτικά” αναφέρεται στην ανάθεση τιμών με βάση την ακριβή έκφραση που πληκτρολογείται. Υπάρχουν κυριολεκτικά τύπου αριθμητικών, boolean, character, string,

- Αριθμητικά Κυριολεκτικά (Number Literals) :

Η Java χρησιμοποιεί τα συνηθισμένα κυριολεκτικά(literals) που είναι προσημασμένοι δεκαδικοί αριθμοί των 32 bit. Τα κυριολεκτικά που αρχίζουν με το 0 είναι οκταδικά, ενώ εκείνα που αρχίζουν με 0x ή 0X είναι δεκαεξαδικά. Τα κυριολεκτικά που είναι μεγαλύτερα από 0X7FFFFFFF θεωρούνται αυτόματα μεγάλοι ακέραιοι(*long*). Γενικά, κάθε ακέραιος μπορεί να γίνει :

1. *long* με την τοποθέτηση ενός *l* ή *L* στο τέλος του,
2. κινητής υποδιαστολής (*float)* με τοποθέτηση ενός f ή F στο τέλος του
3. double με την τοποθέτηση ενός d ή D στο τέλος του.

- Λογικά Κυριολεκτικά (Boolean Literals):

Η Java διαθέτει δύο λογικά κυριολεκτικά : true και false.

- Κυριολεκτικά Χαρακτήρων (Character Literals):

Τα κυριολεκτικά χαρακτήρων περιλαμβάνουν τη δήλωση ενός χαρακτήρα μέσα στα απλά εισαγωγικά (‘ ‘ ). Οι χαρακτήρες αυτοί αποθηκεύονται ως 16-bit Unicode χαρακτήρες. Στη συνέχεια παρουσιάζονται οι ειδικοί μη-εκτυπώσιμοι χαρακτήρες καθώς και αντιπροσωπευτικοί χαρακτήρες από το σύνολο των Unicode χαρακτήρων.

| *Escape ΧΑΡΑΚΤΗΡΑΣ* | *ΕΝΕΡΓΕΙΑ* | *Escape ΧΑΡΑΚΤΗΡΑΣ* | *ΕΝΕΡΓΕΙΑ* |
|---|---|---|---|
| \n | αλλαγή γραμμής | \r | enter |
| \t | tab | \f | αλλαγή σελίδας |
| \b | backspace | \\ | σύμβολο \ |
| \’ | απλή απόστροφος | \” | διπλή απόστροφος |

| \ddd | οκταδικός | \xdd | δεκαεξαδικός |
|---|---|---|---|
| \udddd | Unicode χαρακτήρας | | |

- Κυριολεκτικά Συμβολοσειρών (String Literals):

Τα κυριολεκτικά συμβολοσειρών αφορούν στη δήλωση ενός αριθμού συνεχόμενων χαρακτήρων που περικλείονται μεταξύ των διπλών εισαγωγικών (“ “). Οταν χρησιμοποιούμε ένα κυριολεκτικό συμβολοσειρών, η Java δημιουργεί αυτόματα μία στιγμή ύπαρξης της κλάσης String. Αυτή είναι η βασική διαφοροποίηση των κυριολεκτικών συμβολοσειρών από τους άλλους τύπους κυριολεκτικών.

## 3.3. Ελεγχος Ροής

Η Java παρέχει ένα πλήρες σύνολο εντολών ελέγχου ροής :

- Εντολή if
- Βρόγχος for
- Βρόγχος while
- Εντολή switch

Οι παραπάνω έλεγχοι της ροής του προγράμματος είναι εντολές και όχι παραστάσεις, δηλαδή δεν έχουν τιμή ή τύπο.

### 3.3.1. Η Εντολή if

Η απλούστερη εντολή ελέγχου ροής είναι η εντολή if :

```
if (boolean έκφραση)  {
      // . . . οποιοσδήποτε αριθμός εντολών
}
else {
     // . . . οποιοσδήποτε αριθμός εντολών
}
```

Εάν η boolean έκφραση είναι αληθής (true) εκτελούνται οι εντολές που περιέχονται στο πρώτο σύνολο εντολών, ενώ εάν είναι ψευδής (false) εκτελούνται οι εντολές που ακολουθούν το else. Ο όρος else είναι προαιρετικός.

Ακολουθεί μία ενδεικτική εντολή if που αφορά στην κλάση *Computer* που είχε δημιουργηθεί σε προηγούμενο Κεφάλαιο :

```
if (computerState = = true)
     System.out.println(“Ο Υπολογιστής είναι ήδη σε λειτουργία.”);
else {
     System.out.println(“Ο Υπολογιστής τίθεται τώρα σε λειτουργία.”);
     if (powerButton >= 1)
```

```
                computerState = true;
        else
        System.out.println("Ο Υπολογιστής έχει πρόβλημα λειτουργίας.");
    }
```

Μία εναλλακτική δυνατότητα στις εντολές υποθέσεων, είναι η χρήση του "υποθετικού" τελεστή (conditional operator) αντί για τη χρήση της if - else. Ο τελεστής αυτός επιλέγεται κυρίως σε σύντομες εντολές υποθέσεων και έχει τη μορφή :

```
eleghos          ? true-apotelesma        : false-apotelesma
```

Ο eleghos είναι μία έκφραση που παίρνει μόνο τις τιμές true ή false. Εάν η τιμή του eleghos είναι true, ο υποθετικός τελεστής επιστρέφει την τιμή true-apotelesma ενώ εάν ο eleghos είναι false επιστρέφει την τιμή false-apotelesma.

Για παράδειγμα η εντολή :

```
int mini = a < b ? a : b;
```

ελέγχει τη συνθήκη a < b και αναθέτει την τιμή του μικρότερου από τους a, b στην ακέραια μεταβλητή mini.

### 3.3.2. Ο Βρόγχος while

Ο βρόγχος while παρουσιάζεται σε δύο παραλλαγές

- while (boolean έκφραση) {
  // . . . οποιοσδήποτε αριθμός εντολών
  }

Υπολογίζεται η boolean έκφραση, εάν είναι αληθής εκτελούνται όλες οι εντολές που περιλαμβάνονταιμέσα στα άγκιστρα, ενώ αν είναι ψευδής το πρόγραμμα αγνοεί όλο τον αριθμό των προτάσεων που περικλείει η while και μεταφέρεται στην πρώτη γραμμή εντολής που ακολουθεί το βρόγχο της while. Η εκτέλεση της while συνεχίζεται όσο η λογική παράσταση της παραμένει αληθής

- do {
  // . . . οποιοσδήποτε αριθμός εντολών
  } while (boolean έκφραση);

Υπολογίζεται η boolean έκφραση αφού εκτελεστούν όλες οι εντολές που περικλείονται μέσα στο βρόγχο της do … while. Εάν η boolean έκφραση είναι αληθής ο έλεγχος μεταφέρεται στην αρχή του βρόγχου, ενώ αν είναι ψευδής ο έλεγχος μεταφέρεται στην εντολή που ακολουθεί αμέσως μετά την do … while. Παρατίθενται στη συνέχεια αποσπάσματα των δύο while εντολών :

| *ΠΡΟΓΡΑΜΜΑ* | *ΑΠΟΤΕΛΕΣΜΑΤΑ* |
|---|---|
| ……..<br>int a = 1;<br>while (a <= 5 ) {<br>if (a > 1)<br>System.out.println(“επανάληψη “+a);<br>a++;<br>};<br>…… | <br><br>επανάληψη 2<br>επανάληψη 3<br>επανάληψη 4<br>επανάληψη 5 |

| *ΠΡΟΓΡΑΜΜΑ* | *ΑΠΟΤΕΛΕΣΜΑΤΑ* |
|---|---|
| ……..<br>int a = 1;<br>do {<br>System.out.println(“επανάληψη “+a);<br>a++;<br>} while (a <= 5);<br>…… | <br>επανάληψη 1<br>επανάληψη 2<br>επανάληψη 3<br>επανάληψη 4<br>επανάληψη 5 |

### 3.3.3. Ο Βρόγχος for

Ο βρόγχος for είναι ο συνηθέστερος βρόγχος και έχει τη μορφή :

```
for (έκφραση1; boolean έκφραση; έκφραση2) {
    //  . . . οποιοσδήποτε αριθμός εντολών
}
```

Κατά την εκτέλεση του βρόγχου εκτελούνται τα παρακάτω βήματα :

1. Υπολογίζεται η *έκφραση1*.
2. Υπολογίζεται η *boolean έκφραση*. Εάν το αποτέλεσμα της είναι true, εκτελείται ο αριθμός των εντολών του βρόγχου. Εάν το αποτέλεσμα της είναι false ο έλεγχος του προγράμματος μεταφέρεται στην πρώτη εντολή που ακολουθεί μετά το τέλος του βρόγχου.
3. Υπολογίζεται η *έκφραση2*.
4. Επανάληψη Βήματος 2.

Γενικά η *έκφραση1* αφορά στην εκκίνηση του βρόγχου και οι μεταβλητές που δηλώνονται σε αυτήν είναι τοπικής χρήσης του for βρόγχου. Η boolean έκφραση καθορίζει τον αριθμό επαναλήψεων του βρόγχου, ενώ η *έκφραση2* είναι το βήμα μεταβολής που μπορεί να είναι κάποια έκφραση ή κάποια κλήση συνάρτησης.

### 3.3.4. Η Εντολή switch

Η Εντολή switch είναι χρήσιμη για τις περιπτώσεις που έχουμε επιλογή από ένα σύνολο εναλλακτικών τιμών :

```
switch (έκφραση) {
    case τιμή έκφρασης  :
          // … αριθμός εντολών
          break;
    case τιμή έκφρασης  :
          // … αριθμός εντολών
          break;
    ……..
    default :
          // … αριθμός εντολών
}
```

Αρχικά υπολογίζεται η *έκφραση* που μπορεί να είναι τύπου byte, char, short ή int. Στη συνέχεια, η τιμή της συγκρίνεται με κάθε μία από τις *τιμή-έκφραση* που παρατίθενται στις προτάσεις case. Ο έλεγχος του προγράμματος μεταφέρεται στην case που έχει ίδια τιμή με την τιμή της *έκφρασης*. Αν καμία τιμή δεν είναι ίση με την τιμή της *έκφρασης*, ο έλεγχος μεταφέρεται στον αριθμό των εντολών της default. Εάν δεν υπάρχει default ο έλεγχος μεταφέρεται στην πρόταση που ακολουθεί το τέλος της εντολής switch. Υπάρχει δυνατότητα να υπάρχει ένα σύνολο εντολών για περισσότερες από μία case, όπως στο ακόλουθο παράδειγμα :

```
switch (a ) {
      case 2 :
      case 4 :
      case 6 :
      case 8:
           System.out.println("Ο a είναι άρτιος αριθμός");
      default : System.out.println("Ο a είναι περιττός αριθμός");
};
```

### 3.3.5. Έξοδος από βρόγχους επανάληψης

Υπάρχει δυνατότητα και στην Java να μεταφερθεί ο έλεγχος της ροής του προγράμματος εκτός κάποιου βρόγχου επανάληψης πριν να προκύψει false τιμή της λογικής έκφρασης που καθορίζει το βρόγχο επανάληψης.

Οι λέξεις κλειδιά *break* και *continue* επιτρέπουν στο πρόγραμμα να διακόψει την εκτέλεση των εντολών μέσα στο βρόγχο, Ειδικότερα, η εντολή break μεταφέρει τον έλεγχο αμέσως έξω από το βρόγχο ενώ η εντολή break ακολουθούμενη από μία ετικέτα του προγράμματος μεταφέρει τον έλεγχο στην εντολή με την ετικέτα αυτή που περιλαμβάνεται στην τρέχουσα μέθοδο. Η ελεγχόμενη έξοδος από ένα βρόγχο (με τη χρήση της break με ετικέτα) αποτελεί την εκδοχή της Java που πλησιάζει την εντολή goto άλλων γλωσσών του συμβατικού προγραμματισμού. Η *continue* αγνοεί τις εντολές (του

βρόγχου) που την ακολουθούν και επαναλαμβάνει το πέρασμα του βρόγχου από την αρχή.

Στο παρακάτω παράδειγμα δίνεται μία περίπτωση της εντολή break ακολουθούμενη από ετικέτα, που έχει αποτέλεσμα την ταυτόχρονη έξοδο από όλες τις φωλιασμένες εντολές ελέγχου ροής :

| *ΠΡΟΓΡΑΜΜΑ* | *ΑΠΟΤΕΛΕΣΜΑΤΑ* |
|---|---|
| ……..<br>etiketa :<br>for (int a = 1; a <= 5; a++) {<br>for (int b = 1; b <= 3; b++){<br>System.out.println("a="+a+"b="+b);<br>if ((a+b) > 4)<br>break etiketa;<br>}<br>};<br>System.out.println("Εξοδος από τους βρόγχους");<br>…… | a=1 b=1<br>a=1 b=2<br>a=1 b=3<br>a=2 b=1<br>a=2 b=2<br>a=2 b=3<br>Εξοδος από τους βρόγχους |

## 3.4. Διαχείριση αντικειμένων

Η Java ως αντικειμενοστραφής γλώσσα προγραμματισμού, περιλαμβάνει εργαλεία και τεχνικές που αφορούν στη διαχείριση των αντικειμένων. Οι τεχνικές αυτές περιλαμβάνουν τη δημιουργία των αντικειμένων, την τροποποίηση τους, τη μετακίνηση τους, την αλλαγή των μεταβλητών τους, την κλήση των μεθόδων τους, και ακόμα το συνδυασμό διαφόρων αντικειμένων μεταξύ τους.

### 3.4.1. Δημιουργία νέων αντικειμένων

Οπως περιγράφηκε, όταν γράφεται ένα Java πρόγραμμα, αρχικά δηλώνεται το σύνολο των κλάσεων του προβλήματος. Οι κλάσεις αποτελούν τα πρότυπα με βάση τα οποία δημιουργούνται τα αντικείμενα και κάθε αντικείμενο ακολουθεί τα χαρακτηριστικά και τις ιδιότητες μίας κλάσης. Για να δημιουργηθεί ένα αντικείμενο χρησιμοποιείται η λέξη κλειδί *new* ακολουθούμενη από το όνομα της κλάσης με βάση το πρότυπο της οποίας δημιουργείται το αντικείμενο. Για παράδειγμα οι εντολές :

String lexi = new String(); Computer pc1 = new Computer();

δημιουργούν δύο νέα αντικείμενα(lexi, pc1) που αποτελούν στιγμές των κλάσεων String και Computer αντίστοιχα.

Οι παρενθέσεις που ακολουθούν το όνομα της κλάσης είναι σημαντικές και δεν παραλείπονται ποτέ. Οταν δεν περιλαμβάνεται παράμετρος μεταξύ των

παρενθέσεων τότε έχουμε τη δημιουργία της απλούστερης μορφής ενός αντικειμένου, ενώ η δήλωση παραμέτρων αφορά στις αρχικές τιμές που θα πάρουν οι στιγμιαίες μεταβλητές. Ο αριθμός και ο τύπος των παραμέτρων που μπορεί να χρησιμοποιηθεί με τη *new* δηλώνεται κατά τον ορισμό της κλάσης με χρήση μίας ειδικής μεθόδου που λέγεται μέθοδος "κατασκευαστής" (constructor method). Στο παράδειγμα που ακολουθεί παρουσιάζονται και οι δύο μορφές δημιουργίας αντικειμένων :

| *ΠΡΟΓΡΑΜΜΑ* | *ΑΠΟΤΕΛΕΣΜΑΤΑ* |
|---|---|
| import java.util.Date;<br>import Computer();<br><br>class createObj {<br>public static void main (String args[]) {<br>Date hmerom;<br>Computer pc1=new Computer();<br><br>hmerom = new Date(97, 7, 1, 7, 30);<br>System.out.println("Ημερομηνία :"+ hmerom);<br><br>pc1.model = "Pentium";<br>System.out.println("Αγορά H/Y:"+ pc1.model);<br>}<br>};<br>…… | Ημερομηνία : Fri Aug 01 07:30:00 PDT 1997<br>Αγορά H/Y: Pentium |

Γενικά, όταν χρησιμοποιείται η λέξη κλειδί new εκτελούνται τα παρακάτω :

1. δημιουργείται μία νέα στιγμή της αναφερόμενης κλάσης
2. α απαραίτητος χώρος μνήμης ανατίθεται σε αυτή τη νέα στιγμή της κλάσης
3. καλείται η ειδική μέθοδος ΄"κατασκευής" που έχει δηλωθεί στην κλάση

Οι μέθοδοι κατασκευής περιλαμβάνουν τον απαραίτητο κώδικα που υποστηρίζει την εκκίνηση του αντικειμένου. Ετσι, δίνουν τις αρχικές τιμές στο νέο αντικείμενο, στις μεταβλητές του, δημιουργούν κάθε άλλο αντικείμενο που το νέο αντικείμενο χρειάζεται και γενικά εκτελούν όλες τις λειτουργίες που χρειάζεται το αντικείμενο για να ξεκινήσει να υπάρχει. Πολλαπλές δηλώσεις μεθόδων κατασκευαστών σε μία κλάση επιτρέπονται με δεδομένο ότι κάθε μέθοδος κατασκευαστής έχει το δικό του αριθμό και τύπο παραμέτρων. Επομένως όταν δημιουργείται το νέο αντικείμενο καλείται η κατάλληλη μέθοδος κατασκευής που συμφωνεί με τον αριθμό και τύπο παραμέτρων που ακολουθούν το όνομα της κλάσης στη δήλωση new.

Ειδικότερα για τη δημιουργία πινάκων μπορεί να χρησιμοποιηθεί η λέξη κλειδί new ή να ανατεθούν τιμές κατευθείαν στα στοιχεία του πίνακα. Ετσι, για παράδειγμα, η εντολή :

String[ ] students1 = new String[45];

δημιουργεί ένα πίνακα 45 θέσεων τύπου συμβολοσειράς. Οταν χρησιμοποιείται το new θα πρέπει να δηλώνεται και το μέγεθος του πίνακα με αποτέλεσμα να δημιουργείται ο χώρος με κάθε στοιχείο να περιέχει τις εξ' ορισμού τιμές του τύπου του πίνακα. Η εντολή :

String[ ] students2 = {"Παπαδόπουλος", "Αλεξίου","Αθανασίου"];

δημιουργεί ένα πίνακα 3 θέσεων και αναθέτει τις συγκεκριμένες συμβολοσειρές στις θέσεις του πίνακα.

### 3.4.2. Αναγνώριση - Συγκρίσεις αντικειμένων

Υπάρχει δυνατότητα να βρεθεί η κλάση με βάση την οποία κατασκευάστηκε το αντικείμενο διότι η κλάση Object περιλαμβάνει τη μέθοδο getClass() (προσβάσιμη από όλα τα αντικείμενα) που διαθέτει τη μέθοδο getName() η οποία επιστρέφει το όνομα της κλάσης. Ετσι, η εντολή :

*String onoma = antik.getClass().getName();*

επιστρέφει το όνομα της κλάσης στο οποίο αναφέρεται το αντικείμενο antik.

Ενας ακόμα έλεγχος μπορεί να γίνει με χρήση του τελεστή instanceof. Η σύνταξη του τελεστή είναι :

*αντικείμενο* instanceof *κλάση*

και επιστρέφει true εάν το αντικείμενο προέρχεται από την κλάση, διαφορετικά επιστρέφει false.

Οι τελεστές της ισότητας (= =) και ανισότητας (!=) χρησιμοποιούνται και για συγκρίσεις αντικειμένων και εξετάζουν εάν τα δύο μέλη αφορούν ακριβώς στο ίδιο αντικείμενο ή όχι. Χρειάζεται ανάπτυξη ειδικής κλάσης για να γίνεται σύγκριση μεταξύ των στιγμών ύπαρξης των αντικειμένων. Για παράδειγμα, μπορεί να έχουμε δύο ανεξάρτητα αντικείμενα (το καθένα έχει δικό του χώρο μνήμης με χρήση της new) τύπου String με ακριβώς το ίδιο περιεχόμενο αλλά ο τελεστής = = έχει αποτέλεσμα false διότι αυτά τα αντικείμενα δεν περιλαμβάνονται στο ίδιο αντικείμενο. Ετσι, η κλάση String εμπεριέχει μία μέθοδο *equals*() που ελέγχει κάθε χαρακτήρα της συμβολοσειράς του αντικειμένου και επιστρέφει true όταν έχουμε ακριβώς την ίδια συμβολοσειρά.

Η ανάθεση μνήμης στην Java είναι αυτόματη και δυναμική. Οταν δημιουργείται ένα νέο αντικείμενο η Java αυτόματα αναθέτει τον κατάλληλο χώρο μνήμης για το συγκεκριμένο αντικείμενο. Οταν λήξει η "ζωή" ενός αντικειμένου η μνήμη που είχε δεσμευτεί ελευθερώνεται επίσης αυτόματα από την Java. Στη συνέχεια, η Java διαθέτει ένα συλλογέα απορριμμάτων"

(garbage collector) ο οποίος ελευθερώνει τη μνήμη που κατείχαν όλα τα αντικείμενα που δε χρησιμοποιούνται πλέον.

### 3.4.3. Κλήσεις μεταβλητών και μεθόδων

***ΟΙ ΜΕΤΑΒΛΗΤΕΣ***

Το αντικείμενο που δημιουργήθηκε διαθέτει ένα σύνολο στιγμιαίων μεταβλητών και μεταβλητών κλάσης. Για να γίνει αναφορά στην τιμή της κάθε μεταβλητής δίνεται το όνομα του αντικειμένου ακολουθούμενο από την τελεία (.) και το όνομα της μεταβλητής. Για παράδειγμα για το αντικείμενο myComputer που διαθέτει τη μεταβλητή price, γίνεται αναφορά στην μεταβλητή με τη δήλωση myComputer.price.

Στο παράδειγμα που ακολουθεί δημιουργείται ένα αντικείμενο τύπου Point που περιέχεται στο πακέτο java.awt και δίνει τις (x,y) συντεταγμένες ενός σημείου :

| *ΠΡΟΓΡΑΜΜΑ* | *ΑΠΟΤΕΛΕΣΜΑΤΑ* |
|---|---|
| ……..<br>import java.awt.Point<br><br>class checkPoint {<br>  public static void main (String args[]) {<br>  Point simeio = new Point(15, 20);<br><br>  System.out.println(“Σημείο :”+ simeio.x+”,”+ simeio.y);<br><br>  simeio.x = 20;<br>  simeio.y = 15;<br>  System.out.println(“Σημείο :”+ simeio.x+”,”+ simeio.y);<br><br>  }<br>};<br>…… | Σημείο : 15,20<br>Σημείο : 20,15 |

Κάθε νέα στιγμή της κλάσης διαθέτει και μία τιμή δική της για κάθε στιγμιαία μεταβλητή. Αντίθετα, υπάρχει μόνο μία τιμή για κάθε μεταβλητή κλάσης κλάσης και κάθε στιγμή ύπαρξης της κλάσης αναφέρεται στην συγκεκριμένη αυτή τιμή. Μία μεταβλητή κλάσης δηλώνεται με τη λέξη κλειδί static. Εστω για παράδειγμα, ότι έχουμε την κλάση :

```
class studentDpt {
      static String department = “ΠΛΗΡΟΦΟΡΙΚΗ”΄
      String name;
      float gpa;
      …
}
```

Κάθε στιγμή ύπαρξης της κλάσης *studentDpt* έχει τις δικές τιμές για τις μεταβλητές *name* και *gpa* αλλά η μεταβλητή κλάσης *department* έχει την ίδια τιμή για κάθε στιγμή ύπαρξης της κλάσης.

## ***ΟΙ ΜΕΘΟΔΟΙ***

Οι κλήσεις των μεθόδων γίνονται όπως και στις μεταβλητές, δηλαδή με τη χρήση της τελείας (.). Η γενική μορφή κλήσης μίας μεθόδου είναι :

*αντικείμενο.μέθοδος(παράμετρος 1, παράμετρος 2, παράμετρος 3)*

Το όνομα κάθε μεθόδου ακολουθείται από παρενθέσεις, ακόμα και για τις περιπτώσεις που δεν υπάρχουν παράμετροι σε μία μέθοδο. Υπάρχει περίπτωση να γίνεται κλήση μεθόδου που υπάρχει σε μία άλλη μέθοδο :

*αντικείμενο.μέθοδος1().μέθοδος2();*

ενώ μπορεί να γίνει κλήση μεθόδου που εμπεριέχεται σε μία στιγμιαία μεταβλητή :

*αντικείμενο.μεταβλητή.μέθοδος();*

Για παράδειγμα, η κλάση System (περιέχεται στο java.lang πακέτο) διαθέτει τη μεταβλητή κλάσης System.out που αποτελεί μία στιγμή ύπαρξης της κλάσης PrintStream που αφορά στην καθορισμένη έξοδο του συστήματος και διαθέτει τη μέθοδο println για την εκτύπωση συμβολοσειρών στην έξοδο. Ετσι, προκύπτει η κλήση

*System.out.println()*

που έχει ήδη χρησιμοποιηθεί σε όλα τα παραδείγματα προγραμμάτων για εκτύπωση.

Στο παράδειγμα που ακολουθεί, γίνεται χρήση των διαφόρων μεθόδων που διαθέτει η Java για τη διαχείριση των συμβολοσειρών και που ορίζονται στην κλάση String. Οι μέθοδοι καλούνται με τη δήλωση ενός νέου αντικειμένου τύπου String :

| *ΠΡΟΓΡΑΜΜΑ* | *ΑΠΟΤΕΛΕΣΜΑΤΑ* |
|---|---|
| ……..<br>class checkString {<br>public static void main (String args[]) {<br>String keim="Αρχίζουν οι εγγραφές";<br><br>System.out.println("Μήκος κειμένου:"+ keim.length()); | Μήκος κειμένου:20<br>4ος χαρακτήρας: ί<br>ΑΡΧΙΖΟΥΝ ΟΙ ΕΓΓΡΑΦΕΣ |

```
    System.out.println("4ος χαρακτήρας:"+ keim.charAt(4));
    System.out.println("Κεφαλαία:"+ keim.toUpperCase());
  }
};
……
```

Οι μέθοδοι κλάσης (όπως και οι μεταβλητές κλάσης) αφορούν συνολικά σε όλες τις στιγμές ύπαρξης μίας κλάσης. Οι μέθοδοι κλάσης χρησιμοποιούνται κυρίως για γενικευμένου τύπου ενέργειες που αναφέρονται στη συνολική εννοιολογική υποστήριξη μίας κλάσης.

### 3.4.4. Αναφορές αντικειμένων

Οταν ανατίθενται αντικείμενα σε μεταβλητές ή χρησιμοποιούνται τα αντικείμενα ως παράμετροι σε μεθόδους, στην πραγματικότητα δε χρησιμοποιούνται τα αντικείμενα αυτά καθεαυτά ή αντίγραφα τους, αλλά χρησιμοποιούνται αναφορές των αντικειμένων. Το παρακάτω παράδειγμα είναι ενδεικτικό :

| *ΠΡΟΓΡΑΜΜΑ* | *ΑΠΟΤΕΛΕΣΜΑΤΑ* |
|---|---|
| ……..<br>import java.awt.Point<br><br>class checkRefer {<br>  public static void main (String args[]) {<br>  Point sim1, sim2;<br>  sim1 = new Point(15, 20);<br>  sim2 = sim1;<br><br>  sim1.x = 10;<br>  sim1.y = 10;<br>  System.out.println("Σημείο 1:"+ sim1.x+","+ sim1.y);<br>  System.out.println("Σημείο 2:"+ sim2.x+","+ sim2.y);<br><br>  }<br>};<br>…… | Σημείο 1: 10,10<br>Σημείο 2: 10,10 |

Στο παραπάνω παράδειγμα παρατηρούμε ότι η τιμή της μεταβλητής sim2 έχει αλλάξει (σύμφωνα με τη μεταβλητή sim1). Με την ανάθεση *sim2 = sim1;* δημιουργείται μία αναφορά από την sim2 προς το ίδιο αντικείμενο στο οποίο αναφέρεται η sim1.

Είναι χαρακτηριστικό ότι στην Java δεν υπάρχουν δείκτες(pointers), όπως σε άλλες γλώσσες προγραμματισμού. Η Java διαθέτει μόνο αναφορές. Με τη

χρήση αναφορών και πινάκων καλύπτονται οι ανάγκες υποστήριξης των δεικτών.

### 3.4.5. Προσαρμογή αντικειμένων και πρωταρχικών τύπων

Η Java υποστηρίζει ένα μηχανισμό μετατροπής μίας τιμής από έναν τύπο σε έναν άλλο τύπο. Ο μηχανισμός αυτός (casting) προσαρμόζει μία συγκεκριμένη τιμή ενός αντικειμένου ή ενός πρωταρχικού τύπου και δεν τροποποιεί την αρχική τιμή ή το αρχικό αντικείμενο. Η προσαρμογή έχει τη μορφή :

*(όνομα τύπου) τιμή*

Υπάρχουν τρία είδη προσαρμογών :

- Προσαρμογή μεταξύ πρωταρχικών τύπων : Κυρίως αφορά στις αριθμητικές τιμές. Οι boolean τιμές δεν μπορούν ναλληλο-προσαρμοσθούν με κανένα άλλο πρωταρχικό τύπο. Οπως περιγράφηκε η προσαρμογή έχει μεγαλύτερη προτεραιότητα από τις αριθμητικές πράξεις, επομένως πρέπει να υπάρχει παρένθεση όταν αναφερόμαστε σε μία μαθηματική έκφραση : (int) (a / b); Οταν χρειάζεται να προσαρμοστεί ένας τύπος σε ένα μεγαλύτερο του, υπάρχει αυτόματη μετατροπή (π.χ. προσαρμογή του int σε long, του int σε float).

- Προσαρμογή αντικειμένων : Κλάσεις μπορούν να προσαρμοστούν σε στιγμές ύπαρξης άλλων κλάσεων με τον περιορισμό ότι θα πρέπει οι κλάσεις να συνδέονται ιεραρχικά. Επομένως, ένα αντικείμενο μπορεί να προσαρμοστεί στη στιγμή ύπαρξης μίας υπο-κλάσσης ή της υπερ-κλάσης του και όχι σε οποιαδήποτε τυχαία κλάση. Ο γενικός τύπος είναι :
  *(όνομα κλάσης) αντικείμενο*

- Αλληλο-Προσαρμογή αντικειμένων - πρωταρχικών τύπων : Οι πρωταρχικοί τύποι και τα αντικείμενα αποτελούν διαφορετικά στοιχεία στην Java και δεν υπάρχει διαδικασία αυτόματης προσαρμογής μεταξύ τους. Το πακέτο java.lang περιέχει έναν αριθμό ειδικών κλάσεων που αντιστοιχεί σε κάθε πρωταρχικό τύπο (Integer για το int, Float για float, Boolean για τον boolean κλπ). Με χρήση των μεθόδων που ορίζονται σε αυτές τις ειδικές κλάσεις και τη λέξη κλειδί new, υπάρχει δυνατότητα δημιουργίας αντίστοιχων των αντικειμένων για κάθε πρωταρχικό τύπο.

## 3.5. Η κλάση Βιβλιοθήκης της Java

Η Java περιλαμβάνει βιβλιοθήκη κλάσεων που εμπεριέχει ένα σύνολο κλάσεων που προσαρμόζονται σε κάθε Java περιβάλλον. Οι κλάσεις αυτές περιλαμβάνονται στο πακέτο JDK. Το JDK περιλαμβάνει τεκμηρίωση για τη βιβλιοθήκη κλάσεων της Java με περιγραφή κάθε κλάσης, των μεταβλητών της, των μεθόδων της, των ενδιάμεσων της κλπ. Η

τεκμηρίωση αυτή είναι ιδιαίτερα χρήσιμη και ενδεικτική του τρόπου με τον οποίο λειτουργεί και αναπτύσσεται το προγραμματιστικό περιβάλλον της Java.

Τα πακέτα που περιλαμβάνονται στην βιβλιοθήκη κλάσεων της Java είναι :

| *ΠΑΚΕΤΟ* | *ΠΕΡΙΕΧΟΜΕΝΟ* |
|---|---|
| *java.lang* | κλάσεις που αφορούν στο βασικό κορμό της γλώσσας. Περιέχει τις κλάσεις Object, String, System καθώς και ειδικές κλάσεις για τους πρωταρχικούς τύπους(Integer,Character,Float κλπ) |
| *java.util* | κλάσεις που αφορούν σε λειτουργίες, (για π.χ. Date) καθώς και συλλογές απλούστερων κλάσεων |
| *java.io* | κλάσεις εισόδου-εξόδου για ανάγνωση και εγγραφή καθώς και για διαχείριση αρχείων. |
| *Java.net* | κλάσεις για την υποστήριξη δικτυακών λειτουργιών, όπως Socket, URL. |
| *Java.awt* | Το Ιδεατό Πακέτο Εργαλείων (AWT : *A*bstract *W*indow *T*oolkit) που περιλαμβάνει κλάσεις για τη διαχείριση βασικών παραθυρικών εφαρμογών, όπως Window, Button, Font, CheckBox κλπ. Ακόμα, το πακέτο αυτό, περιλαμβάνει κλάσεις που αφορούν στην επεξεργασία εικόνων (πακέτο java.awt.Image) |
| *java.applet* | κλάσεις που υλοποιούν τις μικρο-εφαρμογές της Java, με κύρια την κλάση Applet. |

Το περιβάλλον ανάπτυξης μπορεί να περιέχει και ένα σύνολο επιπλέον κλάσεων που προσθέτουν λειτουργικότητα και επιπλέον εργαλεία ανάπτυξης. Οι κλάσεις αυτές μπορεί να έχουν αναπτυχθεί από ποικίλους φορείς και να διατίθενται μέσω του δικτύου, χωρίς όμως να υπάρχει πλήρης υποστήριξη και εγγύηση για τη σωστή τους εφαρμογή και προσαρμοστικότητα. Μόνο οι κλάσεις που περιλαμβάνονται στο *java* πακέτο είναι αξιόπιστες σε ότι αφορά στην δια-πλατφορμική τους λειτουργία και εφαρμογή.

# 4. ΣΤΑΔΙΑ ΑΝΑΠΤΥΞΗΣ ΕΦΑΡΜΟΓΩΝ & ΜΙΚΡΟ-ΕΦΑΡΜΟΓΩΝ

## 4.1. Δημιουργία κλάσεων

Οπως περιγράφηκε στο 1ο Κεφάλαιο η δήλωση μίας κλάσης είναι απλή διαδικασία. Γενικά, υπάρχουν τρεις περιπτώσεις δήλωσης μίας κλάσης :

- Η γενική δήλωση μίας νέας κλάσης :
  class *Ονομα-Κλάσσης* {
  …
  }
- Η δήλωση μίας νέας κλάσης που αποτελεί υποκλάση μίας υπάρχουσας κλάσης :
  class *Ονομα-Νέας-Κλάσσης* extends *Ονομα-Υπερ-Κλάσσης* {
  …
  }
- Η δήλωση μίας νέας κλάσης που υλοποιεί ένα συγκεκριμένο ενδιάμεσο (interface) :
  class *ΟνομαΚλάσης* implements *Ενδιάμεσο* {
  …
  }

Στο σώμα της δήλωσης κάθε κλάσης περιλαμβάνονται οι δηλώσεις των μεταβλητών της κλάσης και των μεθόδων της.

### 4.1.1. Δήλωση μεταβλητών

Οπως αναφέρθηκε στην Παράγραφο 3.1, υπάρχουν τρία είδη μεταβλητών : οι στιγμιαίες μεταβλητές, οι μεταβλητές κλάσεις και οι τοπικές μεταβλητές.

Οι μεταβλητές που ορίζονται εκτός των ορισμών των μεθόδων λέγονται στιγμιαίες μεταβλητές (instance variables) και δηλώνονται όπως και οι τοπικές μεταβλητές. Η διαφορά των τοπικών και των στιγμιαίων μεταβλητών είναι η θέση τους στο σώμα δήλωσης της κλάσης.

Οι σταθερές δηλώνονται με χρήση της λέξης κλειδί *final* (π.χ. final int arithmos = 1000;) και είναι ιδιαίτερα χρήσιμες για να κατονομάζονται οι διάφορες καταστάσεις ενός αντικειμένου και για να υπάρχει ενιαίος έλεγχος των καταστάσεων αυτών.

Οι σταθερές που δηλώνονται με τη λέξη κλειδί *final* αφορούν μόνο σε στιγμιαίες μεταβλητές ή σε μεταβλητές κλάσης και δεν μπορούν να αφορούν

σε τοπικές μεταβλητές. Στη συνέχεια, δίνεται ένα παράδειγμα δήλωσης σταθερών και χρήσης τους σε μία εντολή :

| *ΔΗΛΩΣΗ ΣΤΑΘΕΡΩΝ* | *ΕΝΤΟΛΗ ΜΕ ΧΡΗΣΗ ΣΤΑΘΕΡΩΝ* |
|---|---|
| final int PRWTOS = 1;<br><br>final int DEUTEROS = 2;<br><br>final int TRITOS = 3;<br><br>int seira; | switch(seira) {<br>case PRWTOS : // εντολές για πρώτο<br>…<br>break;<br>case DEUTEROS : // εντολές για δεύτερο<br>…<br>break;<br>case TRITOS : // εντολές για τρίτο<br>…<br>break;<br><br>} |

Οι μεταβλητές κλάσης αναφέρονται συνολικά στην κλάση και σε κάθε στιγμή ύπαρξης της κλάσης. Οι μεταβλητές κλάσης είναι ιδιαίτερα χρήσιμες για την επικοινωνία μεταξύ των διαφόρων αντικειμένων του ίδιου τύπου κλάσης και δηλώνονται με τη λέξη κλειδί *static* όπως για παράδειγμα:

```
static int athroisma;                  // δήλωση μίας μεταβλητής κλάσης

static final int megistos = 100;       // δήλωση μίας σταθερής κλάσης
```

### 4.1.2. Δήλωση - Κλήση μεθόδων

Οπως αναφέρθηκε, οι μέθοδοι καθορίζουν τη συμπεριφορά ενός αντικειμένου, δηλαδή το περιεχόμενο των ενεργειών δημιουργίας του αντικειμένου και των διαδικασιών που συνοδεύουν τη διάρκεια ζωής του..

Η δήλωση των μεθόδων περιλαμβάνουν τέσσερα βασικά μέρη :

1. Τον τύπο του αντικειμένου ή τον πρωταρχικό τύπο που "επιστρέφει" η μέθοδος
2. Το όνομα της μεθόδου
3. Κατάσταση των παραμέτρων
4. Το κυρίως σώμα της μεθόδου.

Επομένως, η δήλωση της μεθόδου ακολουθεί τη γενική σύνταξη :

*τύπος-επιστροφής ΟνομαΜεθόδου* (τύπος παραμέτρου1, τύπος παραμέτρου 2, …) {
…
}

όπου ο *τύπος-επιστροφής* είναι ο πρωταρχικός τύπος ή η κλάση της τιμής που προκύπτει από την εκτέλεση της μεθόδου.

Ο *τύπος-επιστροφής* μπορεί να είναι :

- ένας από τους πρωταρχικούς τύπους τιμών της Java
- όνομα υπάρχουσας κλάσης
- η λέξη κλειδί *void* για την περίπτωση που η μέθοδος δεν επιστρέφει κάποια τιμή

Οταν μία μέθοδος επιστρέφει έναν πίνακα, οι αγκύλες δήλωσης του πίνακα μπορούν να τοποθετηθούν είτε μετά τη δήλωση του *τύπου-επιστροφής*, είτε μετά την κατάσταση των παραμέτρων (π.χ. int [] methodP(int low, int up);).

ΣΗΜΕΙΩΣΗ : Η λέξη κλειδί *this* χρησιμοποιείται για αναφορά στο τρέχον αντικείμενο και μπορούμε να τη χρησιμοποιούμε σε οποιοδήποτε σημείο αναφοράς του αντικειμένου. Ετσι, η εντολή *this.x* αναφέρεται στη στιγμιαία μεταβλητή x του τρέχοντος αντικειμένου. Η λέξη κλειδί *this* αναφέρεται μόνο στην τρέχουσα στιγμή της κλάσης και επομένως, μπορεί να γίνεται χρήση του μόνο μέσα στο σώμα δήλωσης της κλάσης. Οι μέθοδοι κλάσης (δηλ.. όσες δηλώνονται με static δεν μπορούν να χρησιμοποιούν το this). Αντίστοιχα, η λέξη κλειδί super αναφέρεται στην υπερ-κλάση της συγκεκριμένης κλάσης και χρησιμοποιείται όπως και το this.

- Η κλήση των μεθόδων περιλαμβάνει την παράθεση του ονόματος της μεθόδου, ακολουθούμενο από τις παραμέτρους της μεθόδου. Οι παράμετροι της μεθόδου είναι μεταβλητές οι οποίες περνούν στη μέθοδο με αναφορά, δηλαδή οποιαδήποτε αλλαγή επέρχεται σε αυτές στο εσωτερικό της μεθόδου επηρεάζει και τα αρχικά αντικείμενα. Στο παράδειγμα που ακολουθεί παρουσιάζεται η κλήση μεθόδων καθώς και η χρήση των παραμέτρων

Σε αντιστοιχία με τις μεταβλητές, υπάρχουν μέθοδοι κλάσης και στιγμιαίες μέθοδοι. Οι μέθοδοι κλάσης είναι διαθέσιμοι σε κάθε στιγμή ύπαρξης της κλάσης και μπορούνα διατεθούν σε άλλες κλάσεις. Η δήλωση των μεθόδων κλάσης γίνεται (όπως και για τις μεταβλητές κλάσης), με χρήση της λέξης κλειδί static πριν από τη δήλωση του ονόματος της μεθόδου. Οι μέθοδοι που αφορούν σε ένα συγκεκριμένο αντικείμενο ορίζονται ως στιγμιαίες μέθοδοι

και ενεργούν ή επηρεάζουν μόνο τη συγκεκριμένη στιγμή ύπαρξης ενός αντικειμένου.

Στο παράδειγμα που ακολουθεί δημιουργείται αρχικά η κλάση που περιλαμβάνει μία μέθοδο αντικατάστασης των μηδενικών στοιχείων ενός πίνακα σε μονάδες. Στη συνέχεια δίνεται το κυρίως πρόγραμμα που καλεί τη μέθοδο της κλάσης για μία συγκεκριμένη περίπτωση ενός πίνακα τριών μηδενικών στοιχείων.

*Δήλωση κλάσης*

```
class klisiM {
    int zerotoOne(int arg[ ]) {
        int athr = 0;

        for (int i=0;i< arg.lenght; i++) {
            if (arg[i] = = 0) {
                athr++;
                arg[i] = 1;
            }
        }
        return athr;
    }
}
```

| *ΠΡΟΓΡΑΜΜΑ* | *ΑΠΟΤΕΛΕΣΜΑΤΑ* |
|---|---|
| public static void main (String args[]) {<br>int miden;<br>int pinakas[ ] = {0, 1, 2, 3, 7, 0, 0};<br>klisiM dokimi = new klisiM();<br><br>System.out.print("Ο πίνακας είναι : [ ");<br>for (int i=0;i< pinakas.lenght; i++) {<br>System.out.print(pinakas[i]+" ");<br>}<br>System.out.println("] ");<br><br>miden = dokimi. ZerotoOne(pinakas);<br>System.out.println("Μηδενικά :"+miden);<br><br>System.out.print("Ο νέος πίνακας είναι : [ ");<br>for (int i=0;i< pinakas.lenght; i++) {<br>System.out.print(pinakas[i]+" ");<br>} | Ο πίνακας είναι : [0 1 2 3 7 0 0 ]<br><br>Μηδενικά : 3<br><br>Ο νέος πίνακας είναι : [1 1 2 3 7 1 1 ] |

```
        System.out.println(“] ”);
}
```

### 4.1.3. Ορια ενέργειας μεταβλητών - μεθόδων

Οταν γίνεται αναφορά σε μία μεταβλητή που περιλαμβάνεται στους ορισμούς των μεθόδων του προγράμματος μας η Java αναζητά τη δήλωση της μεταβλητής με την παρακάτω σειρά :

- Αρχικά αναζήτηση στο τρέχον κομμάτι κώδικα (π.χ. στο block εντολών στο οποίο βρισκόμαστε). Εάν δεν υπάρχει εκεί,
- συνεχίζει την αναζήτηση στο ανώτερο επίπεδο (εκτός του block εντολών) της τρέχουσας μεθόδου. Εάν δεν υπάρχει εκεί,
- τότε η μεταβλητή δεν είναι τοπική μεταβλητή, επομένως αναζητά τον ορισμό της ως μεταβλητή ύπαρξης ή μεταβλητή κλάσης στην τρέχουσα κλάση. Εάν δεν υπάρχει εκεί,
- συνεχίζει την αναζήτηση ανοδικά στο δέντρο ιεραρχίας των κλάσεων, μέχρι να βρεθεί η συγκεκριμένη μεταβλητή σε κάποια υπερκλάση.

Είναι προφανές ότι η δήλωση μίας μεταβλητής που γίνεται σε κάποιο εσωτερικό σημείο του προγράμματος αποκρύπτει τον ορισμό της ίδιας μεταβλητής εάν αυτός έχει πραγματοποιηθεί σε εξωτερικό ή ανώτερο σημείο του προγράμματος. Ετσι, εάν έχουμε το παρακάτω απόσπασμα προγράμματος :

```
class oria {
    int dokimi = 5;
    void ektypwsi() {
        int dokimi = 100;
        System.out.println(“dokimi = “+ dokimi);
    }
}
```

Η εσωτερική δήλωση της μεταβλητής *dokimi* καλύπτει την εξωτερική της δήλωση κι έτσι, το αποτέλεσμα θα είναι να εκτυπωθεί η τιμή 100. Η χρήση του this.dokimi μπορεί να διακρίνει τη μεταβλητή ύπαρξης από την τοπική μεταβλητή.

Αντίστοιχα, μπορεί να δηλωθεί μία μέθοδος σε ένα εσωτερικό σημείο και να καλύψει τη δήλωση της μεθόδου που έχει το ίδιο όνομα, τύπο-επιστροφής και λίστα παραμέτρων με μία μέθοδο που έχει δηλωθεί σε εξωτερικό σημείο του κώδικα του προγράμματος. Συνήθως υπάρχουν δύο κύριοι λόγοι για τη δήλωση ταυτόσημων μεθόδων σε διαφορετικά σημεία στην ιεραρχία των κλάσεων του κώδικα του προγράμματος :

- πλήρης αντικατάσταση της αρχικής δήλωσης μίας μεθόδου
- επέκταση και προσθήκη χαρακτηριστικών στην αρχική μέθοδο.

### 4.1.4. Υπερ-φόρτωση μεθόδων

Η χρήση ίδιου ονόματος για διαφορετική μέθοδο δεν επιτρέπεται στις συμβατικές γλώσσες προγραμματισμού. Στην Java όμως, μπορούμε να έχουμε δύο μεθόδους με το ίδιο όνομα αλλά πάντοτε θα πρέπει να έχουν είτε διαφορετικό τύπο επιστροφής αποτελεσμάτων, είτε διαφορετική κατάσταση παραμέτρων. Η χρήση του ίδιου ονόματος για διαφορετική μέθοδο στην Java καλείται "υπερ-φόρτωση" (overloading).

Για να δημιουργηθεί μία "υπερ-φορτωμένη" μέθοδος χρειάζεται να δηλωθούν στην ίδια κλάση διάφορες μέθοδοι που έχουν ακριβώς το ίδιο όνομα αλλά διαφορετική λίστα παραμέτρων. Η διαφορά στις παραμέτρους σημαίνει διαφορετικό τύπο παραμέτρων ή διαφορετικό αριθμό παραμέτρων. Η Java επιτρέπει την "υπερ-φόρτωση" μεθόδων όταν η μέθοδοι ίδιου ονόματος έχουν μοναδική και ξεχωριστή ακολουθία παραμέτρων και όχι διαφορετικό τύπο επιστροφής. Στο επόμενο παράδειγμα καταγράφεται μία γενική περίπτωση "υπερ-φόρτωσης" μεθόδων :

| *ΥΠΕΡΦΟΡΤΩΣΗ ΜΕΘΟΔΩΝ* | *ΚΛΗΣΗ* |
|---|---|
| …<br>class hyperLoad {<br>double foros;<br><br>// (1)<br>double ypolforos() {<br>System.out.println("Ο φόρος είναι : "+ foros);<br>return foros;<br>}<br><br>// (2)<br>double ypolforos(int newF) {<br>if (newF > 12.5) {<br>foros *= newF;<br>System.out.println("Ο φόρος είναι : "+ foros);<br>}<br>return foros;<br>}<br>} | ….<br><br>HyperLoad eforia;<br><br>eforia.ypolforos;<br>// κλήση της (1)<br><br>eforia.ypolforos(14.8);<br>// κλήση της (2) |

### 4.1.5. Μέθοδοι κατασκευής - ολοκλήρωσης

- Η μέθοδος κατασκευής είναι μία ειδική μέθοδος που καθορίζει τον τρόπο με βάση τον οποίο γίνεται η αρχικοποίηση ενός αντικειμένου. Οι μέθοδοι κατασκευής δεν μπορούν να κληθούν όπως οι άλλες μέθοδοι, με χρήση του ονόματος τους και λίστα παραμέτρων. Οι μέθοδοι κατασκευής καλούνται αυτόματα από τη Java. Η Java ακολουθεί τα παρακάτω 3 βήματα, όταν χρησιμοποιούμε τη λέξη κλειδί *new* για να δημιουργήσουμε ένα αντικείμενο μίας κλάσης :

  1. Αναθέτει μνήμη για το αντικείμενο

  2. Εκκινεί τις στιγμιαίες μεταβλητές του αντικειμένου, δίνοντας τους είτε τις αρχικές τιμές που τους έχουν δοθεί, είτε τις εξ'ορισμού τιμές του τύπου τους (0 για όλους τους αριθμούς, null για τα αντικείμενα, false για τις boolean τύπου μεταβλητές και '\0' για τους χαρακτήρες)

  3. Καλεί τη (τις) μέθοδο (ους) κατασκευής της κλάσης

Οταν μία κλάση δε διαθέτει μέθοδο κατασκευής, δημιουργείται μεν το αντικείμενο αλλά πιθανόν να χρειαστεί στη συνέχεια είτε να δοθούν οι μεταβλητές ύπαρξης είτε να γίνουν κλήσεις και άλλων μεθόδων που θα πραγματοποιήσουν την πλήρη και σωστή έναρξη του αντικειμένου.

Οι μέθοδοι κατασκευής διαφέρουν σε σχέση με τις άλλες μεθόδους στα εξής σημεία :

- Οι μέθοδοι κατασκευής έχουν πάντοτε το ίδιο όνομα με το όνομα της κλάσης

- Οι μέθοδοι κατασκευής δεν έχουν τύπο επιστροφής

Στο παράδειγμα που ακολουθεί έχουμε τη δήλωση μίας κλάσης που αφορά φοιτητές και περιλαμβάνει μία μέθοδο κατασκευής :

| *ΠΡΟΓΡΑΜΜΑ* | *ΑΠΟΤΕΛΕΣΜΑΤΑ* |
|---|---|
| class foititis {<br>int aem;<br>String onoma;<br><br>foititis(int a; String o) {<br>aem = a;<br>onoma = o;<br>}<br><br>void ektypwsi() {<br>System.out.println("Αριθμός Μητρώου: "+aem);<br>System.out.println("Ονοματεπώνυμο: "+onoma); | Αριθμός Μητρώου: 123<br><br>Ονοματεπώνυμο:Παππάς Χρήστος<br><br>********<br><br>Αριθμός Μητρώου: 213<br><br>Ονοματεπώνυμο:Αγγέλου |

```
    }

public static void main (String args[]) {
    foititis f;

    f = new foititis(123, "Παππάς Χρήστος");
    f.ektypwsi();
    System.out.print("******** ");

    f = new foititis(213, "Αγγέλου Ειρήνη");
    f.ektypwsi();
    System.out.print("******** ");
    }
}
```

```
Ειρήνη

********
```

Στις μεθόδους κατασκευής υπάρχει και υπερ-φόρτωση, με τον ίδιο τρόπο όπως και στις άλλες μεθόδους.

- Σε αντίθεση με τη μέθοδο κατασκευής, υπάρχει η μέθοδος οριστικοποίησης (finalizer). Η μέθοδος οριστικοποίησης καλείται ακριβώς πριν από την της λήξη της "ζωής" του αντικειμένου και την ελευθέρωση της μνήμης από το συλλογέα αντικειμένων που δεν υφίστανται πλέον (garbage collector). Η μέθοδος οριστικοποίησης δηλώνεται με το όνομα *finalize()*. Η γενική κλάση των αντικειμένων Object περιλαμβάνει μία εξ'ορισμού finalize() η οποία στην ουσία δεν επιτελεί καμία ενέργεια. Επομένως, μπορούμε να δηλώσουμε μία διαφορετική finalize() μέθοδο που να καλύπτει την finalize() της Java. Η δήλωση της θα πρέπει να γίνεται ως εξής :

```
protected void finalize() {
…
}
```

και στο σώμα της μπορεί να περιλαμβάνει όλες τις ενέργειες τερματισμού που θα συνοδεύουν την παύση του αντικειμένου. Η finalize() μπορεί να κληθεί οποιαδήποτε στιγμή, ακολουθεί τη λογική χρήσης των κοινών μεθόδων

## 4.2. Ανάπτυξη εφαρμογής

Η ανάπτυξη μίας εφαρμογής στην Java περιλαμβάνει συνοπτικά τις παρακάτω ενέργειες :

1. Δημιουργία κλάσεων και αντικειμένων

2. Δήλωση στιγμιαίων μεταβλητών και μεθόδων καθώς και μεταβλητών και μεθόδων κλάσης

3.Ανάπτυξη της εφαρμογής με συνδυασμό των παραπάνω ενεργειών 1. και 2. σε ένα ενιαίο πρόγραμμα που να εκτελείται και να παράγει τα επιθυμητά αποτελέσματα.

Οπως αναφέρθηκε, οι εφαρμογές(applications) στην Java διαφέρουν από τις μικρο-εφαρμογές(applets) οι οποίες απαιτούν προγράμματα πλοήγησης (π.χ. Netscape, HotJava) για να εκτελεστούν. Μία Java εφαρμογή μπορεί να είναι μικρής ή μεγάλης έκτασης ανάλογα με το πρόβλημα που επιλύει και αντίστοιχα να περιλαμβάνει μία ή περισσότερες κλάσεις. Για να εκτελεστεί μία εφαρμογή στην Java απαιτείται η ύπαρξη τουλάχιστον μίας κλάσης που να περιλαμβάνει τη μέθοδο *main()* διότι η main() είναι η πρώτη μέθοδος που καλείται όταν εκτελείται και μεταφράζεται η Java κλάση της εφαρμογής (με χρήση του μεταγλωτιστή).

Το σώμα της main() περιλαμβάνει τον απαραίτητο κώδικα υποστήριξης της έναρξης και της εκτέλεσης της εφαρμογής, δηλαδή την ανάθεση αρχικών τιμών στις μεταβλητές ή τη δημιουργία στιγμών κάθε κλάσης που έχει δηλωθεί.

Η ανάθεση των παραμέτρων της εφαρμογής γίνεται με παράθεση της ακολουθίας των παραμέτρων αμέσως μετά το όνομα της εφαρμογής. Ετσι για παράδειγμα, η εντολή :

*java programma param1 2 param3*

έχει σαν αποτέλεσμα την ανάθεση των παραμέτρων *param1*, *2, param3* στις τρεις πρώτες θέσεις του πίνακα συμβολοσειρών args[ ] της μεθόδου main(). Σημειώνεται ότι ο πίνακας παραμέτρων στην Java δεν είναι ανάλογος της δήλωσης argv της γλώσσας C και του Unix. Συγκεκριμένα εδώ στη θέση args[0] έχουμε την ανάθεση της πρώτης παραμέτρου ενώ στην C στη θέση argv[0] έχουμε το όνομα του προγράμματος.

*Δήλωση μεθόδου main()*

public static void main (String args[ ]) { … }

| | |
|---|---|
| public | δηλώνει ότι η μέθοδος είναι διαθέσιμη σε άλλες κλάσεις και αντικείμενα. Η main() μέθοδος πρέπει πάντοτε να είναι public |
| static | δηλώνει ότι η main() αποτελεί μέθοδο κλάσης |
| void | δηλώνει ότι η main() δεν επιστρέφει τίποτα |
| main( ) | η main() επιτρέπεται να έχει μόνο μία παράμετρο, έναν πίνακα συμβολοσειράς. Η παράμετρος αυτή |

χρησιμοποιείται για το ‘‘πέρασμα’’ εντολών οι οποίες καθορίζουν τον τρόπο με τον οποίο θα εκτελεστεί μία εφαρμογή.

## 4.3. Ανάπτυξη μικρο-εφαρμογής

Οπως ήδη αναφέρθηκε, υπάρχουν βασικές διαφορές μεταξύ των εφαρμογών και των μικροεφαρμογών. Η βασική διαφορά τους έγκειται στο ότι οι εφαρμογές αποτελούν αυτοτελή προγράμματα που εκτελούνται μόνο με χρήση του μεταγλωτιστή της Java ενώ οι μικρο-εφαρμογές εκτελούνται μέσω ενός προγράμματος πλοήγησης του Παγκόσμιου Ιστού (World Wide Web browser) του Internet. Η δυνατότητα εκτέλεσης των Java μικρο-εφαρμογών μέσω προγραμμάτων πλοήγησης έχει το πλεονέκτημα της ενίσχυσης των προγραμμάτων με γραφικά και παραθυρικά εργαλεία που διαθέτει το πρόγραμμα πλοήγησης. Ωστόσο, στις μικρο-εφαρμογές επιβάλλονται και συγκεκριμένοι περιορισμοί που κυρίως αφορούν στην ασφάλεια των συστημάτων που υποστηρίζουν τη δια-πλατφορμική εκτέλεση των μικρο-εφαρμογών. Οι περιορισμοί αυτοί περιλαμβάνουν τα εξής :

- Οι μικρο-εφαρμογές δεν μπορούν να κάνουν ανάγνωση ή εγγραφή στο σύστημα αρχείων του χρήστη που καλεί τη μικρο-εφαρμογή Υπάρχει μόνο η δυνατότητα ανάγνωσης-εγγραφής από συγκεκριμένους καταλόγους που έχουν καθοριστεί από το χρήστη.

- Συνήθως οι μικρο-εφαρμογές δεν μπορούν να επικοινωνούν με άλλον σύστημα εξυπηρέτη(server) εκτός του εξυπηρέτη στον οποίο αποθηκεύεται η μικρο-εφαρμογή.

- Οι μικρο-εφαρμογές δεν μπορούν να εκτελέσουν κανένα πρόγραμμα στο σύστημα του χρήστη.

- Οι μικρο-εφαρμογές δεν μπορούν να φορτώσουν εγγενή προγράμματα της τοπικής πλατφόρμας συστήματος στην οποία εκτελούνται, συμπεριλαμβανόμενων των διαμοιραζόμενων βιβλιοθηκών όπως για παράδειγμα οι DLL.

Ακόμα, η Java διαθέτει διάφορους τύπους ασφάλειας και συνέπειας του συστήματος, μέσω ειδικών ελέγχων του μεταφραστή και του μεταγλωτιστή, για να διασφαλίζει την ομαλή χρήση της γλώσσας.

Κατά τη δημιουργία μίας μικρο-εφαρμογής δημιουργείται πάντοτε μία υπο-κλάση της αρχικής κλάσης *Applet* που περιλαμβάνεται στο πακέτο *java.applet* της Java. Η κλάση Applet παρέχει όλα τα απαραίτητα εργαλεία που επιτρέπουν την εκτέλεση της μικρο-εφαρμογής μέσω των προγραμμάτων πλοήγησης. Τα εργαλεία αυτά περιλαμβάνουν παρέχονται κυρίως από την Αφαιρετική Εργαλειοθήκη Παραθύρων ( AWT : Abstract Window Toolkit)

και αφορούν στα στοιχεία ενδιάμεσων χρήστη (UI:User Interface), στις ενέργειες που θα εκτελεστούν μέσω του ποντικιού ή του πληκτρολογίου καθώς και στη δυνατότητα σχεδίασης στην οθόνη. Ετσι η αρχική κλάση μίας μικρο-εφαρμογής έχει πάντοτε την παρακάτω σύνταξη :

```
public class Ονομα-Κλάσης extends java.applet.Applet {
. . .
}
```

Πάντοτε η αρχική κλάση μίας μικρο-εφαρμογής πρέπει να δηλώνεται public. Οταν μία μικρο-εφαρμογή περιλαμβάνεται σε μία www σελίδα, η Java φορτώνει την αρχική κλάση της μικρο-εφαρμογής μέσω του δικτύου. Στις μικρο-εφαρμογές η Java δημιουργεί μία στιγμή ύπαρξης της αρχικής κλάσης μαζί με ένα σύνολο μεθόδων της κλάσης Applet (οι οποίες καλούνται από τη συγκεκριμένη στιγμή ύπαρξης της κλάσης της μικρο-εφαρμογής). Οταν διαφορετικές μικρο-εφαρμογές βασίζονται στην ίδια κλάση, δημιουργούνται ξεχωριστές στιγμές για καθεμία, οπότε κάθε μικρο-εφαρμογή μπορεί να συμπεριφέρεται διαφορετικά και ανεξάρτητα.

Κάθε μικρο-εφαρμογή έχει ένα συγκεκριμένο κύκλο ζωής και διαθέτει διαφορετικές λειτουργίες που αντιστοιχούν στα διάφορα γεγονότα και στάδια ανάπτυξης της. Κάθε λειτουργία της υλοποιείται με την κλήση κατάλληλης μεθόδου. Επομένως, όταν επιτελείται ένα γεγονός ή μία ενέργεια καλούνται οι κατάλληλες μέθοδοι μέσω του προγράμματος πλοήγησης. Οι κυριότερες μέθοδοι που αφορούν στην εκτέλεση μίας μικρο-εφαρμογής είναι :

- *ΑΡΧΙΚΟΠΟΙΗΣΗ :* πραγματοποιείται όταν η μικρο-εφαρμογή φορτώνεται για πρώτη φορά και περιλαμβάνει τη δημιουργία των αντικειμένων, τον καθορισμό της αρχικής κατάστασης της, τη φόρτωση εικόνων ή γραμματοσειρών καθώς και τη δήλωση των παραμέτρων της. Η αρχικοποίηση πραγματοποιείται με δήλωση της μεθόδου init() :

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

Σημειώνεται ότι η αρχικοποίηση εκτελείται μόνο μία φορά κατά τη διάρκεια της ζωής μίας μικρο-εφαρμογής.

- *ΕΝΑΡΞΗ :* Ακολουθεί την αρχικοποίηση ή συνεχίζει μία μικρο-εφαρμογή που είχε σταματήσει προηγουμένως. Η έναρξη μπορεί να κληθεί αρκετές φορές κατά τη διάρκεια ζωής μίας μικρο-εφαρμογής. Η έναρξη πραγματοποιείται με τη μέθοδο start()

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

- *ΤΕΡΜΑΤΙΣΜΟΣ* : Υπάρχει αλληλο-εξάρτηση μεταξύ της έναρξης και του τερματισμού μίας μικρο-εφαρμογής. Ο τερματισμός πραγματοποιείται είτε

όταν ο χρήστης αφήνει τη WWW σελίδα που περιέχει τη μικροεφαρμογή για να συνεχίσει την πλοήγηση σε άλλη σελίδα είτε όταν η μικροεφαρμογή περιλαμβάνει κλήση της μεθόδου stop().

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

Σημειώνεται ότι όταν ο χρήστης αφήνει τη σελίδα με τη μικρο-εφαρμογή για να συνεχίσει την πλοήγηση σε άλλη σελίδα, οι σπονδυλωτές προγραμματιστικές ενότητες (threads) συνεχίζουν να εκτελούνται.

- *ΕΚΚΑΘΑΡΙΣΗ* : Πραγματοποιεί την πλήρη εξάλειψη της μικρο-εφαρμογής δηλαδή καταργεί όλες τις μεθόδους της, τις σπονδυλωτές προγραμματιστικές ενότητες της καθώς και τα εκτελούμενα αντικείμενα της. Χρησιμοποιείται κυρίως όταν η μικρο-εφαρμογή πρέπει να ελευθερώσει πόρους του συστήματος που κυρίως προέρχονται από τις σπονδυλωτές προγραμματιστικές ενότητες. Η μέθοδος καταστροφής είναι η destroy() :

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

Σημειώνεται ότι η destroy() είναι διαφορετική από τη μέθοδο fiinalize() που περιγράφηκε στην παράγραφο 4.1.5. Η destroy() αφορά μόνο σε μικρο-εφαρμογές ενώ η fiinalize() είναι γενικότερου περιεχομένου και αφορά σε ένα συγκεκριμένο αντικείμενο που χρειάζεται να διαγραφεί.

- *ΣΧΕΔΙΑΣΗ* : περιλαμβάνει τις διαδικασίες μέσω των οποίων η μικρο-εφαρμογή σχεδιάζει κάτι στην οθόνη όπως κείμενο, μία γραμμή, ένα πλαίσιο, μία εικόνα κλπ. Η σχεδίαση πραγματοποιείται πολλές φορές κατά τη διάρκεια ζωής μίας μικρο-εφαρμογής και εκετελείται με τη μέθοδο paint()

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

Η paint() είναι διαφορετική από τις άλλες μεθόδους διότι έχει και παράμετρο κλήσης που είναι μία στιγμή της κλάσης Graphics. Η κλάση Graphics αποτελεί μέρος του πακέτου java.awt της Java και για να υπάρχει δυνατότητα χρήσης της κλάσης Graphics θα πρέπει να έχει γίνει "εισαγωγή" του πακέτου στο πρόγραμμα με την εντολή :

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

*ΠΑΡΑΔΕΙΓΜΑ :*

Στη συνέχεια παρουσιάζεται το πρόγραμμα "Γειά σου κόσμε" που είχε δοθεί στην Παράγραφο 1.5, με προσθήκη κώδικα για κάθε μία από τις παραπάνω μεθόδους που παρακολουθούν την εξέλιξη του κύκλου ζωής της μικρο-εφαρμογής.

*"ενισχυμένο" ΠΡΟΓΡΑΜΜΑ "Γειά σου κόσμε !"*

```
import java.awt.Graphics;
public class geiaKosme extends java.applet.Applet {
    Public void init() {
            System.out.println("Αρχικοποίηση ");
            resize(100, 100);
    }
    Public void paint (Graphics g) {
            g.drawString ("Γειά σου, κόσμε !", 5, 20);
    }
    Public void start() {
            System.out.println("Έκκίνηση ");
    }
    Public void stop() {
            System.out.println("Τερματισμός ");
    }
    Public void destroy() {
            System.out.println("Εεκκαθάριση ");
    }
}
```

Στο παραπάνω παράδειγμα παρουσιάζεται σχετικό μήνυμα κάθε φορά που καλείται η αντίστοιχη μέθοδος κατά τη διάρκεια ζωής της μικρο-εφαρμογής και κάθε φορά εκτυπώνεται η μέθοδος η οποία έχει κληθεί. Εάν η εκτέλεση πραγματοποιηθεί ομαλά, η μικρο-εφαρμογή καλεί τις μεθόδους σύμφωνα με την εξής σειρά : init(), start(), stop(), destroy().

Οπως περιγράφηκε στο 1ο Κεφάλαιο μετά τη δημιουργία της κλάσης που περιλαμβάνει τον παραπάνω κώδικα προγράμματος, καλείται ο μεταφραστής που δημιουργεί το αντίστοιχο .class αρχείο. Στη συνέχεια, πρέπει να συμπεριληφθεί το όνομα του .class αρχείου σε κάποια www σελίδα. Η γλώσσα HTML περιλαμβάνει την ειδική "ετικέτα" <applet>, για τη δήλωση

των μικρο-εφαρμογών στο περιεχόμενο των www σελίδων. Οταν το πρόγραμμα πλοήγησης συναντά την <applet> αναζητά το αντίστοιχο .class αρχείο και εκτελεί τη μικρο-εφαρμογή. Στη συνέχεια παραθέτουμε ένα απόσπασμα από μία www σελίδα που περιέχει τη δήλωση της παραπάνω μικρο-εφαρμογής geiaKosme :

*WWW ΣΕΛΙΔΑ*
*που περιέχει το ΠΡΟΓΡΑΜΜΑ "Γειά σου κόσμε !"*

```
<HTML>
<HEAD><TITLE> Γειά σου κόσμε !</ TITLE></HEAD>
  <BODY>
  <p> Η Java μικρο-εφαρμογή μας παρουσιάζει το μήνυμα :

  <APPLET CODE="geiaKosme.class" WIDTH=200 HEIGHT=50>

</ APPLET> </BODY></HTML>
```

- Στη σύνταξη της ετικέτας <APPLET> περιλαμβάνεται η λέξη-κλειδί CODE που ακολουθείται από το όνομα του .class αρχείου που περιέχει τη μικρο-εφαρμογή. Στην περίπτωση του παραπάνω παραδείγματος το .class αρχείο πρέπει να βρίσκεται στον ίδιο κατάλογο με το html αρχείο. Σε περίπτωση που τα αρχεία είναι αποθηκευμένα σε διαφορετικούς καταλόγους υπάρχει η λέξη-κλειδί CODEBASE η οποία ακολουθείται από την κατάλληλη διαδρομή ανεύρεσης των κλάσεων. Εάν για παράδειγμα, έχουμε αποθηκεύσει τις κλάσεις στον κατάλογο /claseis η αντίστοιχη δήλωση της <applet> θα ήταν :

```
<APPLET CODE="geiaKosme.class" CODEBASE=claseis WIDTH=200 HEIGHT=50>
```

- Οι παράμετροι WIDTH και HEIGHT καθορίζουν τα όρια του ορθογωνίου που περικλείει τη μικρο-εφαρμογή κατά την εκτέλεση της στην www σελίδα. Χρειάζεται προσοχή στη δήλωση των ορίων αυτών διότι σε κάποια προγράμματα πλοήγησης μπορεί να δημιουργηθεί πρόβλημα στην παρουσίαση των ενεργειών της.

- Οι ετικέτες <APPLET> και </APPLET> δεν αναγνωρίζονται από όλα τα προγράμματα πλοηγούς επομένως θα ήταν χρήσιμο να συμπεριλαμβάνουμε κάποιο κείμενο ενδεικτικό, περιγραφικό της μικρο-εφαρμογής ώστε ο χρήστης που δεν έχει Java συμβατό πρόγραμμα πλοηγό να πληροφορείται ότι στο σημείο αυτό της www σελίδας υπάρχει Java μικρο-εφαρμογή.

### 4.3.1. Χαρακτηριστικά της ετικέτας <APPLET>

Τα χαρακτηριστικά που μπορούν να δηλωθούν μέσα στα όρια δήλωσης της ετικέτας <APPLET> είναι σχεδόν ταυτόσημα με τα χαρακτηριστικά που δηλώνονται στην ετικέτα <IMG> της HTML. Οι κυριότερες επιλογές χαρακτηριστικών της είναι :

*ALIGN :* καθορίζει τον τρόπο τοποθέτησης της μικρο-εφαρμογής στη σελίδα. Υπάρχουν 9 επιλογές που αφορούν στην ALIGN :

1. ALIGN = LEFT, η μικρο-εφαρμογή τοποθετείται στο αριστερά περιθώριο της σελίδας και όλο το κείμενο που ακολουθεί τη δήλωση της μικρο-εφαρμογής διαχέεται δεξιά της. Το κείμενο εξακολουθεί να βρίσκεται στη θέση αυτή έως το τέλος της μικρο-εφαρμογής.

2. ALIGN = RIGHT, αντίθετα με την 1., γίνεται δεξιά τοποθέτηση της μικρο-εφαρμογής.

3. ALIGN = TOP, στοιχίζει τη μικρο-εφαρμογή σύμφωνα με το μέγιστο ύψος της συγκεκριμένης γραμμής

4. ALIGN = TEXTTOP, στοιχίζει την κορυφή της μικρο-εφαρμογής με το υψηλότερο σημείο κειμένου της γραμμής.

5. ALIGN = ABSMIDDLE, στοιχίζει το μέσον της μικρο-εφαρμογής σύμφωνα με το μέσο του μεγαλύτερου αντικειμένου της γραμμής

6. ALIGN = MIDDLE, στοιχίζει το μέσον της μικρο-εφαρμογής με το μέσο της γραμμής βάσης του κειμένου.

7. ALIGN = BOTTOM, στοιχίζει τη μικρο-εφαρμογή στη βάση του κειμένου.

8. ALIGN = ABSBOTTOM, στοιχίζει τη βάση της μικρο-εφαρμογής σύμφωνα με το χαμηλότερο αντικείμενο της γραμμής

9. ALIGN = BASELINE στοιχίζει το κάτω μέρος της μικρο-εφαρμογής σύμφωνα με τη γραμμή βάσης του κειμένου. Είναι η ίδια ακριβώς δήλωση με την BOTTOM.

*HSPACE και VSPACE :* δηλώσεις που καθορίζουν το χώρο (μετρήσιμο σε αριθμό pixels) που παρεμβάλλεται μεταξύ των ορίων της μικρο-εφαρμογής και του κειμένου που την περιβάλλει. Η HSPACE καθορίζει τον οριζόντιο χώρο(αριστερά και δεξιά της μικρο-εφαρμογής), ενώ η VSPACE τον κάθετο χώρο (επάνω και κάτω της).

*CODE και CODEBASE :* Οπως περιγράφηκε προηγουμένως, η CODE ακολουθείται από το όνομα της κλάσης δηλαδή του .class αρχείου που περιέχει τη μικρο-εφαρμογή ενώ η CODEBASE χρησιμοποιείται όταν χρειάζεται να δηλωθεί η διαδρομή ανεύρεσης των κλάσεων της μικρο-εφαρμογής.

### 4.3.2. Κλήση μικρο-εφαρμογών με χρήση παραμέτρων

Οπως περιγράφηκε στην Παράγραφο 4.2 η κλήση των εφαρμογών μπορεί να ακολουθείται από μία σειρά παραμέτρων. Στις μικρο-εφαρμογές χρειάζεται η ετικέτα <PARAM> (δηλώνεται μέσα στα όρια της <Applet>) για να γίνει το πέρασμα των παραμέτρων της μικρο-εφαρμογής. Γενικά, χρειάζονται δύο ενέργειες για να δηλωθούν και να ανατεθούν τιμές στις παραμέτρους μίας μικρο-εφαρμογής :

- Χρήση της ετικέτας παραμέτρων <PARAM> στο HTML αρχείο. Η <PARAM> πρέπει να περιέχεται μεταξύ των <Applet> και </Applet> και αποτελείται από το όνομα της μεταβλητής και την τιμή της. Για παράδειγμα η εντολή :

<APPLET CODE=”geiaSou.class” WIDTH=200 HEIGHT=50>

<PARAM NAME=onoma VALUE=”Δανάη”> Java Μικρο-εφαρμογή </APPLET>

καλεί τη μικρο-εφαρμογή *geiaSou* με την παράμετρο *onoma* που έχει την τιμή *Δανάη*.

- Ανάπτυξη κατάλληλου κώδικα στην μικρο-εφαρμογή που να διαχειρίζεται αυτές τις παραμέτρους. Οι παράμετροι με τις οποίες μπορεί να κληθεί μία μικρο-εφαρμογή δηλώνονται στον κώδικα της που αφορά στην αρχικοποίηση (δηλ. στη μέθοδο init()). Η μικρο-εφραμογή λαμβάνει τις τιμές της παραμέτρου μέσω της μεθόδου getParameter(). Ετσι η μικρο-εφαρμογή geiaSou που καλείται παραπάνω θα γίνει :

*ΠΡΟΓΡΑΜΜΑ “Γειά σου !”*

```
import java.awt.Graphics;
public class geiaSou extends java.applet.Applet {
String onoma;

    Public void init() {
        onoma = getParameter(“onoma”);
        onoma = “Γειά σου ”+onoma + “ !”;
    }

    Public void paint (Graphics g) {
```

```
        g.drawString (onoma, 5, 30);
    }
}
```

Η μέθοδος init() παίρνει την παράμετρο onoma και προσθέτει την τιμή της στην φράση "Γειά σου " και στη συνέχεια η paint() εμφανίζει τη νέα φράση στην οθόνη. Αρα, παραπάνω εντολή του html αρχείου :

```
<APPLET CODE="geiaSou.class" WIDTH=200 HEIGHT=50>
<PARAM NAME=onoma VALUE="Δανάη">  Java Μικρο-εφαρμογή </APPLET>
```

έχει ως αποτέλεσμα την εκτύπωση της φράσης :

*Γειά σου Δανάη !*

στην θέση που έχει η μικρο-εφαρμογή στην html σελίδα.

# 5. ΤΑ ΓΡΑΦΙΚΑ ΣΤΗΝ JAVA

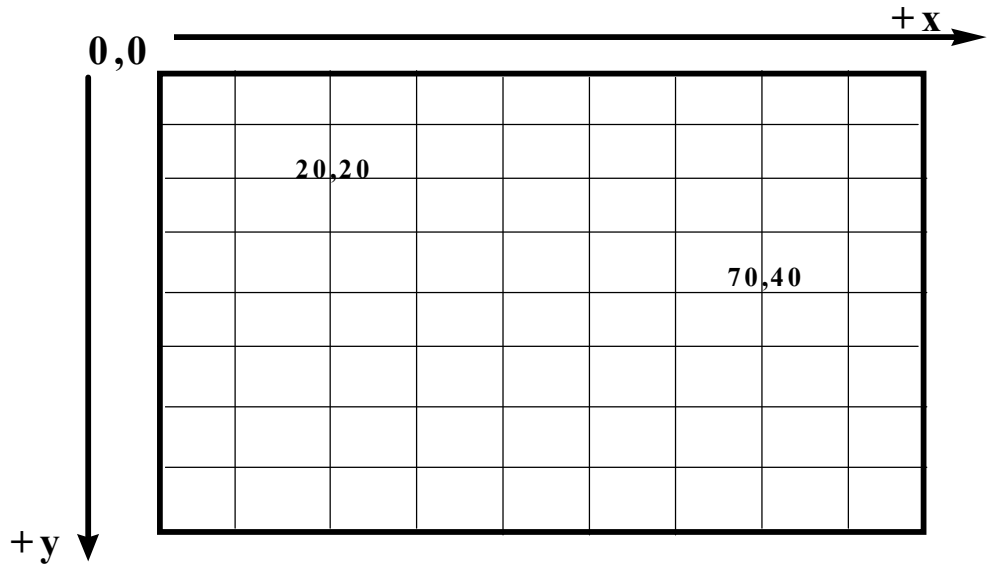
## 5.1. Η κλάση Graphics

Ενα από τα κυριότερα χαρακτηριστικά της Java είναι οι αυξημένες δυνατότητες της σε ό,τι αφορά στη χρήση και στη διαχείριση γραφικών και ειδικότερα στη σχεδίαση σχημάτων, γραμμών, εικόνων, κειμένου κλπ. Η Java διαθέτει την κλάση Graphics η οποία περιλαμβάνει το σύνολο των μεθόδων που υποστηρίζουν τη σχεδίαση και την ανάπτυξη γραφικών. Η κλάση Graphics αποτελεί μέρος του πακέτου java.awt και εάν μία μικρο-εφαρμογή περιέχει οποιοδήποτε είδος σχεδίασης θα πρέπει να δηλώνεται ως εξής :

```
import java.awt.Graphics;
public class Ονομα-Κλάσσης extends java.applet.Applet{
…
}
```

### 5.1.1. Σχεδίαση Σχημάτων

Για να σχεδιαστεί οτιδήποτε στην οθόνη θα πρέπει να κληθούν οι κατάλληλες μέθοδοι σχεδίασης της κλάσης Graphics. Ολες οι μέθοδοι σχεδίασης χρειάζονται παραμέτρους που να δηλώνουν τα σημεία που καθορίζουν το σχεδιαζόμενο αντικείμενο και οι παράμετροι αυτές βασίζονται στο σύστημα συντεταγμένων της μικρο-εαφρομγής. Η Java έχει την αρχή των αξόνων (0,0) στην επάνω αριστερά γωνία. Οι x θετικές συντεταγμένες είναι προς τα δεξιά, ενώ οι y θετικές συντεταγμένες είναι προς τα κάτω. Ολες οι τιμές των pixels είναι ακέραιες. Στο παρακάτω σχήμα παρουσιάζεται το σύστημα συντεταγμένων της Java :




Σε ότι αφορά στη σχεδίαση η Java έχει ενσωματωμένες πρωταρχικές μεθόδους σχεδίασης που περιλαμβάνουν τη σχεδίαση γραμμών, ορθογωνίων, πολυγώνων, ελλείψεων και τόξων

*ΣΧΕΔΙΑΣΗ ΓΡΑΜΜΩΝ :* Πραγματοποιείται με χρήση της μεθόδου drawLine() η σύνταξη της οποίας περιλαμβάνει 4 παραμέτρους : τις x,y συντεταγμένες για καθένα από τα δύο σημεία της ευθείας. Στο παρακάτω παράδειγμα θα σχεδιαστεί η ευθεία γραμμή που ενώνει το σημείο 20,20 με το σημείο 70,40 (τα δύο σημεία που παρουσιάζονται στο προηγούμενο σύστημα συντεταγμένων της Java) :

```
public void paint(Graphics g) {
        g.drawLine(20,20,70,40);
}
```

*ΣΧΕΔΙΑΣΗ ΠΑΡΑΛΛΗΛΟΓΡΑΜΜΩΝ:* Υπάρχουν τρία ήδη παραλληλόγραμμων : απλά, κυκλικών γωνιών και 3-διάστατα. Για κάθε είδος παραλληλογράμμου υπάρχουν δύο μέθοδοι από τις οποίες κάνουμε επιλογή. Η μία αφορά τη σχεδίαση της περιμέτρου του παραλληλογράμμου ενώ η άλλη αφορά στην κάλυψη της επιφάνειας του με κάποιο χρώμα. Ετσι, στη σχεδίαση απλών παραλληλογράμμων υπάρχει η drawRect() για τη σχεδίαση της περιμέτρου και η fillRect() για την κάλυψη της επιφάνειας του. Και οι δύο μέθοδοι δέχονται 4 παραμέτρους, τις x,y συντεταγμένες του σημείου της επάνω αριστερά γωνίας και το πλάτος και ύψος του παραλληλογράμμου που θα σχεδιαστεί :

```
public void paint(Graphics g) {
        g.drawRect(20,20,70,40);
        g.fillRect(100,20,60,60);
}
```

Αντίστοιχα, για τα παραλληλόγραμμα κυκλικών γωνιών υπάρχουν οι αντίστοιχες μέθοδοι drawRoundRect() και fillRoundRect(). Οι μέθοδοι αυτές

έχουν δύο επιπλέον παραμέτρους που αφορούν στο πλάτος και στο ύψος των γωνιών των παραλληλογράμμων και καθορίζουν την απόσταση από την ακμή από την οποία θα αρχίσει το τόξο της γωνίας.

```
public void paint(Graphics g) {
        g.drawRoundRect(20,20,70,40,10,10);
        g.fillRoundRect(100,20,60,60,20,20);
}
```

Για τα 3-διάστατα παραλληλόγραμμα οι αντίστοιχες μέθοδοι είναι draw3DRect() και fill3DRect() οι οποίες έχουν τις παραμέτρους των drawRect() και η fillRect() των απλών παραλληλόγραμμων και επιπλέον μία παράμετρο boolean τύπου η οποία είναι true όταν έχουμε την 3η διάσταση με ανύψωση της περιμέτρου του παραλληλογράμμου και false όταν η 3η διάσταση γίνεται με της εις βάθος πρόοψη του παραλληλογράμμου.

```
public void paint(Graphics g) {
        g.draw3D Rect(20,20,70,40, true);
        g.fill3D Rect(100,20,60,60, false);
}
```

*ΣΧΕΔΙΑΣΗ ΠΟΛΥΓΩΝΩΝ:* Για να σχεδιαστεί ένα πολύγωνο χρειάζεται ένας αριθμός από x, y συντεταγμένες που καθορίζουν τα διαδοχικά σημεία που ενώνονται με ευθείες γραμμές για να δημιουργηθεί το πολύγωνο. Αντίστοιχα με τα παραλληλόγραμμα υπάρχει δυνατότητα σχεδίασης της περιμέτρου (μέθοδος drawPolygon()) και κάλυψης της επιφάνειας του (μέθοδος fillPolygon()). Υποστηρίζονται δύο τρόποι σχεδίασης ενός πολυγώνου :

- Τα δεδομένα είναι δύο πίνακες για τις x, y συντεταγμένες αντίστοιχα καθώς και ένας ακέραιος που εκφράζει το συνολικό αριθμό κορυφών του πολυγώνου :

```
public void paint(Graphics g) {
        int timesx = {40, 95, 99, 143, 54, 59, 27};
        int timesy = {34, 75, 37, 71, 109, 81, 107};
        int simeia = timesx.length;
        g.drawPolygon(timesx, timesy, simeia);
}
```

  Η Java δεν κλείνει την περίμετρο του πολυγώνου αυτόματα. Για να κλείσει το πολύγωνο θα πρέπει να δηλωθούν οι x, y συντεταγμένες του πρώτου σημείου στους αντίστοιχους πίνακες. Αντίθετα, η fillPolygon() σχεδιάζει την επιφάνεια ενός κλειστού πολυγώνου.

- Με χρήση της κλάσης Polygon δημιουργείται ένα αντικείμενο τύπου Polygon και στη συνέχεια καλούνται οι μέθοδοι drawPolygon() ή fillPolygon(). Η συνολική διαδικασία σχεδίασης του πολυγώνου περιλαμβάνει τη δημιουργία του αντικειμένου με χρήση των πινάκων των x, y συντεταγμένων. Στη συνέχεια επεκτείνουμε το πολύγωνο με

προσθήκη σημείων (με χρήση της μεθόδου addPoint() της κλάσης Polygon) :

```
public void paint(Graphics g) {
        int timesx = {40, 95, 99, 143, 54, 59, 27};
        int timesy = {34, 75, 37, 71, 109, 81, 107};
        int simeia = timesx.length;
        Polygon p = new Polygon(timesx, timesy, simeia);
        g.drawPolygon(p);
}
```

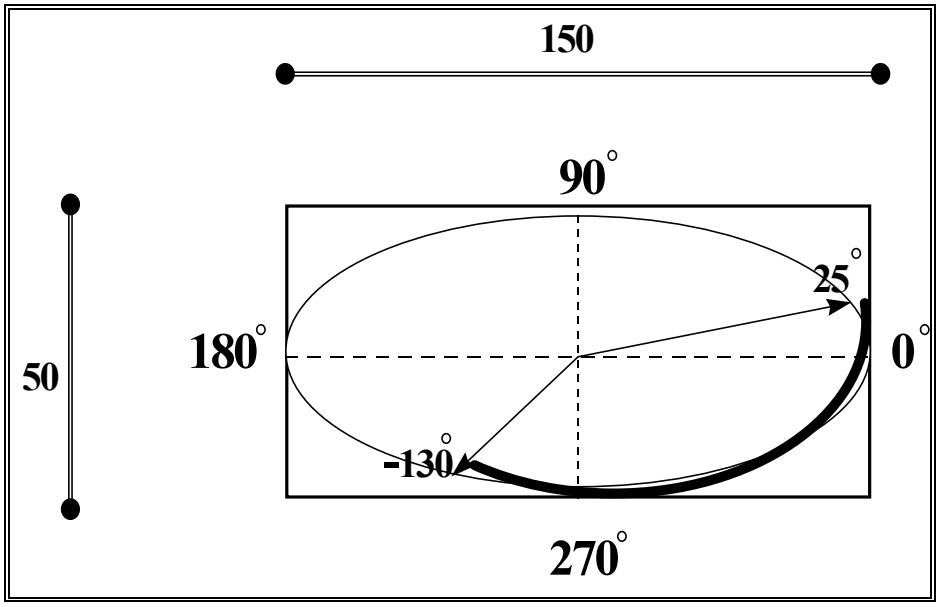
*ΣΧΕΔΙΑΣΗ ΕΛΛΕΙΨΕΩΝ-ΚΥΚΛΩΝ:* Αποτελούν παρόμοια περίπτωση με τα παραλληλόγραμμα με τη διαφορά ότι έχουν στρογγυλεμένες γωνίες. Χρειάζονται τέσσερις παράμετροι για να σχεδιαστούν : οι x,y συντεταγμένες του σημείου της γωνίας της κορυφής καθώς και το πλάτος και το ύψος του σχήματος. Υπάρχουν για τα σχήματα αυτά οι αντίστοιχες μέθοδοι drawOval() και fillOval() :

```
public void paint(Graphics g) {
        g.drawOval(20,20,40,40);
        g.fillOval(100,20,60,60);
}
```

*ΣΧΕΔΙΑΣΗ ΤΟΞΩΝ:* Τα τόξα παρουσιάζουν τη μεγαλύτερη δυσκολία σε ότι αφορά στη σχεδίαση σε σχέση με τα υπόλοιπα σχήματα. Το τόξο αποτελεί μέρος μίας έλλειψης ή ενός κύκλου και έτσι η μέθοδος drawArc() χρειάζεται έξι παραμέτρους : τις x,y συντεταγμένες του σημείου εκκίνησης, το πλάτος και το ύψος, τη γωνία κατά την οποία θα ξεκινήσει το τόξο και τον αριθμό των μοιρών που θα συνεχίσει να σχεδιάζει πριν τελειώσει το τόξο.

```
public void paint(Graphics g) {
        g.drawArc(10, 20, 150, 50, 25, -130);
}
```

Το αποτέλεσμα αυτής της μεθόδου είναι η κατασκευή του τόξου του παρακάτω σχήματος:




Γενικά, για την κατασκευή των τόξων πρέπει να ακολουθούνται τα παρακάτω βήματα :

1.Το τόξο θεωρείται ως κομμάτι ενός κύκλου ή μίας έλλειψης

2.Γίνεται κατασκευή του πλήρες κύκλου ή της έλλειψης με δεδομένα το αρχικό τους σημείο, το πλάτος και το ύψος τους

3.Καθορίζεται η γωνία εκκίνησης του τόξου

4.Καθορίζεται ο αριθμός των μοιρών και η κατεύθυνση μετατόπισης για τη σχεδίαση του επιθυμητού τόξου.

### 5.1.2. Κείμενο και Γραμματοσειρές

Η κλάση Graphics υποστηρίζει την εμφάνιση κειμένου στην οθόνη με χρήση της κλάσης Font (ή της κλάσης FontMetrics). Η κλάση Font περιλαμβάνεται στο πακέτο java.awt και πριν χρησιμοποιηθεί θα πρέπει να έχει γίνει εισαγωγή του αντίστοιχου πακέτου (import java.awt.Font).

Η κλάση Font αναπαριστά ένα δεδομένο τύπο γραμματοσειράς με χαρακτηριστικά στοιχεία για το όνομα, το στιλ και το μέγεθος της γραμματοσειράς. Η κλάση FontMetrics δίνει επιπλέον πληροφορίες για τη συγκεκριμένη γραμματοσειρά όπως για παράδειγμα ύψος και πλάτος των χαρακτήρων κλπ. Για να καταγραφεί το κείμενο στην οθόνη αρχικά πρέπει να δημιουργηθεί μία στιγμή ύπαρξης της κλάσης Font με δεδομένα :

- Το όνομα κάθε γραμματοσειράς αντιπροσωπεύει την κατηγορία στην οποία εντάσσεται π.χ. "TimesRoman", "Courier" ή "Helvetica". Οι γραμματοσειρές που μπορεί να χρησιμοποιηθούν σε μία εφαρμογή είναι αυτές που έχουν εγκατασταθεί στο υπάρχον σύστημα. Το πακέτο java.awt διαθέτει τη μέθοδο getFontList() η οποία εμφανίζει το σύνολο των γραμματοσειρών που είναι διαθέσιμες στο υπάρχον σύστημα.

- Το στιλ κάθε γραμματοσειράς χαρακτηρίζεται από τις σταθερές που διαθέτει η κλάση Font και είναι Font.PLAIN, Font.BOLD, Font.ITALIC.

- Το μέγεθος της γραμματοσειράς που μπορεί να είναι το ύψος των χαρακτήρων

Μετά τη δήλωση των χαρακτηριστικών της γραμματοσειράς χρειάζεται να κληθεί η μέθοδος *setFont*() η οποία αναθέτει τη γραμματοσειρά στο αντικείμενο που δημιουργήθηκε. Η εμφάνιση του κειμένου γίνεται με τη χρήση των μεθόδων *drawChars*() και *drawString*().

```
public void paint(Graphics g) {
    Font f = new Font("TimesRoman", Font.BOLD+ Font.ITALIC, 36);

    g. setFont(f);
    g.drawString("Νέα γραμματοσειρά", 20,40);
}
```

Στη μέθοδο drawString οι δύο τιμές που ακολουθούν το κείμενο δηλώνουν το σημείο της οθόνης από την οποία θα ξεκινήσει η εμφάνιση του κειμένου. Η μέθοδος drawChars() χρησιμοποιείται σπανιότερα και έχει ως παράμετρο έναν πίνακα χαρακτήρων, έναν ακέραιο που αντιπροσωπεύει τον πρώτο χαρακτήρα του πίνακα που θα εμφανιστεί, έναν ακόμα ακέραιο για τον τελευταίο χαρακτήρα του πίνακα που θα εμφανιστεί καθώς και τις x, y συντεταγμένες του σημείου εκκίνησης της εμφάνισης του κειμένου.

Στον παρακάτω πίνακα παρουσιάζονται οι κυριότερες μέθοδοι που περιλαμβάνονται στη κλάση Font.

| *Μέθοδος* | *Αντικείμενο* | *Ενέργεια* |
|---|---|---|
| getFont() | Graphics | Επιστρέφει το τρέχον αντικείμενο Font που είχε καθοριστεί με χρήση της setFont() |
| getName() | Font | Επιστρέφει το όνομα του τρέχοντος Font |
| getSize() | Font | Επιστρέφει το μέγεθος του τρέχοντος Font |
| getStyle() | Font | Επιστρέφει το στιλ του τρέχοντος Font |
| isPlain() | Font | Επιστρέφει true ή false ανάλογα με το εάν |

| | | |
|---|---|---|
| | | το στιλ του Font είναι απλό(plain) ή όχι. |
| isBold() | Font | Επιστρέφει true ή false ανάλογα με το εάν το στιλ του Font είναι έντονης γραφής (bold) ή όχι. |
| isItalic() | Font | Επιστρέφει true ή false ανάλογα με το εάν το στιλ του Font είναι πλάγιας γραφής (italic) ή όχι. |

- Οπως αναφέρθηκε, για τη συγκέντρωση ειδικότερων πληροφοριών σε ότι αφορά στη γραμματοσειρά χρησιμοποιείται η κλάση FontMetrics() της οποίας οι κυριότερες μέθοδοι είναι :

| *Μέθοδος* | *Ενέργεια* |
|---|---|
| stringWidth(string) | Επιστρέφει το πλήρες πλάτος του δεδομένου string, εκφραζόμενο σε αριθμό pixels. |
| CharWidth(char) | Επιστρέφει το πλάτος του δεδομένου χαρακτήρα |
| getAscent() | Επιστρέφει την απόσταση μεταξύ της γραμμής βάσης και της κορυφής των χαρακτήρων της γραμματοσειράς. |
| GetDescent() | Επιστρέφει την απόσταση μεταξύ της γραμμής βάσης και του βάθους των χαρακτήρων της γραμματοσειράς (π.χ. γράμμα p ή q). |
| getLeading() | Επιστρέφει την απόσταση μεταξύ δύο γραμμών κειμένου. |
| getHeight() | Επιστρέφει το ύψος του Font είναι έντονης γραφής (bold) ή όχι. |

### 5.1.3. Χρήση Χρωμάτων

Η Java παρέχει μεθόδους κατάλληλες για τη διαχείριση χρωμάτων μέσω της κλάσης Color που περιλαμβάνεται στο πακέτο java.awt. Η κλάση αυτή υποστηρίζει ακόμα και την εμφάνιση "φόντου" συγκεκριμένων χρωμάτων. Το χρωματικό μοντέλο της Java χρησιμοποιεί 24-bit χρώματα όπου κάθε χρώμαναπαριστάτε από μία τριάδα τιμών (RGB τιμή) ως συνδυασμός των τριών βασικών χρωμάτων κόκκινο(Red), πράσινο(Green) και μπλε(Blue). Κάθε ένας από τους τρεις αυτούς χρωματικούς παράγοντες μπορεί να έχει μία τιμή από 0 έως και 255. Ο συνδυασμός 0,0,0 είναι το μαύρο χρώμα και ο 255,255,255 το άσπρο. Για να σχεδιαστεί ένα αντικείμενο σε κάποιο

συγκεκριμένο χρώμα, αρχικά πρέπει να δημιουργηθεί μία στιγμή ύπαρξης της κλάσης Color. Η κλάσης Color περιλαμβάνει όλα τα συνηθισμένα χρώματα τα οποία παρουσιάζονται στον παρακάτω πίνακα :

| *Χρώμα* | *RGB τιμή* | *Χρώμα* | *RGB τιμή* |
|---|---|---|---|
| Color.white | 255,255,255 | Color.blue | 0, 0, 255 |
| Color.black | 0, 0, 0 | Color.yellow | 255, 255, 0 |
| Color.lightGray | 192, 192, 192 | Color.magenta | 255, 0, 255 |
| Color.gray | 128, 128, 128 | Color.cyan | 0, 255, 255 |
| Color.darkGray | 64, 64, 64 | Color.pink | 255, 175, 175 |
| Color.red | 255, 0, 0 | Color.orange | 255, 200, 0 |
| Color.green | 0, 255, 0 | | |

Εκτός από τα παραπάνω δεδομένα χρώματα που υποστηρίζει η Java υπάρχει δυνατότητα δημιουργίας οποιουδήποτε χρώματος όπως για παράδειγμα, με την εντολή :

```
Color chroma = new Color(134, 123, 200);
```

Για να σχεδιαστεί οποιοδήποτε αντικείμενο σε ένα συγκεκριμένο χρώμα θα πρέπει να δηλώσουμε το χρώμα για το αντικείμενο με χρήση της μεθόδου setColor() η οποία περιλαμβάνεται στην κλάση Graphics. Ακόμα υπάρχουν οι μέθοδοι setBackground() και setForeground() οι οποίες έχουν ως παράμετρο το χρώμα του "φόντου" και το χρώμα εργασίας της οθόνης.

```
public void paint(Graphics g) {
    g. setColor(Color.green);
    g.drawRect(10, 30, 25, 25);
}
```

## 5.2. Η κλάση Image

Η διαχείριση εικόνας στην Java είναι ιδιαίτερα απλή διαδικασία διότι η Java διαθέτει στις κλάσεις Applet και Graphics που περιλαμβάνουν τις κατάλληλες μεθόδους που αφορούν στη φόρτωση εικόνων καθώς και στην εμφάνιση τους μέσω των μικρο-εφαρμογών. Η κλάση Applet διαθέτει τη μέθοδο getImage()

η οποία φορτώνει μία εικόνα και αυτόματα δημιουργεί μία στιγμή ύπαρξης της κλάσης Image. Υπάρχουν δύο τρόποι κλήσης της μεθόδου getImage() :

- Κλήση της getImage() με παράμετρο ένα αντικείμενο τύπου URL, ώστε η μέθοδος να βρίσκει την εικόνα από το συγκεκριμένο URL .
- Κλήση της getImage() με παραμέτρους το αντικείμενο τύπου URL και μία συμβολοσειρά (string) που περιγράφει τη διαδρομή ή το όνομα της εικόνας που αναζητούμε στο συγκεκριμένο URL .

Εάν για παράδειγμα έχουμε την εικόνα computer.gif στο URL "http://www.csd.auth.gr" θα έχουμε την εντολή :

```
Image eikona = getImage(
                    new URL("http://www.csd.auth.gr/computer.gif"));
```

Η κλάση Applet διαθέτει ακόμα τις μεθόδους getDocumentBase() και getCodeBase() οι οποίες χρησιμεύουν στην ανεύρεση του αντικείμενο URL βάσης. Η getDocumentBase() χρησιμοποιείται για τις περιπτώσεις που οι εικόνες είναι σχετιζόμενες με τα html αρχεία ενώ η getCodeBase() χρησιμοποιείται για τις περιπτώσεις που οι εικόνες είναι σχετιζόμενες με τα java αρχεία.

- Η μέθοδος getDocumentBase() επιστρέφει ένα αντικείμενο τύπου URL που εκφράζει τον κατάλογο του HTML αρχείου που περιέχει τη μικρο-εφαρμογή. Στο παράδειγμα :
  Image eikona = getImage(getDocumentBase(), "computer.gif");
  το αρχείο computer.gif βρίσκεται στον ίδιο κατάλογο με τα html αρχεία.

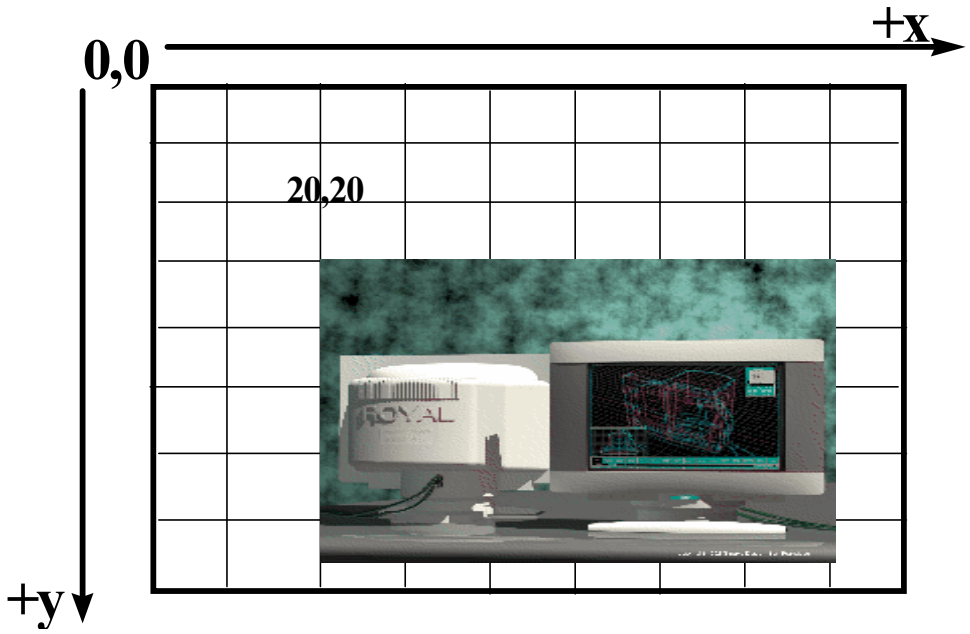
- Η μέθοδος getCodeBase()επιστρέφει μία συμβολοσειρά (string) που περιγράφει τον κατάλογο στον οποίο βρίσκεται η μικρο-εφαρμογή. Στο παράδειγμα :
  Image eikona = getImage(getCodeBase(), "computer.gif");
  το αρχείο computer.gif βρίσκεται στον ίδιο κατάλογο με τη μικρο-εφαρμογή.

Η Java δίνει τη δυνατότητα ρύθμισης και τοποθέτησης της εικόνας σε συγκεκριμένα σημεία στην οθόνη ή μέσα σε κατάλληλα πλαίσια μέσω της κλάσης Graphics που διαθέτει τη μέθοδο drawImage(). Για παράδειγμα, στο παρακάτω πρόγραμμα η εικόνα computer.gif θα τοποθετηθεί έτσι ώστε η επάνω αριστερά γωνία της να βρίσκεται στο σημείο 20, 30 των αξόνων :

*ΠΡΟΓΡΑΜΜΑ Διαχείρισης Εικόνας*

```
import java.awt.Graphics;
import java.awt. Image;

public main ypologistis extends java.applet.Applet {
    Image terminal;

    public void init() {
        terminal = getImage(getCodeBase(), "computer.gif");
    }

    public void paint(Graphics g) {
        g. drawImage(terminal, 20, 30, this);
    }
}
```

Το αποτέλεσμα αυτού του προγράμματος φαίνεται στο παρακάτω σχήμα όπου καταγράφεται η θέση της εικόνας computer.gif στο σύστημα των συντεταγμένων της Java.




Εναλλακτικά, η drawImage() μπορεί να χρησιμοποιηθεί με έξι παραμέτρους : το όνομα του αρχείου εικόνας, τις x,y συντεταγμένες της επάνω αριστερά γωνίας τοποθέτησης της εικόνας, το πλάτος και το ύψος του πλαισίου μέσα στο οποίο θα τοποθετηθεί η εικόνα και τη λέξη-κλειδί *this*. Με αυτόν τον τρόπο υπάρχει δυνατότητα ρύθμισης του μεγέθους μίας εικόνας στο επιθυμητό μέγεθος σχετικά με τις ανάγκες της εφαρμογής. Υπάρχει βέβαια απόκλιση στην ευκρίνεια της εικόνας λόγω των επιπλέον ρυθμίσεων του αρχικού μεγέθους της. Η καλύτερη τεχνική είναι οι αυξο-μειώσεις του μεγέθους να ακολουθούν τα ποσοστά ύψους-πλάτους του αρχικού μεγέθους της εικόνας. Οι μέθοδοι getWidth() και getHeight() της κλάσης Image δίνουν αντίστοιχα το πλάτος και το ύψος ενός αρχείου εικόνας.

Η κλάση Image διαθέτει σημαντικό αριθμό κλάσεων και μεθόδων που δίνουν τη δυνατότητα τροποποίησης των αρχείων εικόνων με προσθήκη χρώματος ή ακόμα και σχεδίαση εικόνας bitmap μέσω κατάλληλων εντολών. Απαιτείται ιδιαίτερη γνώση θεμάτων διαχείρισης εικόνας έτσι ώστε να γίνει λειτουργική χρήση των εργαλείων που παρέχει το πακέτο java.awt.image.

## 5.3. Κινούμενες εικόνες

Η δημιουργία και διαχείριση κινούμενων εικόνων(animation) στην Java περιλαμβάνει τα εξής βήματα :

- δημιουργία του πλαισίου της κινούμενης εικόνας
- σχεδίαση μέσα σε αυτό το πλαίσιο
- επανάληψη της σχεδίασης όσες φορές χρειάζεται έτσι ώστε να δημιουργείται η "αυταπάτη" της κίνησης στην οθόνη.

Οπως περιγράφηκε στις προηγούμενες παραγράφους η μέθοδος paint() καλείται από την Java όταν η μικρο-εφαρμογή χρειάζεται να σχεδιάσει. Επομένως μπορεί να επαναληφθεί η σχεδίαση κάθε στιγμή με επαναληπτική κλήση της μικρο-εφαρμογής. Εάν η επανάληψη της σχεδίασης γίνει αρκετά γρήγορα δίνεται η ψευδαίσθηση της κίνησης μέσα στο πλαίσιο που έχει δημιουργηθεί. Η μέθοδος paint() δημιουργεί το τρέχον πλαίσιο σχεδίασης και στη συνέχεια χρειάζεται να κληθεί η μέθοδος repaint().Η repaint() επανα-καλεί την paint() για να πραγματοποιήσει τη σχεδίαση μέσα στο πλαίσιο που έχει δημιουργηθεί. Στην πραγματικότητα, η repaint() πραγματοποιεί την επανα-σχεδίαση το συντομότερο δυνατό διάστημα που το σύστημα είναι διαθέσιμο.

Οι μέθοδοι start() και stop() που περιγράφηκαν στην Παράγραφο 4.3 είναι ιδιαίτερα χρήσιμες για τη δημιουργία των κινούμενων εικόνων. Η start() ξεκινά την εκτέλεση της μικρο-εφαρμογής και συνήθως εδώ τοποθετούνται οι κατάλληλες εντολές που ρυθμίζουν το χρόνο εκτέλεσης. Η stop() σταματά την εκτέλεση της μικρο-εφαρμογής ώστε να μην καταλαμβάνονται πόροι του συστήματος, όταν ο χρήστης μετακινηθεί εκτός της τρέχουσας www σελίδας.

Στο παράδειγμα που ακολουθεί παρουσιάζεται η ημερομηνία και η ώρα με ενημέρωση της ώρας ανά δευτερόλεπτο.Ετσι, δημιουργείται ένα ψηφιακό ρολογιού με κινούμενη εικόνα .
Στο ακόλουθο παράδειγμα η χρήση της μεθόδου sleep() επιτρέπει τη ρύθμιση του χρόνου κατά τον οποίο θα επαναληφθεί η σχεδίαση ενώ οι εντολές try και catch επιτρέπουν στην Java να ενεργήσει κατάλληλα στην περίπτωση που προκύψει κάποιο λάθος. Η επαναληπτική κλήση της paint() με παράμετρο το αντικείμενο τύπου Date κάθε φορά δίνει τη δυνατότητα της εμφάνισης της τρέχουσας ώρας κάθε φορά.

*ΠΡΟΓΡΑΜΜΑ Κινούμενο Ψηφιακό Ημερολόγιο*

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

public class clock extends java.applet.Applet {
  Font keimeno = new Font("TimesRoman",Font.BOLD,24);
  Date hmerom;
  public void start() {
          while(true) {
             hmerom = new Date();
             repaint();
             try{Thread.sleep(1000);}
             catch (InterruptedException e) { }
          }
  }
  public void paint (Graphics g) {
          g.setFont(keimeno);
          g.drawString (hmerom.toString( ), 5, 20);
    }
}
```

Η δημιουργία κινούμενων εικόνων με χρήση αρχείων εικόνων πραγματοποιείται με παρόμοιο τρόπο. Η διαφορά είναι ότι χρειάζεται ένας αριθμός διαφορετικών εικόνων οι οποίες θα εναλλάσσονται αντί να έχουμε χρήση και εναλλαγή διαφορετικών μεθόδων σχεδίασης. Τα ονόματα των αρχείων των διαφορετικών εικόνων συνήθως αποθηκεύονται σε έναν πίνακα από όπου καλείται η κατάλληλη εικόνα κάθε φορά με βάση τη σειρά εμφάνισης τους ώστε να καταλήγουμε στην κίνηση εικόνων στην οθόνη.

## 5.4. Προγραμματισμός ακολουθίας σπονδυλωτών ενοτήτων

Στην Java υποστηρίζεται η εκτέλεση προγραμμάτων που αποτελούνται από σπονδυλωτές προγραμματιστικές. ενότητες (threads). Επομένως, υπάρχει η δυνατότητα παράλληλης εκτέλεσης διαφόρων σπονδυλωτών ενοτήτων (multithreading) οι οποίες εκτελούνται παράλληλα χωρίς να παρεμβαίνει η μία στην εκτέλεση της άλλης. Η εκτέλεση μίας σπονδυλωτής ενότητας περιλαμβάνει την εκκίνηση με εκτέλεση του κώδικα αρχικοποίησης, την κλήση των μεθόδων και των ενοτήτων προγραμμάτων που απαιτούνται και τη συνέχιση της απαραίτητης επεξεργασία έως το τέλος και την έξοδο του προγράμματος. Επομένως, με τη χρήση πολλαπλών παράλληλων

σπονδυλωτών ενοτήτων προγραμμάτων που εκτελούνται ταυτόχρονα και ανεξάρτητα υπάρχει η δυνατότητα της παράλληλης εκτέλεσης μικρο-εφαρμογών μέσα από μία www σελίδα. Η χρήση πολλαπλών ανεξάρτητων σπονδυλωτών ενοτήτων αποτελεί μία χρήσιμη πρακτική στην Java ειδικά για τις περιπτώσεις που έχουμε κώδικα που θα συνεχίζει να εκτελείται για αρκετό χρονικό διάστημα.
Για να δημιουργηθεί μία μικρο-εφαρμογή που χρησιμοποιεί σπονδυλωτές ενότητες χρειάζεται να γίνουν οι παρακάτω ενέργειες :

- μετατροπή της δήλωση της κλάσης με προσθήκη των λέξεων implements Runnable

```
public class efarmogi extends java.applet.Applet
implements Runnable { . . .
}
```

όπου το ενδιάμεσο Runnable καθορίζει το περιβάλλον και τη συμπεριφορά που χρειάζεται η μικρο-εαφρομογή για να εκτελέσει την ακολουθία των σπονδυλωτών ενοτήτων.

- προσθήκη μίας στιγμιαίας μεταβλητής που να κρατά την κάθε σπονδυλωτή ενότητα της μικρο-εφαρμογής

```
Thread runner;
```

Η Thread είναι μία κλάση του πακέτου java.lang

- τροποποίηση της μεθόδου start() έτσι ώστε να δημιουργεί την σπονδυλωτή ενότητα και να τη θέτει σε εκτελέσιμη κατάσταση. Για παράδειγμα μία τυπική start() μέθοδος έχει τη μορφή :

```
public void start() {
    if (runner = = null) {
        runner = new Thread(this);
        runner.start();
    }
}
```

Η start() απλώς δημιουργεί τη σπονδυλωτή ενότητα ενώ το κυρίως κομμάτι δημιουργίας αναπτύσσεται στην μέθοδο run()

- δημιουργία της μεθόδου run() που να περιλαμβάνει τον κώδικα που θέτει τη μικροεφαρμογή σε εκτέλεση.

```
public void run() {
    // κώδικας των ενεργειών που εκτελεί η μικρο-εφαρμογή
}
```

Η run() αποτελεί την "καρδιά" της μικρο-εφαρμογής όπου δίνεται ο κώδικας εκκίνησης, ο βρόγχος που ακολουθεί η μικρο-εφαρμογή, πραγματοποιείται η δημιουργία νέων αντικειμένων, η κλήση μεθόδων και γενικά το σύνολο των ενεργειών που θα επιτρέψουν στην μικρο-εφαρμογή να εκτελεσθεί κανονικά.

- προσθήκη μίας μεθόδου stop() η οποία έχει ως σκοπό τη λήξη της εκτέλεσης της συγκεκριμένης σπονδυλωτής ενότητας, μόλις ο χρήστης

φύγει από τη συγκεκριμένη www σελίδα η οποία καλεί τη μικρο-εφαρμογή. Μία ενδεικτική stop() μέθοδος είναι :

```
public void stop() {
    if (runner ! = null) {
        runner.stop();
        runner = null;
    }
}
```

Ουσιαστικά, η μέθοδος stop() σταματά την εκτέλεση της σπονδυλωτής ενότητας και δίνει στη μεταβλητή runner την τιμή null ώστε το αντικείμενο τύπου Thread που είχε δημιουργηθεί να είναι διαθέσιμο για διαγραφή από τη μνήμη μετά από συγκεκριμένο χρονικό διάστημα (μέσω του συλλέκτη "απορριμμάτων"). Εάν ο χρήστης επανέλθει στη www σελίδα αυτή η start() μέθοδος θα δημιουργήσει μία νέα σπονδυλωτή ενότητα και θα εκκινήσει και πάλι τη μικρο-εφαρμογή.

Στη συνέχεια παρουσιάζεται το πρόγραμμα παρουσίασης του ψηφιακού ρολογιού με την προσθήκη υποστήριξης σπονδυλωτών ενοτήτων :

*ΠΡΟΓΡΑΜΜΑ Κινούμενο Ψηφιακό Ημερολόγιο*
*με χρήση σπονδυλωτών ενοτήτων (threads)*

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

public class clockN extends java.applet.Applet implements Runnable {
  Font keimeno = new Font("TimesRoman",Font.BOLD,24);
  Date hmerom;
  Thread runner;

  public void start() {
      if (runner = = null) {
          runner = new Thread(this);
          runner.start();
      }
  }
  public void stop() {
          if (runner ! = null) {
              runner.stop();
              runner = null;
          }
  }

  public void run() {
          while(true) {
              hmerom = new Date();
              repaint();
```

```
                try{Thread.sleep(1000);}
                catch (InterruptedException e) { }
            }
        }

        public void paint (Graphics g) {
            g.setFont(keimeno);
            g.drawString (hmerom.toString( ), 5, 20);
        }
    }
```

Η δημιουργία και εμφάνιση των κινούμενων εικόνων στην Java με τον τρόπο που παρουσιάστηκε μέχρι στιγμής, προκαλεί κάποιο τρεμούλιασμα στην οθόνη. Το τρεμούλιασμα οφείλεται στο ότι στην πραγματικότητα η repaint() καλεί τη μέθοδο update() η οποία καθαρίζει την οθόνη από ότι έχει παρουσιάσει και στη συνέχεια καλεί την paint() η οποία σχεδιάζει το περιεχόμενο του τρέχοντος πλαισίου. Το καθάρισμα της οθόνης μεταξύ των διαφορετικών πλαισίων προκαλεί το τρεμούλιασμα της οθόνης και υπάρχουν δύο τρόποι επίλυσης του :

- νέα δήλωση της μεθόδου update() έτσι ώστε να μην καθαρίζει την οθόνη κάθε φορά, ή εάν χρειάζεται να καθαρίζει κάποια επιμέρους κομμάτια της οθόνης.

- νέα δήλωση των μεθόδων update() και paint().

# 6. ΔΙΑΧΕΙΡΙΣΗ ΓΕΓΟΝΟΤΩΝ ΚΑΙ ΔΙΑΛΟΓΙΚΗ ΕΠΙΚΟΙΝΩΝΙΑ

## 6.1. Διαχείριση απλών γεγονότων

Τα γεγονότα στην Java αποτελούν μέρος του πακέτου AWT (Abstract Window Toolkit). Ενα γεγονός αντιπροσωπεύει τον τρόπο με τον οποίο το πακέτο AWT επικοινωνεί τόσο με τον προγραμματιστή, όσο και με τα άλλα μέρη του AWT, όταν πραγματοποιείται μία ενέργεια. Αυτή η ενέργεια μπορεί να αφορά στην εισαγωγή στοιχείων από το χρήστη (π.χ. κινήσεις και επιλογές που γίνονται με το ποντίκι, πληκτρολόγηση), σε μετατροπές στο περιβάλλον του συστήματος(π.χ. άνοιγμα κλείσιμο παραθύρων) ή σε οποιοδήποτε άλλο σύνολο ενεργειών που επηρεάζουν τη λειτουργία του συστήματος. Το πακέτο AWT περιλαμβάνει ένα σύνολο κλάσεων που υποστηρίζουν όλα τα κατάλληλα εργαλεία διαχείρισης των κυριότερων ενεργειών των χρηστών όπως παράθυρα, πλήκτρα επιλογής, καταστάσεις επιλογών(menus) κλπ. Η διαχείριση και δημιουργία των γεγονότων γίνεται μέσω του πακέτου AWT που υλοποιεί και όλα τα ενδιάμεσα προσαρμογής των χρηστών (UI : User Interfaces)

Επομένως, όταν συμβεί οποιοδήποτε γεγονός σε κάποιο από τα συστατικά μέρη του πακέτου AWT, δημιουργείται ένα γεγονός. Κάποια από τα γεγονότα τα διαχειρίζεται το πρόγραμμα πλοήγησης(browser) και δε χρειάζεται η παρέμβαση του πακέτου AWT. Μερικά γεγονότα όμως χρειάζονται ειδικό προγραμματισμό έτσι ώστε να γίνεται κατάλληλη διαχείριση τους από τη μικρο-εφαρμογή.

### 6.1.1. Διαχείριση κινήσεων ποντικιού

Οι κινήσεις και οι επιλογές που γίνονται με χρήση του ποντικιού εκφράζουν και τις επιλογές που κάνει ο χρήστης σε διάφορα σημεία εκτέλεσης μίας μικρο-εφαρμογής. Η κίνηση και η επιλογή μέσω του ποντικιού(κλικ) που κάνει ο χρήστης κάθε φορά μπορεί να υποστηριχτεί με απλές ή με σύνθετες ενέργειες που θα εκτελέσει η μικρο-εφαρμογή.

Οταν γίνεται ένα απλό κλικ με το ποντίκι το πακέτο AWT δημιουργεί δύο γεγονότα : το mouseDown όταν πατάμε το πλήκτρο του ποντικιού και το mouseUp όταν ελευθερώνουμε το ποντίκι. Χρειάζεται η δημιουργία δύο γεγονότων διότι είναι διαφορετική η χρήση κατά το πάτημα και κατά την ελευθέρωση του πλήκτρου του ποντικιού. Η διαχείριση αυτών των γεγονότων είναι απλή. Χρειάζεται να δηλωθούν νέες μέθοδοι *mouseDown*() και *mouseUp*() για την υλοποίηση των νέων ενεργειών που πρέπει να γίνονται. Οι μέθοδοι mouseDown() και mouseUp() δέχονται τρεις παραμέτρους : το γεγονός, και τις x,y συντεταγμένες του σημείου της οθόνης όπου έγινε το κλικ.

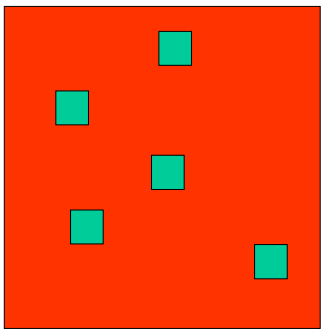
```
public boolean mouseDown(Event gegonos, int x, int y) {
    System.out.println("Πάτημα ποντικιού στη θέση "+x+","+y);
    return true;
}
```

Η παράμετρος *gegonos* είναι μία στιγμή ύπαρξης της κλάσης Event, ενώ κάθε πάτημα ποντικιού θα έχει ως αποτέλεσμα την εκτύπωση του μηνύματος ακολουθούμενο από το σημείο όπου έγινε το κλικ του ποντικιού. Η μέθοδος αυτή επιστρέφει boolean τιμή και ειδικότερα, θα επιστρέφει true όταν η μέθοδος αφορά στο συγκεκριμένο γεγονός. Ολα τα γεγονότα που δημιουργούνται από το σύστημα αποτελούν μία στιγμή ύπαρξης της κλάσης Event και περιέχουν πληροφορίες για το πότε και σε ποιο σημείο προέκυψε το γεγονός, το είδος του γεγονότος και οποιαδήποτε άλλη χρήσιμη σχετική πληροφορία. Συνήθως είναι χρήσιμο να υπάρχει ένας σύνδεσμος διαχείρισης με κάθε γεγονός που δημιουργείται ώστε να υπάρχει τρόπος αναφοράς και κατάλληλης διαχείρισης του γεγονότος. Η γνώση των x,y συντεταγμένων είναι ιδιαίτερα χρήσιμη διότι έτσι είναι γνωστή η ακριβής θέση του σημείου στο οποίο έγινε το κλικ.

Στο παράδειγμα της μικρο-εφαρμογής που ακολουθεί σχεδιάζουμε επιφάνειες πράσινων παραλληλόγραμμων πλάτους 10 και ύψους 10 σε κόκκινο φόντο, με την επάνω αριστερά γωνία τους στο σημείο που έγινε το κλικ του ποντικιού. Το πρόγραμμα επιτρέπει τη σχεδίαση έως το πολύ 5 παραλληλογράμμων.

| *ΠΡΟΓΡΑΜΜΑ* | *ΑΠΟΤΕΛΕΣΜΑ* |
|---|---|

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

public class parall extends java.applet.Applet {
final int maxp = 5;
int simeiax[ ] = new int[maxp];
int simeiay ] = new int[maxp];
int nump = 0;

public void init() {
    setBackground(Color.red);
}
public boolean mouseDown(Event geg, int x, int y) {
        if (nump < maxp)
            news(x,y);
       else
           System.out.println("Τέλος επιλογής σημείων");
        return true;
}
void news(int x, int y) {
       simeiax[nump] = x;
       simeiay[nump] = y;
       nump++;
```



```
        repaint();
    }
    public void paint(Graphics g) {
        g.setColor(Color.green);
        for (int i=0; i<nump; i++) {
            g.fillRect(simeiax[i], simeiay[i], 10, 10);
        }
    }
}
```

Η οποιαδήποτε μετακίνηση του ποντικιού προς οποιαδήποτε κατεύθυνση δημιουργεί ένα αντίστοιχο γεγονός. Υπάρχουν δύο μέθοδοι για τη μετακίνηση του ποντικιού : η *mouseMove()* όταν γίνεται απλή μετακίνηση του ποντικιού και η *mouseDrag()* όταν γίνεται μετακίνηση με πατημένο συνεχώς το πλήκτρο του ποντικιού οπότε και σύρουμε τις επιλογές που έχουν γίνει με το ποντίκι. Οι μέθοδοι αυτές παίρνουν τις ίδιες παραμέτρους όπως και οι προηγούμενες μέθοδοι δηλαδή οι παράμετροι τους είναι *Event geg, int x, int y.* Επιπλέον υπάρχουν και οι μέθοδοι *mouseEnter()* και *mouseExit()* που καλούνται αντίστοιχα όταν ο δείκτης (δρομέας) του ποντικιού εισέρχεται ή εξέρχεται της μικρο-εφαρμογής. Οπως και προηγούμενα, οι μέθοδοι αυτές παίρνουν τις ίδιες παραμέτρους : *Event geg, int x, int y.*

### 6.1.2. Διαχείριση ενεργειών πληκτρολόγησης

Τα γεγονότα πληκτρολόγησης δημιουργούνται κάθε φορά που ο χρήστης πατήσει κάποιο πλήκτρο του πληκτρολογίου. Με τη διατήρηση γεγονότων πληκτρολόγησης δίνεται η δυνατότητα ελέγχου και κατάλληλων ρυθμίσεων με βάση το εκάστοτε πλήκτρο που πατήθηκε. Για να "συλλάβουμε" το πάτημα ενός πλήκτρου χρησιμοποιούμε τη μέθοδο keyDown(). Η ακόλουθη keyDown() μέθοδος έχει ως αποτέλεσμα την εκτύπωση του πλήκτρου που πατήθηκε.

```
public boolean keyDown(Event gegonos, int pliktro) {
    System.out.println("Πάτημα χαρακτήρα "(char) pliktro);
    return true;
}
```

Ακόμα υπάρχει και η *keyUp*() που αντιστοιχεί με την ενέργεια ελευθέρωσης του πλήκτρου και έχει ως παραμέτρους το γεγονός και το πλήκτρο δηλαδή : *Event gegonos, int pliktro.* Η κλάση Event παρέχει ένα σύνολο μεταβλητών κλάσης που αναφέρονται στα συνήθη αλφαριθμητικά πλήκτρα και παρουσιάζονται στον πίνακα που ακολουθεί :

| *Μεταβλητή Κλάσης* | *Πλήκτρο* |
|---|---|
| Event.HOME | Το πλήκτρο HOME |
| Event.END | Το πλήκτρο END |

| | |
|---|---|
| Event.PGUP | Το πλήκτρο Page Up |
| Event.PGDN | Το πλήκτρο Page Down |
| Event.UP | Το τόξο με κατεύθυνση προς τα επάνω. |
| Event.DOWN | Το τόξο με κατεύθυνση προς τα κάτω. |
| Event.LEFT | Το τόξο με κατεύθυνση προς τα αριστερά. |
| Event.RIGHT | Το τόξο με κατεύθυνση προς τα δεξιά. |

Γενικά, στην κλάση Event μπορούν να ελεγχθούν τα παρακάτω γεγονότα σχετικά με τα στοιχεία πληκτρολόγησης ή τις κινήσεις του ποντικιού :

| *Γεγονός* | *Ενέργεια* |
|---|---|
| Event.KEY_PRESS | Προκύπτει από το πάτημα οποιουδήποτε πλήκτρου, είναι ακριβώς η ίδια με τη μέθοδο keyDown(). |
| Event.KEY_RELEASE | Προκύπτει όταν αφήνουμελεύθερο κάποιο πλήκτρο. |
| Event.KEY_ACTION | Προκύπτει όταν πατάμε ένα πλήκτρο |
| Event.KEY_ACTION_ RELEASE | Προκύπτει όταν αφήνουμε το πάτημα ενός πλήκτρου |
| Event.MOUSE_DOWN | Προκύπτει από το πάτημα του πλήκτρου του ποντικιού (ίδιο με τη μέθοδο mouseDown()) |
| Event. MOUSE_UP | Προκύπτει όταν αφήνουμελεύθερο το πλήκτρο του ποντικιού (ίδιο με τη μέθοδο mouseUp()) |
| Event. MOUSE_MOVE | Προκύπτει όταν μετακινούμε το ποντίκι (ίδιο με τη μέθοδο mouseMove()) |
| Event. MOUSE_DRAG | Προκύπτει όταν σύρουμε το ποντίκι (ίδιο με τη μέθοδο mouseDrag()) |
| Event.MOUSE_ENTER | Προκύπτει όταν το ποντίκι εισέρχεται στα όρια της μικρο-εφαρμογής(ίδιο με τη μέθοδο mouseEnter()) |
| Event.MOUSE_EXIT | Προκύπτει όταν το ποντίκι εξέρχεται από τα όρια της μικρο-εφαρμογής(ίδιο με τη μέθοδο mouseExit()) |

## 6.2. Ο AWT διαχειριστής γεγονότων

Το πακέτο AWT διαθέτει μία γενική μέθοδο διαχείρισης των γεγονότων, την handleEvent(). Η handleEvent() εκφράζει το γενικό τρόπο με τον οποίο το πακέτο AWT διαχειρίζεται τα γεγονότα που υφίστανται μεταξύ των διαφόρων συστατικών κομματιών μίας εφαρμογής καθώς και των γεγονότων που βασίζονται στα δεδομένα που εισάγει ο χρήστης.

Στην εξ'ορισμού handleEvent() μέθοδο πραγματοποιείται η επεξεργασία των βασικών γεγονότων και των μεθόδων που παρουσιάστηκαν στις προηγούμενες Παραγράφους. Κάθε αντικείμενο τύπου Event διαθέτει τη στιγμιαία μεταβλητή ID που είναι η ακέραια τιμή που αφορά στη συγκεκριμένη ενέργεια που έχει γίνει. Για να οριστεί διαφορετικός τρόπος αντιμετώπισης των γεγονότων και των μεθόδων δηλώνεται νέα handleEvent() μέθοδος όπως στο παράδειγμα που ακολουθεί, όπου εκτυπώνεται πληροφορία για το γεγονός το οποίο εκτελείται. :

*ΠΡΟΓΡΑΜΜΑ*

```
public boolean handleEvent(Event geg) {

switch(geg.id) {
    case Enent.MOUSE_DOWN :
         System.out.println("Ποντίκι προς τα κάτω :"+geg.x+","+geg.y);
         return true;
    case Enent.MOUSE_UP :
         System.out.println("Ποντίκι προς τα επάνω :"+geg.x+","+geg.y);
         return true;
    case Enent.MOUSE_MOVE :
         System.out.println("Μετακίνηση Ποντικιού :"+geg.x+","+geg.y);
         return true;
    case Enent.MOUSE_DRAG :
         System.out.println("Σύρσιμο Ποντικιού ¨:"+geg.x+","+geg.y);
         return true;
    default :
         return false;
    }
}
```

## 6.3. Η εργαλειοθήκη του πακέτου AWT

Η εργαλειοθήκη Αφαιρετικών Παραθύρων (AWT: Abstract Windows Toolkit) αποτελεί το πακέτο χαρτογράφησης της Java. Το πακέτο AWT παρέχει τη δυνατότητα χρήσης των παρακάτω στοιχείων :

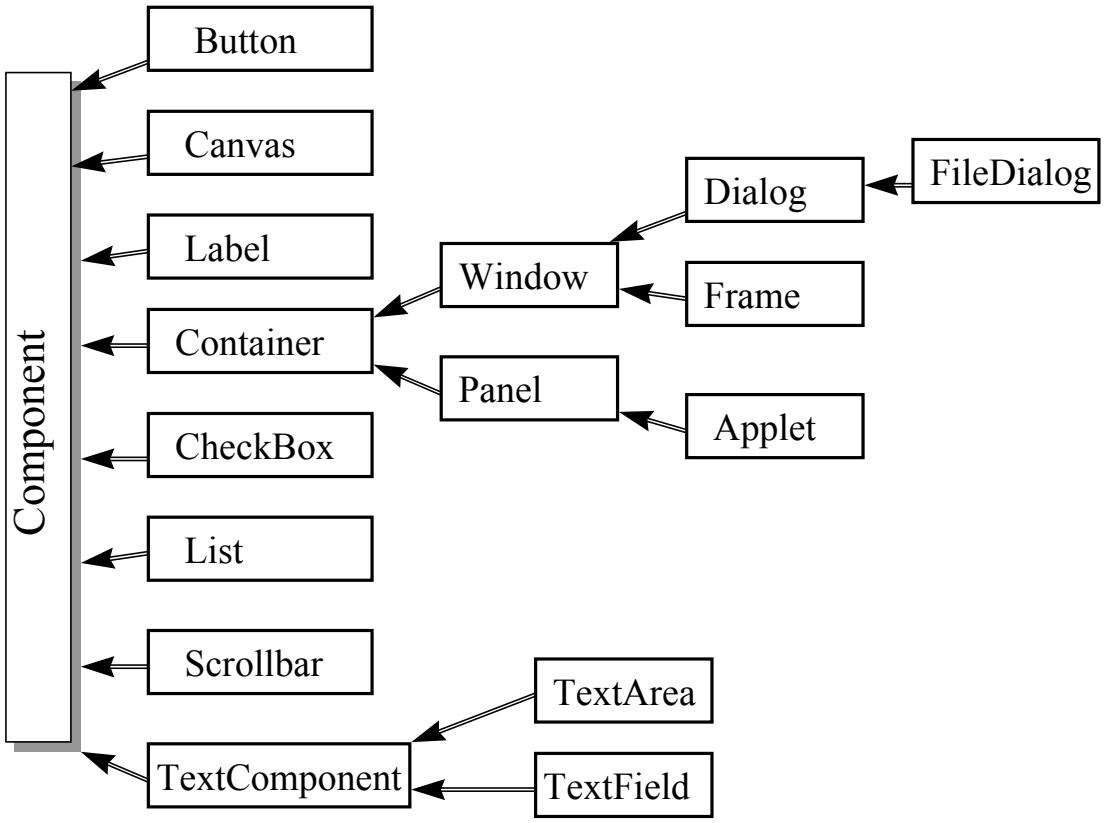
- πλήρες σύνολο προσαρμοστικών εργαλείων για το χρήστη (UI : User Interfaces), όπως παράθυρα, πλήκτρα, menus, κυλιόμενες λίστες επιλογής κλπ.

- υποστήριξη πολλαπλών υποδοχών UI, με ενσωμάτωση ενός UI σε κάποιο άλλο.

- σύστημα διαχείρισης γεγονότων και διαχείριση γεγονότων χρηστών μεταξύ των επιμέρους ενοτήτων του AWT.

- μηχανισμούς παράθεσης των επιμέρους στοιχείων έτσι ώστε να επιτρέπεται ο σχεδιασμός δια-πλατφορμικών UI.

Η βασική ιδέα του πακέτου AWT είναι ότι ένα Java παράθυρο είναι ένα σύνολο από συστατικά τα οποία περιέχονται το ένα μέσα στο άλλο, ξεκινώντας από το πλέον εξωτερικό παράθυρο και προχωρώντας προς το πλέον εσωτερικό UI συστατικό. Η ύπαρξη ένθετων στοιχείων επιβάλλει και μία ιεραρχία στα συστατικά στοιχεία ενός παραθύρου. Η ιεραρχία αυτή καθορίζει την τοποθέτηση των διαφόρων αντικειμένων στην οθόνη καθώς και την εναλλαγή γεγονότων από το ένα στοιχείο στο άλλο. Στο παρακάτω σχήμα καταγράφεται η ιεραρχία των κυριότερων συστατικών του AWT.



Η υπερ-κλάση των περισσοτέρων κλάσεων που αποτελούν το AWT είναι η κλάση Component(συστατικό). Τα κυριότερα συστατικά του AWT είναι :

- Υποδοχέας (Container) : αποτελεί το πλέον γενικό συστατικό του AWT και μπορεί να περιλαμβάνει ένθετους άλλους υποδοχείς ή άλλα συστατικά. Η συχνότερη περίπτωση υποδοχέα είναι το *panel* που αναπαριστά έναν υποδοχέα που μπορεί να εμφανιστεί στην οθόνη. Η κλάση Applet είναι μία υπο-κλάση της κλάσης Panel και έτσι οι μικρο-εφαρμογές αποτελούν ένα πλήρες μέρος της ιεραρχίας των συστατικών του AWT πακέτου.

- Επιφάνεια(canvas): αποτελεί μία απλή επιφάνεια σχεδίασης, ιδιαίτερα χρήσιμη για τη σχεδίαση εικόνων ή τη διενέργεια άλλων ενεργειών σχετικά με τη χρήση γραφικών.

- Συστατικά χρηστών(UI): περιλαμβάνουν πλήκτρα επιλογής (Button), λίστες, λίστες επιλογών αναδίπλωσης (popup menus) και άλλα τυπικά στοιχεία προσαρμοστικών εργαλείων χρηστών.

- Παραθυρικά συστατικά : περιλαμβάνουν τα παράθυρα, τα πλαίσια, μπάρες επιλογών και πλαίσια διαλόγου.

*ΣΗΜΕΙΩΣΗ :* Το πακέτο JDK της Java περιέχει αναλυτική περιγραφή και καταγραφή όλων των μεθόδων και κλάσεων που αφορούν στα παρακάτω εργαλεία και τεχνικές. Υπάρχει πλήρης υποστήριξη και τεκμηρίωση τους στο κομμάτι API (Application Programming Interface) του προγραμματισμού εφαρμογών.

### 6.3.1. Τα βασικά προσαρμοστικά εργαλεία χρηστών

Αποτελούν τα βασικά και συνηθέστερα χρησιμοποιούμενα εργαλεία του AWT πακέτου. Για να προστεθεί ένα συστατικό προσαρμογής χρηστών (UI) (για παράδειγμα στο panel) χρειάζεται η μέθοδος add() :

```
public void init() {
    Button b = new Button("OK");
    add(b);
}
```

Η θέση που θα καταλάβει αυτό το νέο πλήκτρο στο panel εξαρτάται από τη ρύθμιση της επιφάνειας του panel, η εξ'ορισμού παράθεση είναι η τοποθέτηση στο κέντρο(centered). Αφού δημιουργηθεί ένα συστατικό υπάρχει δυνατότητα προσθήκης γεγονότων που θα συσχετίζονται με τα συγκεκριμένα συστατικά. Στη συνέχεια παρουσιάζονται τα κυριότερα συστατικά που αφορούν στην εργαλειοθήκη των UI :

- *Label:* αποτελούν συμβολοσειρές κειμένου οι οποίες χρησιμοποιούνται για να καθορίζουν τη θέση (ως ετικέτες) άλλων UI συστατικών.

Ακολουθούν τη διάταξη του panel και δε χρειάζεται να επανα-δηλωθούν κατά την επανασχεδίαση του panel. Για να δημιουργηθεί ένα label μπορεί να χρησιμοποιηθεί μία από τις παρακάτω μεθόδους :

Label( ) : δημιουργεί μία ετικέτα άνευ κειμένου, η οποία τοποθετείται προς τα αριστερά

Label(String) : δημιουργεί μία ετικέτα με όνομα το κείμενο(String) που είναι μέσα στην παρένθεση, η οποία επίσης τοποθετείται προς τα αριστερά

Label(String, int) : δημιουργεί μία ετικέτα με όνομα το κείμενο(String) που είναι μέσα στην παρένθεση και τοποθέτηση στη θέση που δηλώνεται μέσω του ακεραίου(int). Οι δεδομένες τοποθετήσεις αποθηκεύονται στις μεταβλητές κλάσης Label.RIGHT που έχει τιμή ακεραίου =2, Label.LEFT με τιμή 0 και Label.CENTER με τιμή 1.

- *Button:* αποτελούν την αναπαράσταση απλών πλήκτρων που διαθέτουν κείμενο και εάν ο χρήστης επιλέξει κάποιο από αυτά εκτελείται η αντίστοιχη ενέργεια. Για να δημιουργηθεί ένα Button μπορεί να χρησιμοποιηθεί μία από τις παρακάτω μεθόδους :

  Button( ) : δημιουργεί ένα πλήκτρο άνευ κειμένου

  Button(String) : δημιουργεί ένα πλήκτρο με το κείμενο της συμβολοσειράς (τύπου String )της παρένθεσης.


Τα παραπάνω πλήκτρα είναι αποτέλεσμα της παρακάτω ακολουθίας εντολών :

```
add(new Button("ΞΕΚΙΝΗΣΕ");
add(new Button("ΣΤΑΜΑΤΗΣΕ");
add(new Button("ΕΠΙΣΤΡΟΦΗ");
```

- *Checkbox:* αποτελούν την αναπαράσταση πλήκτρων διακοπτών που μπορεί να έχουν δύο καταστάσεις (on ή off, true ή false κλπ). Για να δημιουργηθεί ένα Checkbox μπορεί να χρησιμοποιηθεί μία από τις παρακάτω μεθόδους :

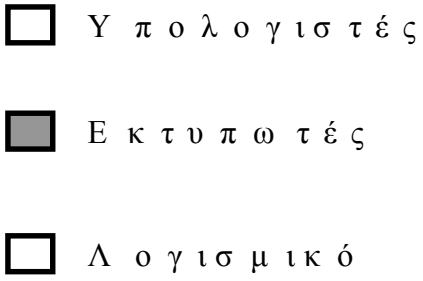
  Checkbox( ) : δημιουργεί ένα κενό πλήκτρο διακόπτη, μη επιλεγμένο

  Checkbox(String) : δημιουργεί ένα πλήκτρο διακόπτη, με το κείμενο της παρένθεσης ως ετικέτα.

Checkbox(String, null, boolean) : δημιουργεί ένα πλήκτρο διακόπτη το οποίο είναι επιλεγμένο ή όχι εάν η τιμή της boolean είναι αντίστοιχα true ή false.

Στο παρακάτω σχήμα παρουσιάζεται μία ακολουθία 3 πλήκτρων διακοπτών με επιλεγμένο το ένα από αυτά, τα οποία είναι αποτέλεσμα των εντολών :

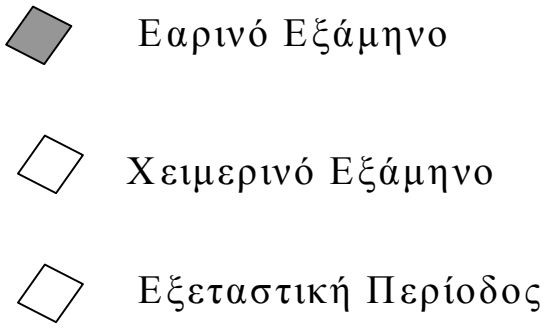
```
add(new Checkbox("Υπολογιστές");
add(new Checkbox("Εκτυπωτές", null, true);
add(new Checkbox("Λογισμικό");
```



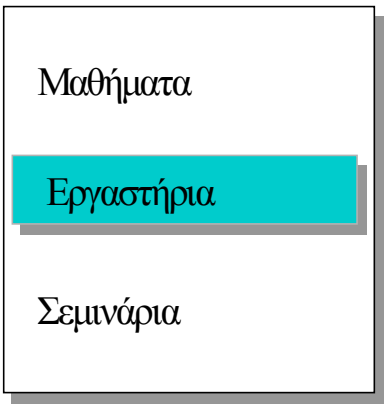

- *Radio Button* : είναι παρόμοια με τα Checkbox δηλαδή, αποτελούν την αναπαράσταση πλήκτρων διακοπτών που μπορεί να έχουν δύο καταστάσεις, με τη διαφορά ότι μόνο ένα πλήκτρο από τη σειρά των πλήκτρων, μπορεί να είναι επιλεγμένο. Για να δημιουργηθεί ένα Radio Button πρέπει να δημιουργηθεί μία στιγμή ύπαρξης ενός CheckboxGroup και στη συνέχεια να προστίθενται ξεχωριστά Checkbox :

  Στο παρακάτω σχήμα παρουσιάζεται μία ακολουθία 3 πλήκτρων διακοπτών με επιλεγμένο το ένα από αυτά, τα οποία είναι αποτέλεσμα των εντολών :

```
CheckboxGroup omada = new CheckboxGroup();

add(new Checkbox("Εαρινό Εξάμηνο", omada, true));
add(new Checkbox("Χειμερινό Εξάμηνο", omada, false));
add(new Checkbox("Εξεταστική Περίοδος",omada, false));
```



- *Choice Menu* : αποτελούν καταστάσεις επιλογής που αναδιπλώνονται προς τα επάνω(popup) ή προς τα κάτω(pull down) και δίνουν τη δυνατότητα επιλογής ενός αντικειμένου από ένα menu. Το menu εμφανίζει την επιλογή στην οθόνη. Για να δημιουργηθεί ένα Choice Menu δημιουργείται μία στιγμή ύπαρξης της κλάσης Choice και στη συνέχεια με τη μέθοδο addItem() προστίθενται διακριτά αντικείμενα με τη σειρά που θέλουμε να εμφανίζονται στο menu. Τα Choice Menu επιτρέπουν την επιλογή ενός αντικειμένου της λίστας επιλογών κάθε φορά.




Το παραπάνω Choice menu κατασκευάζεται από την παρακάτω ακολουθία εντολών :

```
Choice  epil  = new Choice();
epil.addItem("Μαθήματα");
epil.addItem("Εργαστήρια");
epil.addItem("Σεμινάρια");
```

και την προσθήκη του menu στο panel με την εντολή : add(epil);

Μία σειρά μεθόδων βοηθά στη διαχείριση των menus, για παράδειγμα :

countItems() : επιστρέφει τον αριθμό των επιλογών ενός menu
select(int) : επιστρέφει την επιλογή της συγκεκριμένης θέσης του int
select(String) : επιστρέφει την επιλογή της συγκεκριμένης θέσης του String

- *Text Field* : δίνουν τη δυνατότητα στον αναγνώστη να εισάγει κείμενο. Για να δημιουργηθεί ένα Text Field μπορεί να χρησιμοποιηθεί μία από τις παρακάτω μεθόδους :

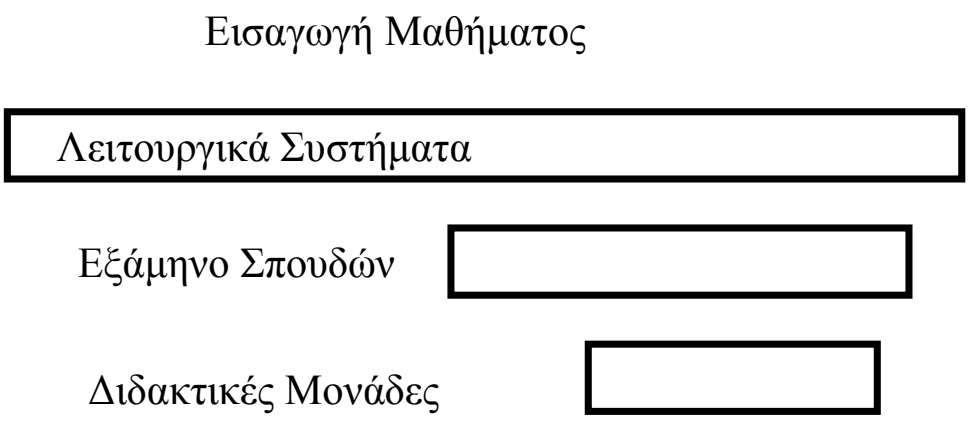
TextField( ) : δημιουργεί ένα κενό πεδίο κειμένου πλάτους 0

TextField(int) : δημιουργεί ένα κενό πεδίο κειμένου πλάτους που καθορίζεται από τον ακέραιο της παρένθεσης.

TextField(String) : δημιουργεί ένα πεδίο κειμένου πλάτους 0 με τιμή εκκίνησης της συμβολοσειρά που δίνεται στην παρένθεση.

TextField(String, int) : δημιουργεί ένα πεδίο κειμένου πλάτους (τύπου int) και τιμή τη συμβολοσειρά (τύπου String) που δίνονται στην παρένθεση.

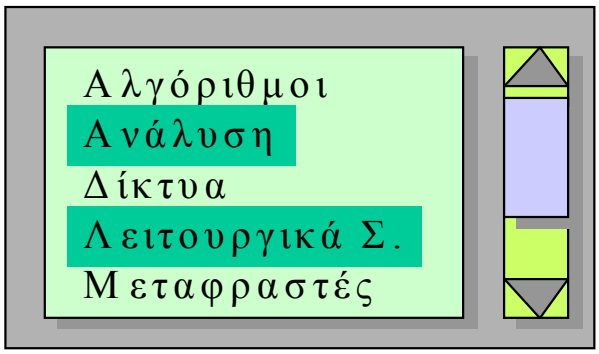
Τα TextField συνήθως έχουν περιορισμένη έκταση (π.χ. μίας γραμμής), ενώ για μεγαλύτερη έκταση κειμένου χρησιμοποιούνται τα TextArea. Οι κλάσεις TextField και TextArea αποτελούν υπο-κλάσεις της κλάσης TextComponent.




Τα παραπάνω TextField προκύπτουν με τις παρακάτω εντολές :

```
add(new Label("Εισαγωγή Μαθήματος"))
add(new TextField("Λειτουργικά Συστήματα", 50));
add(new Label("Εξάμηνο Σπουδών"));
add(new TextField(25));
add(new Label("Διδακτικές Μονάδες"));
TextField keim = new TextField(18);
add(keim);
```

- *Scrolling List* : οι λίστες ολίσθησης είναι παρόμοιες με την περίπτωση των Choice menu, με τη διαφορά ότι μπορεί να γίνουν περισσότερες από μία επιλογές κάθε φορά και υπάρχει μία κατάσταση επιλογών που μπορούμε να διασχίσουμε με τη βοήθεια κάποιας ράβδου κύλισης όπως φαίνεται στο παρακάτω σχήμα :




Για να δημιουργηθεί μία Scrolling List δημιουργείται μία στιγμή ύπαρξης της κλάσης List και στη συνέχεια προστίθενται τα αντικείμενα επιλογής της λίστας. Υπάρχουν δύο μέθοδοι κατασκευής της κλάσης List :

List( ) : δημιουργεί μία κενή λίστα ολίσθησης που επιτρέπει μία επιλογή κάθε φορά.

List(int, boolean) : δημιουργεί μία λίστα ολίσθησης με ορατό αριθμό επιλογών που καθορίζεται από τη μεταβλητή τύπου int και μία boolean μεταβλητή που καθορίζει εάν η λίστα ολίσθησης επιτρέπει πολλαπλές επιλογές ή όχι.

Στη συνέχεια παρατίθεται ο κώδικας που έχει ως αποτέλεσμα τη δημιουργία της λίστας του παραπάνω σχήματος :

```
List lista = new List (5, true);

lista.addItem("Αλγόριθμοι");
lista.addItem("Ανάλυση");
lista.addItem("Δίκτυα");
lista.addItem("Λειτουργικά Σ.");
lista.addItem("Μεταφραστές");
```

### 6.3.2. Επιφάνεια και διάταξη σχεδίασης

Οπως αναφέρθηκε, η επιφάνεια σχεδίασης (panel) του AWT μπορεί να περιέχει ένα σύνολο από προσαρμοστικά εργαλεία χρηστών (UI interfaces) καθώς και άλλες υπο-επιφάνειες. Η Java διαθέτει ένα σύνολο από ρυθμιστές των επιφανειών σχεδίασης(layout managers) ώστε να διευκολύνεται η διάταξη των συστατικών της οθόνης. Οι ρυθμιστές αυτοί έχουν αυτοματοποιημένες διαδικασίες τοποθέτησης και διαχείρισης των κυριότερων σχεδιαστικών εργαλείων.

Η υλοποίηση και παρουσίαση των κυριότερων AWT συστατικών στην οθόνη καθορίζεται από τη σειρά με την οποία αυτά προστίθενται στην επιφάνεια σχεδίασης και από το ρυθμιστή σχεδίασης που χρησιμοποιείται. Κάθε επιφάνεια σχεδίασης μπορεί να έχει το δικό της ρυθμιστή σχεδίασης

που καθορίζει και συντονίζει τη διάταξη των επιφανειών σχεδίασης. Υπάρχουν 5 βασικοί ρυθμιστές των επιφανειών σχεδίασης : FlowLayout(), GridLayout(), GridBagLayout(), BorderLayout(), και CardLayout(). Για να δημιουργηθεί ένας ρυθμιστής σχεδίασης για μία δεδομένη επιφάνεια σχεδίασης χρησιμοποιείται η μέθοδος setLayout() για τη συγκεκριμένη επιφάνεια σχεδίασης :

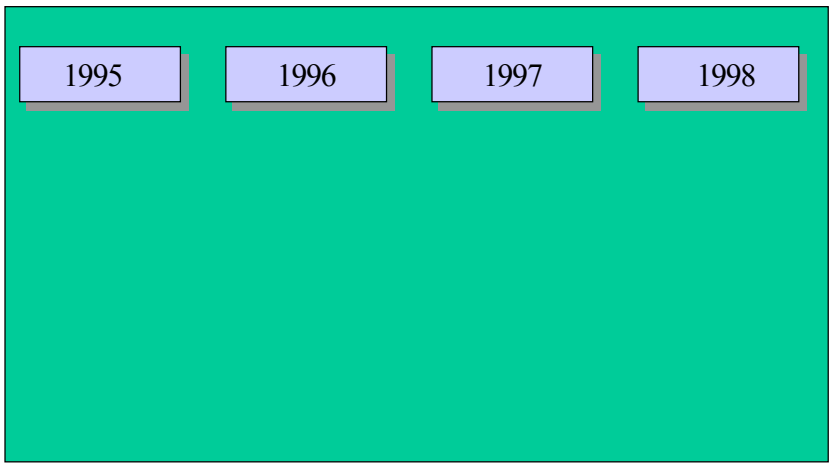
```
public void init() {
    setLayout(new FlowLayout());
}
```

Μετά την παραπάνω δήλωση του ρυθμιστή σχεδίασης, μπορεί να ξεκινήσει η προσθήκη των συστατικών της επιφάνειας σχεδίασης. Η σειρά προσθήκης είναι σημαντική για κάποιους από τους ρυθμιστές σχεδίασης

- *FlowLayout* : η FlowLayout κλάση αποτελεί την πλέον βασική από τους ρυθμιστές σχεδίασης. Σύμφωνα με την κλάση αυτή τα συστατικά της επιφάνειας σχεδίασης προστίθενται το ένα μετά το άλλο σειρά προς σειρά. Εάν ένα αντικείμενο δε χωράει σε μία σειρά μεταφέρεται στην επόμενη σειρά. Επίσης διαθέτει δυνατότητα στοίχισης για κάθε σειρά της επιφάνειας σχεδίασης. Η εντολή :

  *setLayout(new FlowLayout(FlowLayout.LEFT));*

  δημιουργεί την παρακάτω επιφάνεια σχεδίασης αριστερής στοίχισης, στην οποία προστίθενται τα συστατικά τύπου Button.

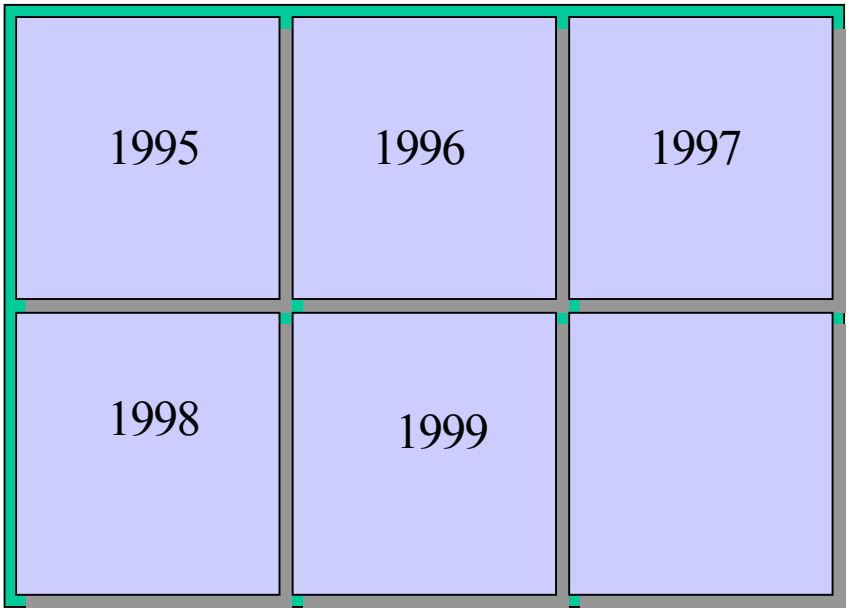



- *Grid και GridBagLayout* : με αυτές τις κλάσεις αυξάνονται οι δυνατότητες διάταξης των συστατικών σε μία επιφάνεια σχεδίασης διότι η επιφάνεια χωρίζεται σε σειρές και στήλες η τομή των οποίων καθορίζει ένα πλέγμα από "κελιά" τοποθέτησης συστατικών σχεδίασης. Κάθε συστατικό μπορεί να τοποθετεί σε ένα κελί του πλέγματος ξεκινώντας από το επάνω αριστερά κελί και προχωρώντας από αριστερά προς τα δεξιά ανά σειρά. Για να δημιουργηθεί ένα πλέγμα επιφάνειας σχεδίασης πρέπει να καθοριστεί στην setLayout() ο αριθμός των σειρών και των στηλών του πλέγματος που θέλουμε να

δημιουργηθεί. Επίσης, μπορεί να υπάρχει και περιθώριο μεταξύ των συστατικών. Η εντολή :

*setLayout(new GridLayout t(3,3));*

δημιουργεί την παρακάτω επιφάνεια σχεδίασης :

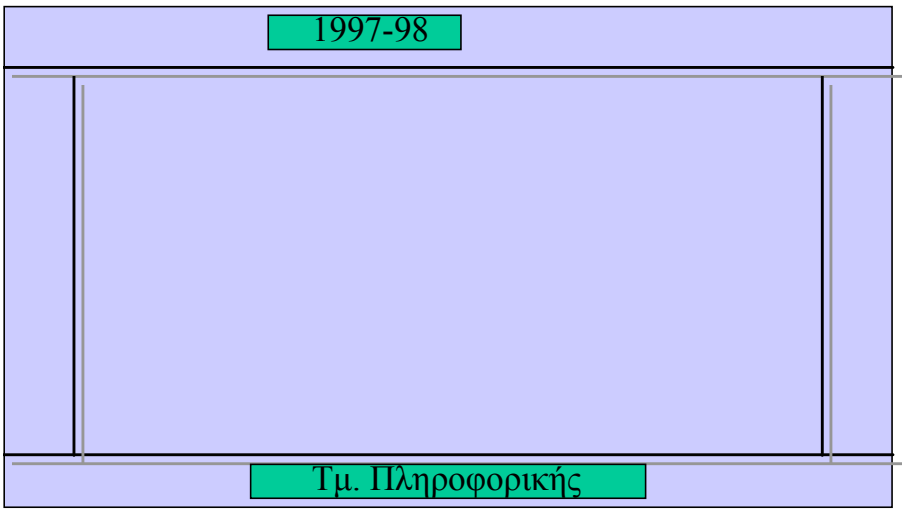



Η GridBagLayout κλάση αποτελεί μία παραλλαγή της GridLayout, με τη διαφορά ότι παρέχει περισσότερες δυνατότητες ελέγχου της παρουσίασης των συστατικών της επιφάνειας σχεδίασης.

- *BorderLayout* : παρουσιάζουν διαφορετική συμπεριφορά από τους προηγούμενους ρυθμιστές σχεδίασης. Όταν προστίθεται ένα συστατικό σχεδίασης σε μία επιφάνεια τύπου BorderLayout χρειάζεται να καθορίζεται η γεωγραφική κατεύθυνση δηλαδή προς βορά, προς νότο, ανατολικά, δυτικά και στο κέντρο. Η δημιουργία γίνεται και πάλι με χρήση της μεθόδου setLayout(), αλλά όταν προστίθενται τα συστατικά πρέπει να καθορίζεται η κατεύθυνση τους. Η εντολή :

```
setLayout(new BorderLayout());
add("North", new TextField("1997-98", 15));
add("South", new TextField("Τμ. Πληροφορικής", 30));
```

δημιουργεί την παρακάτω επιφάνεια σχεδίασης




- *CardLayout* : διαφέρουν από τους υπόλοιπους ρυθμιστές σχεδίασης διότι με τους ρυθμιστές της κλάσης CardLayout τα συστατικά σχεδίασης δεν εμφανίζονται καθόλου στην οθόνη. Οι CardLayout ρυθμιστές παράγουν ξεχωριστές επιφάνειες σχεδίασης που μπορούν να εμφανίζονται μία κάθε φορά, όπως στις παρουσιάσεις διαφανειών. Γενικότερα, χρησιμοποιούνται όταν χρειάζεται να κατασκευαστούν νέοι υποδοχείς συστατικών σχεδίασης ή νέες επιφάνειες σχεδίασης(panels). Δημιουργούνται οι "κάρτες" επιφανειών σχεδίασης οι οποίες μπορούν να εναλάσονται στην επιφάνεια σχεδίασης.

### 6.3.3. Αντιμετώπιση ενεργειών και γεγονότων

Οι προηγούμενες παράγραφοι αφορούν στη σχεδίαση διαφόρων συστατικών στην οθόνη τα οποία παρατίθενται με κάποια από τις περιγραφόμενες παραπάνω διατάξεις. Για να δοθεί έννοια στη χρήση και στην παρουσίαση των συστατικών σχεδίασης θα πρέπει να ανατεθεί κάποια ενέργεια ή πράξη η οποία να εκτελείται κατά την επιλογή ενός ή περισσοτέρων από τα συστατικά σχεδίασης της οθόνης. Ο έλεγχος της ενέργειας που πρέπει να εκτελεστεί κατά την επιλογή ενός εργαλείου προσαρμογής χρηστών πραγματοποιείται μέσω της διαχείρισης γεγονότων (Παράγραφος 6.1.) Ειδικότερα, τα σχεδιαστικά εργαλεία προσαρμογής χρηστών(UI interfaces) παράγουν μία ειδική κατηγορία γεγονότων την επονομαζόμενη "ενέργεια"(action). Ετσι χρειάζεται να οριστεί η αντίστοιχη μέθοδος action() στη μικροεφαρμογή ή στην κλάση που χρησιμοποιείται. Η γενική σύνταξη της action() είναι :

```
public boolean action(Event gegonos, Object param) {
. . .
}
```

Η παράμετρος *Object param* αφορά στη μέθοδο που συνοδεύει την ενέργεια που θα εκτελεστεί και εξαρτάται από το UI συστατικό που έχει χρησιμοποιηθεί. Κάθε ένα από τα βασικά UI συστατικά μπορούν να συνοδευτούν από διαφορετικές ενέργειες και παραμέτρους :

Buttons : δημιουργούν ενέργεια όταν επιλεγούν και η παράμετρος τους είναι η ετικέτα του πλήκτρου του τύπου Button.
Checkboxes : δημιουργούν ενέργεια όταν ένα κουτί επιλεχτεί. Η παράμετρος είναι πάντοτε true.
Choice menus : δημιουργούν ενέργεια όταν ένα επιλεχτεί μία από τις επιλογές του menu και παράμετρος είναι το αντικείμενο επιλογής.
Text Fields : δημιουργούν ενέργεια όταν ο χρήστης πατήσει το Enter για να τελειώσει την καταχώρηση του κειμένου στο συγκεκριμένο πεδίο. Η χρήση του πλήκτρου Tab δεν καταλήγει σε κάποια ενέργεια.

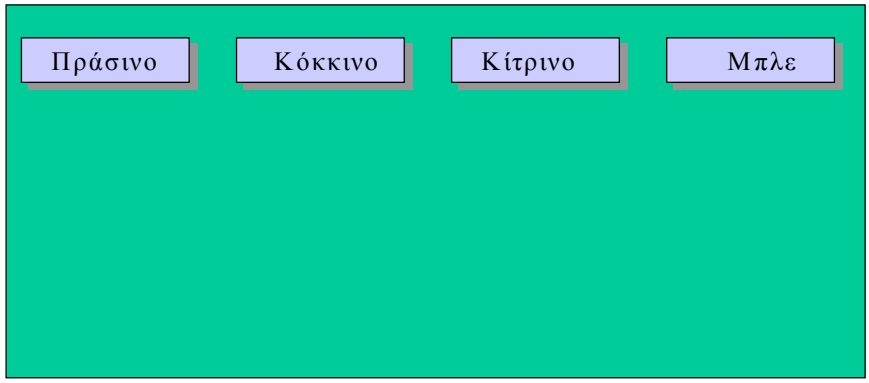
Για να γίνει η διαχείριση των διαφορετικών ενεργειών που προκύπτουν από τα διάφορα UI συστατικά, χρειάζεται μέσα στα όρια της μεθόδου action(), να γίνει έλεγχος του τύπου του αντικειμένου που δημιούργησε τη συγκεκριμένη ενέργεια. Ο έλεγχος αυτός γίνεται με τη μεταβλητή *target* του γεγονότος και τον τελεστή *instanceof* που θα επιστρέφει τον τύπο του UI αντικειμένου που προκάλεσε την ενέργεια. Στο παράδειγμα που ακολουθεί δημιουργείται μία επιφάνεια σχεδίασης που έχει 4 πλήκτρα που αντιστοιχούν σε ένα χρώμα το καθένα. Όταν ο χρήστης επιλέγει ένα πλήκτρο αλλάζει το χρώμα του φόντου σχεδίασης και γίνεται το χρώμα που δηλώνει το πλήκτρο.

*ΠΡΟΓΡΑΜΜΑ διαχείρισης ενεργειών*

```
import java.awt.*;

public class hrwmaButton extends java.applet.Applet{

    public void init() {
      setLayout(new FlowLayout());
      setBackground(Color.darkGray);

      add(new Button("Πράσινο"));
      add(new Button("Κόκκινο"));
      add(new Button("Κίτρινο"));
      add(new Button("Μπλε"));
  }
   public boolean action (Event geg, Object par) {
      if (geg.target instanceof Button)
      allagiHrwma((String) par);
      return true;
  }
  void allagiHrwma((String) hrwma) {
    if (hrwma.equals("Πράσινο")) setBackground(Color.green);
    else if (hrwma.equals("Κόκκινο")) setBackground(Color.red);
    else if (hrwma.equals("Κίτρινο")) setBackground(Color.yellow);
```

```
        else if (hrwma.equals("Μπλε")) setBackground(Color.blue);
    }
}
```

με αποτέλεσμα τη δημιουργία της παρακάτω επιφάνειας σχεδίασης :




### 6.3.4. Μικρο-εφαρμογές με χρήση του πακέτου AWT

Η ανάπτυξη μίας πλήρους μικρο-εφαρμογής που να περιλαμβάνει εργαλεία του πακέτου AWT, δεν είναι πάντοτε μία απλή υπόθεση. Χρειάζεται να καθοριστούν με τη σειρά τα βασικά χαρακτηριστικά και συστατικά σχεδίασης που θα περιλαμβάνει και στη συνέχεια πρέπει να χρησιμοποιηθούν οι κατάλληλες μέθοδοι. Τα γενικά βήματα μπορούν να συνοψιστούν στα παρακάτω στάδια ανάπτυξης :

- Δημιουργία της επιφάνειας σχεδίασης της μικρο-εφαρμογής. Πολλές φορές είναι χρήσιμη η πρόχειρη σχεδίαση (στο χαρτί) του μοτίβου της επιφάνειας σχεδίασης έτσι ώστε να είναι περισσότερο οργανωμένη η σχεδίαση της επιφάνειας μέσα από τις προγραμματιστικές εντολές.
- Δημιουργία της επιφάνειας σχεδίασης(panel)
- Ορισμός των υπο-επιφανειών σχεδίασης
- Διαχείριση των ενεργειών που θα επιτελούνται για κάθε UI συστατικό της επιφάνειας σχεδίασης.
- Ενημέρωση των αποτελεσμάτων σε περίπτωση μετατροπής κάποιων επιλογών.

Εκτός από όλα τα εργαλεία που περιγράφηκαν στις προηγούμενες παραγράφους το πακέτο AWT διαθέτει επιπλέον εργαλεία που βοηθούν στην ανάπτυξη αυτόνομων παραθυρικών εφαρμογών. Τα πρόσθετα αυτά εργαλεία περιλαμβάνουν την υποστήριξη παραθύρων, καταστάσεων επιλογών καθώς και μεθόδων διαλόγου.

- *Παράθυρα:* Η κλάση Window του AWT παρέχει τη δυνατότητα δημιουργίας παραθύρων ανεξάρτητων από τα παράθυρα του προγράμματος πλοήγησης που περιέχει τη μικρο-εφαρμογή. Η κλάση

Window περιλαμβάνει δύο βασικές υποκλάσεις : τη *Frame* με την οποία δημιουργείται ένα πλήρες λειτουργικό παράθυρο και την Dialog που δημιουργεί πιο περιορισμένα παράθυρα με τη μορφή κουτιών διαλόγου. Υπάρχουν δύο μέθοδοι κατασκευής ενός παραθύρου τύπου Frame :

new Frame( ) : δημιουργεί ένα βασικό πλαίσιο παραθύρου χωρίς τίτλο

new Frame(String) : δημιουργεί ένα βασικό πλαίσιο παραθύρου με τίτλο τη συμβολοσειρά τύπου String.

Στη συνέχεια προστίθενται τα συστατικά στοιχεία του παραθύρου με τον ίδιο τρόπο όπως και στις επιφάνειες σχεδίασης(panels) δηλαδή με τη μέθοδο add(). Οταν αρχικά δημιουργείται το παράθυρο είναι αόρατο και χρειάζεται η μέθοδος show() για να εμφανιστεί το παράθυρο στην οθόνη, ενώ με την hide() το παράθυρο εξαφανίζεται και πάλι από την οθόνη. Για μετατροπή του μεγέθους του παραθύρου χρησιμοποιείται η μέθοδος resize() ενώ για τον καθορισμό της θέσης του παραθύρου μπορεί να χρησιμοποιηθεί η μέθοδος move(). Ακόμα, υπάρχει η μέθοδος location() που παρέχει πληροφορία για τη θέση που κατέχει το παράθυρο της μικρο-εφαρμογής στην οθόνη.

- <u>*Καταστάσεις επιλογών:*</u>. Κάθε νέο παράθυρο μπορεί στη συνέχεια να αποκτήσει τη δική του λίστα επιλογών (menu) και τη ράβδο διολίσθησης των επιλογών(menubar). Το AWT διαθέτει τις κλάσεις MenuBar, Menu και MenuItem για τη διαχείριση των επιλογών ενός παραθύρου. Η επιλογή του χρήστη κάποιας από τις δυνατές επιλογές του menu προκαλεί τη δημιουργία ενός γεγονότος της αντίστοιχης ενέργειας και η διαχείριση του γίνεται και πάλι με τη μέθοδο action().

- <u>*Διαλογική υποστήριξη :*</u> Παρέχεται με τη σχεδίαση απλούστερων παραθύρων που δίνουν τη δυνατότητα εισαγωγής δεδομένων και σύντομων απαντήσεων από το χρήστη. Το AWT παρέχει δύο κλάσεις Dialog και την File Dialog η οποία παρέχει ένα διαλογικό παράθυρο προσαρμοσμένο στην αντίστοιχη πλατφόρμα που αφορά στο αρχείο που θα σωθεί ή θα ανοιχτεί. Η δημιουργία ενός διαλογικού παραθύρου γίνεται με τις μεθόδους :

  Dialog(Frame, boolean) : δημιουργεί ένα μη ορατό διαλογικό παράθυρο που προσαρμόζεται στο συγκεκριμένο παραθυρικό πλαίσιο.

  Dialog(Frame, String, boolean): δημιουργεί ένα μη ορατό διαλογικό παράθυρο που προσαρμόζεται στο συγκεκριμένο παραθυρικό πλαίσιο με τίτλο τη συμβολοσειρά τύπου String.

Μπορεί να κατασκευαστεί διαλογικό παράθυρο μόνο σε ήδη υπάρχοντα πλαίσια παραθύρων. Το διαλογικό παράθυρο γίνεται ορατό

με τη μέθοδο show() και αποκρύπτεται από την οθόνη με τη μέθοδο hide().

Αντίστοιχη είναι και η δημιουργία των διαλογικών παραθύρων αρχείου τα οποία διαθέτουν τις μεθόδους κατασκευής FileDialog που είναι αντίστοιχες με τις παραπάνω Dialog μεθόδους. Τα διαλογικά πλαίσια αρχείων διαθέτουν ακόμα τις βασικές μεθόδους getDirectory() και getFile() που επιστρέφουν συμβολοσειρές με το όνομα του καταλόγου και του αρχείου που αναζητά ο χρήστης.

# 7. ΠΡΟΧΩΡΗΜΕΝΕΣ ΛΕΙΤΟΥΡΓΙΕΣ ΣΤΗΝ JAVA

## 7.1. Υποστήριξη δικτυακών λειτουργιών

Με την υποστήριξη δικτυακών λειτουργιών δίνεται η δυνατότητα δια-σύνδεσης μίας μικρο-εφαρμογής με άλλες εφαρμογές που μπορεί να βρίσκονται σε ένα οποιοδήποτε σύστημα του δικτύου. Η Java παρέχει κατάλληλες κλάσεις για την υποστήριξη των δικτυακών λειτουργιών, μέσω του πακέτου της java.net που αφορά δια-πλατφορμικές ενέργειες δια-σύνδεσης και προσπέλασης με χρήση των πλέον διαδεδομένων Web πρωτοκόλλων. Με τα εργαλεία αυτά η Java κάνει την ανάγνωση και εγγραφή από/σε αρχεία του δικτύου, μία απλή διαδικασία, αντίστοιχη του τοπικού συστήματος. Υπάρχουν βέβαια κάποιοι περιορισμοί που αφορούν κυρίως στην ασφάλεια και στην προστασία των μεταφερόμενων μέσω του δικτύου πληροφοριών.

Η επικοινωνία με συστήματα του δια-δικτύου περιλαμβάνει εργαλεία όπως :

showDocument( ) : η μέθοδος που επιτρέπει στην μικρο-εφαρμογή να αποκτήσει σύνδεση με κάποια www σελίδα.

openStream( ): η μέθοδος που ξεκινά μία σύνδεση με ένα αντικείμενο URL και επιτρέπει τη συλλογή πληροφοριών από αυτή τη σύνδεση.

Socket και ServerSocket : κλάσεις που επιτρέπουν τη διάνοιξη μίας τυπικής υποδοχής σύνδεσης με άλλους εξυπηρέτες του δια-δικτύου, μέσω των οποίων μπορεί να γίνει ανάγνωση ή εγγραφή πληροφορίας.

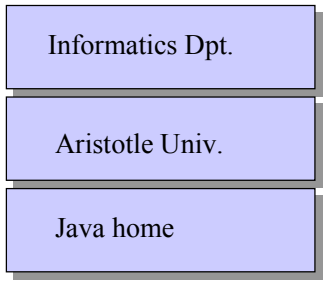
Η δικτυακή υποστήριξη μέσα σε μία μικρο-εφαρμογή γίνεται με τον καθορισμό της www σελίδας που χρειάζεται να φορτωθεί. Η σύνδεση με κάποια www σελίδα γίνεται με τη δημιουργία μίας στιγμής ύπαρξης της κλάσης URL. Η κλάση URL αναπαριστά την διαδρομή εντοπισμού των απαιτούμενων πόρων του δια-δικτύου. Για να δημιουργηθεί ένα νέο URL μπορεί να χρησιμοποιηθεί μία από τις παρακάτω μεθόδους :

URL(String, String, int, String) : δημιουργεί ένα URL αντικείμενο που οι παράμετροι του είναι συμβολοσειρές που εκφράζουν το πρωτόκολλο (http,ftp,gopher,file), το όνομα του διαθέτη(host), τον αριθμό του πόρου σύνδεσης(π.χ. 80 για το http), και συμβολοσειρά με το όνομα του αρχείου ή της διαδρομής που θα φορτωθεί.

URL(String, String, String) : δημιουργεί ένα URL αντικείμενο με παρεχόμενες όλες τις παραπάνω πληροφορίες εκτός από τον αριθμό του πόρου σύνδεσης.

URL(URL, String) : δημιουργεί ένα URL αντικείμενο με τη βασική διαδρομή και με μία επιπλέον σχετική διαδρομή. Για τη βασική διαδρομή μπορεί να χρησιμοποιηθεί η getDocumentBase() που καθορίζει το URL του αρχείου κλάσης της μικρο-εφαρμογής.

URL(String) : δημιουργεί ένα URL αντικείμενο με βάση τη συμβολοσειρά δήλωσης η οποία πρέπει να περιλαμβάνει το είδος του πρωτοκόλλου, το όνομα του διαθέτη, τον αριθμό του πόρου σύνδεσης(προαιρετικά) και το όνομα του αρχείου.

| *ΠΡΟΓΡΑΜΜΑ* | *ΑΠΟΤΕΛΕΣΜΑ* |
|---|---|
| import java.awt.*;<br>import java.net.URL;<br>import java. net.MalformedURLException;<br><br>public class klisiWpage extends java.applet.Applet {<br>Bookmark bl[] = new Bookmark[3];<br><br>public void init() {<br>bl[0] =new Bookmark("Informatics Dpt",<br>"http://www.csd.auth.gr");<br>bl[1] =new Bookmark("Aristotle Univ.",<br>"http://www.ccf.auth.gr");<br>bl[2] =new Bookmark("Java home",<br>"http://java.sun.com");<br>setLayout(new GridLayout(bl.length,1));<br>for (int i=0; i< bl.length;i++)<br>add(new Button(bl[i].name));<br>}<br>public boolean action(Event geg, Object p) {<br>if (geg.target instanceof Button ) {<br>syndesi((String) p);<br>return true;<br>}<br>}<br>void syndesi(String onoma) {<br>URL toURL =null;<br>for (int i=0; i< bl.length;i++) {<br>if (onoma.equals(bl[i].name))<br>toURL = bl.url;<br>}<br>if (toURL !=null)<br>getAppletContext().showDocument(toURL ) ;<br>}<br>class Bookmark {<br>String onoma;<br>URL url; |  |

```
Bookmark(String onoma, String toURL) {
thiw.onoma = onoma;
      try {this.url = new URL(toURL);}
      catch (MalformedURLException e) {
        System.out.println("Λάθος URL"+toURL);
      }
}
}
```

Στο παραπάνω πρόγραμμα εμφανίζονται τρία πλήκτρα επιλογής καθένα από τα οποία όταν επιλεγεί , φορτώνει μία www σελίδα. Η δήλωση ενός URL αντικειμένου στο πρόγραμμα πλοήγησης γίνεται με την εντολή :

*getAppletContext().showDocument(το ζητούμενο URL);*

με την οποία το πρόγραμμα πλοήγησης που περιλαμβάνει το ζητούμενο URL θα φορτώσει και θα εμφανίσει το αρχείο σε αυτό το URL.

- Για λόγους ασφάλειας οι μικρο-εφαρμογές έχουν τη δυνατότητα σύνδεσης μόνο με τον ίδιο διαθέτη από τον οποίο φορτώθηκαν αρχικά. Εάν χρειάζεται να φορτωθεί κάτι από το δίκτυο χρειάζεται μα γίνει σύνδεση με URL. Η κλάση URL διαθέτει τη μέθοδο openStream() η οποία ανοίγει τη δικτυακή σύνδεση χρησιμοποιώντας το δεδομένο URL και επιστρέφει μία στιγμή ύπαρξης της κλάσης InputStream() που ανήκει στο πακέτο java.io. Εάν στη συνέχεια μετατραπεί αυτή η ροή δεδομένων σε DataInputStream() μπορεί να πραγματοποιηθεί ανάγνωση χαρακτήρων και γραμμών από τη ροή αυτή των στοιχείων.

  Η μέθοδος openStream() αποτελεί απλοποίηση της κλάσης URLconnection. Η κλάση URLconnection δίνει τη δυνατότητα ανάγνωσης αρχείων με χρήση URL και ακόμα επιτρέπει την έξοδο ροής στοιχείων εάν το επιτρέπει το συγκεκριμένο πρωτόκολλο. Για να χρησιμοποιηθεί η URL σύνδεση ακολουθείται η παρακάτω διαδικασία :

  1.δημιουργία μίας στιγμής ύπαρξης της κλάσης URLconnection
  2.καθορισμός των παραμέτρων της
  3.χρήση της μεθόδου connect() για να ανοίξει η σύνδεση

- Επιπλέον η Java παρέχει δύο κλάσεις την Socket και την ServerSocket οι οποίες προσομοιάζουν τις τυπικές προγραμματιστικές τεχνικές τύπου Socket. Η κλάση Socket παρέχει μία δίοδο που αποτελεί ένα ενδιάμεσο εξυπηρετούμενου(client) τύπου Socket που προσομοιάζει με τις τυπικές Unix sockets. Για να ανοίξει μία σύνδεση δημιουργείται μία στιγμή ύπαρξης της Socket με παραμέτρους το όνομα του διαθέτη και τον αριθμό της πόρτας επικοινωνίας. Αφού ανοίξει η δίοδος γίνεται δυνατή η ροή δεδομένων εισόδου-εξόδου διαμέσου αυτής της διόδου (Socket). Αντίστοιχα, δημιουργούνται και οι δίοδοι τύπου εξυπηρέτη(ServerSocket) οι οποίες περιμένουν από μία πόρτα τύπου TCP για σύνδεση από κάποιον εξυπηρετούμενο. Χρησιμοποιώντας διόδους και των δύο τύπων

(εξυπηρέτη και εξυπηρετούμενου) δημιουργούνται εφαρμογές που μπορούν να επικοινωνούν μεταξύ τους μέσω του δικτύου.

## 7.2. Προπορευόμενοι τελεστές κλάσεων

Η Java διαθέτει ένα σύνολο προπορευόμενων τελεστών που τροποποιούν την πρόσβαση και την επικοινωνία μεταξύ των κλάσεων. Ολοι οι υποστηριζόμενοι τροποποιητές είναι προαιρετικής χρήσης αλλά είναι χρήσιμο να δηλώνονται όπου είναι απαραίτητο διότι διευκολύνουν την επικοινωνία των κλάσεων και ταυτόχρονα παρέχουν ένα είδος προστασίας κατά την ανταλλαγή δεδομένων. Η πρόσβαση σε μία κλάση έχει να κάνει με τον έλεγχο της "ορατότητας" των στοιχείων της κλάσης από άλλες κλάσεις. Οταν μία μέθοδος ή μία μεταβλητή είναι ορατή από κάποια άλλη κλάση, οι μέθοδοι αυτής της κλάσης μπορούν να επηρεάσουν αυτήν την ορατή μέθοδο ή μεταβλητή. Επομένως για να προστατευθεί μία μέθοδος ή μία μεταβλητή από τυχόν ανεπιθύμητες προσβάσεις υπάρχουν 4 επίπεδα ορατότητας : public, package, protected, και private (γνωστά ως 4 p).

- *public* : αυτό το είδος πρόσβασης κάνει τις μεθόδους και τις μεταβλητές μίας κλάσης ορατές από όλες τις άλλες κλάσεις. Είναι καλή η τακτική να ξεκινάει κανείς δηλώνοντας public τις κλάσεις και στη συνέχεια να τις περιορίζει με βάση τις ανάγκες της εφαρμογής που αναπτύσσει. Οι δηλώσεις είναι της μορφής :

```
public class miaPublicklasi {
        public String publicLexi;
        public int publicAkeraios;

        public boolean publicMethodos() { . . .
        }
}
```

Μία μέθοδος ή μία μεταβλητή που έχει δηλωθεί public έχει τη μέγιστη δυνατή ορατότητα και όλοι μπορούν να τη χρησιμοποιούν.

- *package*: εκφράζει το επόμενο στάδιο πρόσβασης(μετά την public) και δεν υπάρχει συγκεκριμένος τελεστής που να δηλώνεται για αυτό το είδος πρόσβασης.Υποστηρίζει την πρόσβαση και την ανταλλαγή πληροφοριών μεταξύ των κλάσεων που ανήκουν στην ίδια ομάδα. Οταν ένα μεγάλο σύστημα διαχωρίζεται σε έναν αριθμό ομάδων κλάσεων που συνεργάζονται υπάρχει μεγαλύτερη ανάγκη ενδο-επικοινωνίας μέσα σε μία ομάδα συνεργασίας από ότι εκτός της ομάδας.

```
public class neaPublicklasi {
        String packageLexi="ΠΛΗΡΟΦΟΡΙΚΗ";
        int packageAkeraios=50;
```

```
        boolean packageMethodos() { . . .
        }
    }
    public class klasiIdiouPaketou {
        public void dokimi() {
            neaPublicklasi k = new neaPublicklasi();

            System.out.println(k.packageLexi+ k.packageAkeraios);.
        }
    }
```

- *protected*: Καθορίζει το συσχετισμό μεταξύ μίας κλάσης και των υπαρχόντων αλλά και των μελλοντικών υπο-κλάσεων της. Ο τελεστής αυτός καθορίζει τη σχέση που έχουν οι υποκλάσεις με τις γονικές τους κλάσεις. Περιορίζει ("προστατεύει") την πρόσβαση στην κλάση από εξωτερικές κλάσεις ενώ παρέχει πλήρη δικαιώματα στις υπο-κλάσεις της.

```
public class miaProtectedKlasi {
    private protected String protectedLexi="ΠΛΗΡΟΦΟΡΙΚΗ";
    private protected int protectedAkeraios=50;

    private protected boolean protectedMethodos() { . . .
    }
}

public class ypoklasi extends miaProtectedKlasi {
    public void dokimi() {
        miaProtectedKlasi  k = new  miaProtectedKlasi();

        System.out.println(k.protectedLexi + k.protectedAkeraios);.
        k.protectedMethodos();
    }
}
```

Στην παραπάνω κλάση *miaProtectedKlasi* οι μεταβλητές και οι μέθοδοι έχουν δηλωθεί ως private protected, γεγονός που δεν επιτρέπει την πρόσβαση από οποιαδήποτε άλλη (μη υπο-κλάση) κλάση του ίδιου πακέτου. Επομένως, οι παρακάτω εντολές αναφοράς στις μεταβλητές και στις μεθόδους δεν είναι επιτρεπτές

```
    public class alliKlasiIdiouPaketou{
        public void dokimi() {
        miaProtectedKlasi  k = new  miaProtectedKlasi();

        System.out.println(k.protectedLexi + k.protectedAkeraios);.
        k.protectedMethodos();
    }
```

Εάν είχε χρησιμοποιηθεί μόνο ο τελεστής protected θα επιτρεπόταν η πρόσβαση από τις υπο-κλάσεις αλλά και από τις άλλες κλάσεις του ίδιου πακέτου.

- *private*: αποτελεί τον πλέον περιορισμένης ορατότητας τελεστή, με τη μέγιστο δυνατό επίπεδο προστασίας, δηλαδή είναι διαμετρικά αντίθετο με τον τελεστή public. Οι μεταβλητές και οι μέθοδοι που δηλώνονται ως private δεν μπορούν να προσπελαθούν από καμία άλλη κλάση εκτός από αυτήν στην οποία ορίζονται. Οι δηλώσεις είναι της μορφής :

```
public class miaPrivateklasi {
        privateString privateLexi;
        privateint privateAkeraios;

        private boolean privateMethodos() { . . .
        }
}
```

Η δήλωση private συνοδεύει όλα τα στοιχεία που χρειάζεται να αφορούν στη συγκεκριμένη κλάση και να προστατεύονται από την προσπέλαση άλλων κλάσεων. Ενας χρήσιμος κανόνας είναι να δηλώνονται οι στιγμιαίες μεταβλητές private εκτός εάν πρόκειται για σταθερές. Αυτό χρειάζεται να γίνεται διότι διαφορετικά υπάρχει δυνατότητα πρόσβασης και αλλαγής μεταβλητών από άλλες κλάσεις με ανεπιθύμητα αποτελέσματα. Γιά παράδειγμα εάν υπάρχει μία κλάση :

```
public class dokimi {
public String keimeno;
// όπου το keimeno είναι το μήνυμα του Προέδρου του Τμήματος
}
```

στην οποία η μεταβλητή keimeno δεν είναι private, μπορεί από οποιαδήποτε κλάση να χρησιμοποιηθεί και να τροποποιηθεί το περιεχόμενο της μεταβλητής keimeno. Το πρόβλημα είναι ότι δεν υπάρχει μηχανισμός διάκρισης της διαδικασίας ανάγνωσης ή εγγραφής νέου περιεχομένου σε μία μεταβλητή και επομένως η δήλωση private είναι απαραίτητη για την προστασία των μεταβλητών. Μία εναλλακτική λύση είναι η ανάπτυξη μεθόδων που θα διακρίνουν τη διαδικασία της ανάγνωσης από τη διαδικασία της εγγραφής. Ετσι μπορεί να γραφεί μία public μέθοδος που θα επιστρέφει την τιμή μίας μεταβλητής και μία protected μέθοδο για να καθορίζει την τιμή της.

### 7.2.1. Ο τελεστής final

Η χρήση του τελεστή final αποτελεί καθοριστικό παράγοντα που μπορεί να καθορίσει τα παρακάτω :

- όταν προπορεύεται της δήλωσης μίας κλάσης, σημαίνει ότι δεν μπορεί η συγκεκριμένη κλάση να αποκτήσει υπο-κλάσεις.

- όταν προπορεύεται της δήλωσης μίας μεταβλητής, σημαίνει ότι η μεταβλητή αυτή αποτελεί σταθερή.

- όταν προπορεύεται της δήλωσης μίας μεθόδου, σημαίνει ότι η μέθοδος δεν μπορεί να παραγραφεί από τις υπο-κλάσεις της.

*final κλάσεις* : Η κλάση final δηλώνεται κυρίως για λόγους ασφάλειας και ευελιξίας του αναπτυσσόμενου κώδικα. Η δήλωση μίας κλάσης final γίνεται :

```
public final class miaFinalKlasi {
. . .
}
```

*final μεταβλητές* : Οι μεταβλητές αυτού του τύπου δεν μπορούν να μετατραπούν και πρέπει να τους δοθεί η σταθερή τιμή τους κατά τη δήλωση τους. Οι τοπικές μεταβλητές (που συνήθως δηλώνονται στα όρια των βρόγχων while, for) δεν μπορούν ποτέ να δηλωθούν final. Η δήλωση μίας μεταβλητής final γίνεται :

```
public class neaFinalKlasi {
public final String keimeno=”Τμήμα Πληροφορικής”;
 public static final int akeraios = 1997;
}
```

*final μέθοδοι*: Οι μέθοδοι που δηλώνονται ως final δεν μπορούν να τροποποιηθούν από τις υπο-κλάσεις της κλάσης στην οποία περιέχονται. Η δήλωση μίας μεθόδου final γίνεται :

```
public class alliFinalKlasi {
public static final void method1() { …
}
 public final void method2() { …
}
}
```

Οι final μέθοδοι παρέχουν ευελιξία και επιτάχυνση στην εκτέλεση του κώδικα διότι μπορούν να κληθούν άμεσα και με ασφαλή τρόπο, παραμένουν σταθερές και δεν επηρεάζεται η δήλωση τους.

### 7.2.2. Ο τελεστής abstract

Οι αρχές της ιεραρχίας και κληρονομικότητας των κλάσεων επιβάλλουν τη γενίκευση και την αφαιρετικότητα στις ανώτερες ιεραρχικά κλάσεις. Ειδικότερα, όταν μία υπερ-κλάση δηλώνεται abstract αποτελεί τη γενική διαμοιραζόμενη “αποθήκη” δηλώσεων και είναι αναμενόμενο ότι θα

χρησιμοποιηθούν κυρίως οι υπο-κλάσεις της. Δεν μπορούν να δημιουργηθούν στιγμές ύπαρξης μίας κλάσης τύπου abstract αλλά μόνο να δημιουργούνται υπο-κλάσεις της. Γενικά, η κλάση τύπου abstract αποτελεί τη βάση από την οποία ξεκινά η δημιουργία των υπόλοιπων κλάσεων. Η δήλωση μίας κλάσης abstract γίνεται :

```
public abstract class abstractKlasi {
        int metavliti;
        public abstract int method1();
                        // θα αναπτυχθεί από τις μη-abstract υπο-κλάσσεις

        public void method2() { … // μία συνηθισμένη μέθοδος
        }
}
```

Στο σώμα δήλωσης μίας κλάσης τύπου abstract μπορεί να έχουμε και δήλωση μεθόδων abstract. Μέθοδοι abstract επιτρέπεται να ορίζονται μόνο στο σώμα δήλωσης μίας κλάσης abstract. Δεν μπορεί να υπάρξει εντολή της μορφής :

*Object o = new abstractKlasi();*

διότι η abstractKlasi είναι τύπου abstract. Επομένως η χρήση των κλάσεων abstract επιβάλλει τη δημιουργία μίας υπο-κλάσης της κλάσης abstract και στη συνέχεια χρήση αυτής της υπο-κλάσης. Για παράδειγμα, δημιουργείται η :

```
public class abstractYpoKlasi extends abstractKlasi {
        int metavliti;
        // ακολουθεί η δήλωση της μεθόδου τύπου abstract της αρχικής abstractKlasi
        public int method1() { …
        }
}
```

και στη συνέχεια επιτρέπονται εντολές της μορφής :

*Object o = new abstractYpoKlasi();*

## 7.3. Πακέτα και προσαρμοστικά εργαλεία

### 7.3.1. Τα Πακέτα στην Java

Η έννοια του πακέτου στην Java είναι ιδιαίτερα χρήσιμη για να υπάρχει σύνδεση και συσχετισμός κάποιων κλάσεων που αφορούν σε κοινή λειτουργία και χρήση. Τα πακέτα εκφράζουν τον τρόπο με τον οποίο η Java πραγματοποιεί σχεδιασμό και οργάνωση μεγάλης και μικρής κλίμακας. Γενικά, τα πακέτα χρησιμοποιούνται τόσο για την ομαδοποίηση όσο και για την κατηγοριοποίηση των κλάσεων. Οι κλάσεις ενός πακέτου δεν είναι απαραίτητο να σχετίζονται και ιεραρχικά στο δέντρο της ιεραρχίας των

κλάσεων της εφαρμογής. Η δήλωση για τη δημιουργία ενός πακέτου είναι της μορφής :

```
package paketo1;
public class miaPublicClass extends alliKlasi { …
}
```

Η εντολή δήλωσης του πακέτου πρέπει πάντοτε να είναι η πρώτη εντολή στο αρχείο δήλωσης και όλες οι κλάσεις που δηλώνονται στη συνέχεια της ομαδοποιούνται και περιλαμβάνονται στο συγκεκριμένο πακέτο. Στη συνέχεια μπορεί να υπάρχει κάποια υπο-ομαδοποίηση στο πακέτο με κάποιο είδος ιεραρχίας του πακέτου. Για παράδειγμα, η κλάση βιβλιοθήκης της Java αποτελεί ένα πακέτο στο οποίο το ανώτερο επίπεδο καλείται java και αποτελείται από υπο-επίπεδα τα net, io, util και awt. Στη συνέχεια το awt διαθέτει επιπλέον υπο-επίπεδα όπως π.χ. το πακέτο του image κ.ο.κ.

Κάθε κλάση της Java συνήθως αποθηκεύεται σε ξεχωριστό αρχείο και η ομαδοποίηση των κλάσεων ενός πακέτου είναι αντίστοιχη με την ιεραρχία και τη δόμηση των αρχείων σε καταλόγους και υπο-καταλόγους. Ο μεταφραστής της Java επιβάλλει τη δημιουργία καταλόγων κάτω από τον κατάλογο της αρχικής κλάσης του πακέτου, έτσι ώστε να διατηρείται η ιεραρχία των υπο-πακέτων του με τη δήλωση των κατάλληλων κλάσεων στη θέση του καταλόγου που αφορά στο πακέτο στο οποίο ανήκουν. Για παράδειγμα, η ιεραρχία των καταλόγων της κλάσης βιβλιοθήκης της Java εκφράζει ακριβώς την αντίστοιχη ιεραρχία των πακέτων της. Αρα υπάρχει ο κατάλογος …/classes/java/awt/image/ που περιλαμβάνει τις απαραίτητες κλάσεις για τη διαχείριση των αρχείων εικόνων. Εάν έχει ήδη δημιουργηθεί ένα πακέτο(paketo1), μπορεί να δημιουργηθεί στη συνέχεια ένα υπο-πακέτο του (paketo2) με τις εντολές :

```
package paketo1.paketo2;
public class alliPublicClass extends alliKlasi { …
}
```

και το αρχείο δήλωσης του νέου πακέτου θα πρέπει να έχει το όνομα *alliPublicClass.java* και να βρίσκεται σε ένα κατάλογο κάτω από τον κατάλογο του πρώτου πακέτου δηλαδή θα πρέπει να βρίσκεται στον κατάλογο …/classes/paketo1/paketo2 έτσι ώστε να μπορεί ο μεταφραστής javac να το βρει.

Οταν υπάρχει αναφορά σε μία κλάση με το όνομα της μέσα από το κώδικα του Java προγράμματος, στην πραγματικότητα χρησιμοποιείται ένα πακέτο. Οι συχνότερα χρησιμοποιούμενες κλάσεις ανήκουν στο πακέτο java.lang που εισάγει αυτόματα ο μεταφραστής της Java. Για παράδειγμα, η κλάση String αποτελεί μέρος του υπο-πακέτου java.lang. Οταν χρειάζεται να γίνει αναφορά σε μία κλάση που ορίζεται σε ένα πακέτο χρησιμοποιείται το όνομα του πακέτου ακολουθούμενο από το όνομα της κλάσης, όπως για το παραπάνω παράδειγμα μπορεί να υπάρχει η κλήση *paketo1.miaPublicClass*. Γενικά, το όνομα ενός πακέτου μπορεί να εισαχθεί και να υπάρχουν αναφορές σε αυτό

μέσα από τον κώδικα της εφαρμογής. Η εισαγωγή του πακέτου γίνεται με χρήση της εντολής import, η οποία γενικά πρέπει να ακολουθεί την package και να προηγείται των εντολών δήλωσης των κλάσεων. Ιδιαίτερη μέριμνα χρειάζεται για να πραγματοποιηθεί εισαγωγή όλων των απαιτούμενων υπο-καταλόγων των υπο-πακέτων που χρειάζεται η εφαρμογή διότι η εντολή *import java.awt.** δε συνεπάγεται την εισαγωγή των υπο-πακέτων.

| *Δηλώσεις πακέτων* | |
|---|---|
| package paketoA;<br><br>public class onomaKlassis { …<br>}<br>public class klassiA { …<br>} | package paketoB;<br><br>public class onomaKlassis { …<br>}<br>public class klassiB { …<br>} |
| ΛΑΘΟΣ ΧΡΗΣΗΣ ΠΑΚΕΤΩΝ<br>import paketoA.*;<br>import paketoB.*;<br>onomaKlassis antikeimeno; | ΣΩΣΤΗ ΧΡΗΣΗ ΠΑΚΕΤΩΝ<br>import paketoA.*;<br>import paketoB.*;<br>paketoA.onomaKlassis antikeimeno1;<br>paketoB.onomaKlassis antikeimeno2; |

### 7.3.2. Προσαρμοστικά πρότυπα και εργαλεία στην Java

Αποτελούν πρότυπα συμπεριφοράς των κλάσεων αντίστοιχα των abstract κλάσεων και μεθόδων, με τη διαφορά ότι τα προσαρμοστικά πρότυπα (interfaces) είναι περισσότερο ισχυρά. Η ιεραρχία των κλάσεων που περιγράφηκε, πολλές φορές είναι περιοριστική. Για να ξεπερασθούν οι περιορισμοί που επιβάλλει η ιεραρχία των κλάσεων, η Java διαθέτει μία επιπλέον υποστήριξη ιεραρχίας, την ιεραρχία των interfaces. Τα interfaces δεν περιορίζονται στην δυνατότητα ύπαρξης μίας μόνο υπερ-κλάσης αλλά επιτρέπουν την πολλαπλή κληρονομικότητα, επιτρέποντας όμως μόνο το πέρασμα της περιγραφής των μεθόδων στους απογόνους και όχι την υλοποίηση των μεθόδων και των στιγμιαίων μεταβλητών.

Τα interfaces ενισχύουν και επεκτείνουν την ισχύ των κλάσεων. Οπως και οι κλάσεις ορίζονται σε αρχεία πηγαίου κώδικα, ένα interface ανά αρχείο. Μεταφράζονται σε .class αρχεία όπως και οι κλάσεις, με τη βασική τους διαφορά στο ότι ένα interface δεν μπορεί να αρχικοποιηθεί, η λέξη-κλειδί new μπορεί να ακολουθείται μόνο από τη στιγμή ύπαρξης μίας κλάσης. Η δήλωση ενός interface γίνεται ως εξής :

```
package paketo1;
public interface enaInterface extends interface1, interface2, . . . {
 …
// όλες οι μέθοδοι εδώ πρέπει να είναι public ή abstract
```

```
// όλες οι μεταβλητές πρέπει να είναι public, static, και final
}
```

Στην παραπάνω δήλωση καταγράφεται η πολλαπλή κληρονομικότητα με τη σειρά δήλωσης διαφόρων interfaces που ακολουθούν τη λέξη-κλειδί extends.

Η πλέον ισχυρή δυνατότητα που δίνουν τα interfaces στην Java είναι η δυνατότητα διαχωρισμού της κληρονομικότητας σχεδιασμού από την κληρονομικότητα υλοποίησης. Η ξεχωριστή κληρονομικότητα των interfaces δίνει τη δυνατότητα παρεμβολής στο δέντρο της αρχικής κληρονομικότητας των κλάσεων. Για παράδειγμα η δήλωση :

```
class prwtiPeriptwsi extends Klassi implements interface2 {…
}
```

αφορά στο αρχικό δέντρο κληρονομικότητας των κλάσεων κάτω από την κλάση *Klassi* και ταυτόχρονα ακολουθεί την υλοποίηση του *interface2,* ενώ η δήλωση :

```
class deuteriPeriptwsi  implements interface1, interface2 {…
}
```

ακολουθεί την υλοποίηση των *interface1, interface2*

Το απλό δέντρο της ιεραρχίας των κλάσεων περιλαμβάνει την απλή κληρονομικότητα υλοποίησης ενώ η ιεραρχία της σχεδίασης εκφράζεται από το ιεραρχικό δέντρο των interfaces.

*Προσαρμοστικά Πρότυπα (interfaces)*

```
interface programa extends spoudes { …
}
class mathima extends ptyhio implements programa { …
}
interface thewria { …
}
class ergastirio extends mathima implements thewria {…
}
```

Στο παραπάνω παράδειγμα η κλάση *ergastirio* δε χρειάζεται να δηλώσει *implements programa* διότι αυτό ήδη πραγματοποιείται λόγω του ότι επεκτείνει το *mathima* που κάνει implements programa. Η γενική ιδέα της ανάμιξης ιεραρχιών είναι ότι επιτρέπεται σε διάφορες κλάσεις που είναι διασκορπισμένες σε ένα δέντρο απλής κληρονομικότητας να υλοποιούν και να χρησιμοποιούν το ίδιο σύνολο μεθόδων. Οταν μία μεταβλητή έχει δηλωθεί ως τύπου interface, τότε κάθε αντικείμενο που αφορά σε αυτή η μεταβλητή υλοποιεί το συγκεκριμένο interface, δηλαδή έχει στη διάθεση του όλες τις μεθόδους που καθορίζει το συγκεκριμένο interface. Στην πράξη, τα interfaces υλοποιούνται και χρησιμοποιούνται στην κλάση βιβλιοθήκης της Java όταν κάποιες καταστάσεις ή συμπεριφορές αναμένεται να υιοθετηθούν από έναν αριθμό ανόμοιων και ασύνδετων κλάσεων.

## 7.4. Χρήση εξαιρέσεων στην Java

Ο μηχανισμός της εξαίρεσης(exception) είναι ο τρόπος που διαθέτει η Java για τον εντοπισμό και την αναφορά σφαλμάτων. Η θεωρία υποστήριξης της επιστροφής σφαλμάτων θεωρεί ότι κάθε συνάρτηση πρέπει να επιστρέφει μία ένδειξη σφάλματος και ακόμα θα πρέπει να ελέγχει τις ενδείξεις των σφαλμάτων που επιστρέφονται. Ετσι, για να υπάρχει δυνατότητα σύνταξης εύκολα αναγνώσιμου και λειτουργικού κώδικα, χρησιμοποιούνται οι εξαιρέσεις για να διαχειρίζονται τις ανεπιθύμητες καταστάσεις ενός προγράμματος. Μία εξαίρεση είναι κάθε αντικείμενο που αποτελεί μία στιγμή ύπαρξης της κλάσης *Throwable* ή οποιασδήποτε υπο-κλάσης της.

Μετά την κατασκευή πολύπλοκων προγραμμάτων στην Java είναι αναμενόμενο μετά τη σχεδίαση των κλάσεων και των interfaces και του καθορισμού της συμπεριφοράς τους, να υπάρχει και μέριμνα για τη συμπεριφορά του προγράμματος σε περίπτωση εμφάνισης σφαλμάτων. Η εξαίρεση είναι μία κατάσταση σφάλματος που διακόπτει τη ροή του προγράμματος και συμβαίνει όταν το πρόγραμμα εκτελεί τη λέξηκλειδί throw. Η εντολή throw μεταβιβάζει τον έλεγχο του προγράμματος σε ένα κομμάτι "σύλληψης" (catch block) που είναι ικανό να χειριστεί το σφάλμα. Η γενική δήλωση είναι :

```
public class miaExairesiClass {
    public void methodosExairesis() throws Exairesi { . . .
    }
}
```

με την οποία δηλώνεται στον μεταφραστή ότι ο κώδικας που περιέχεται μέσα στην *methodosExairesis*() μπορεί να προκαλέσει την εξαίρεση με όνομα *Exairesi*. Οταν προκύψει κάποιο σφάλμα από τη μέθοδο που έχει δηλωθεί, μπορεί να εκτελεσθεί η μέθοδος διαχείρισης των ενεργειών που αφορούν στο συγκεκριμένο σφάλμα. Στην Java υπάρχουν μόνο 6 τύποι εξαιρέσεων (στο πακέτο java.lang) που δηλώνονται στο throws :

1.ClassNotFoundException
2.CloneNotSupportedException
3.IllegalAccessException
4.InstantiationException
5.InterruptedException
6.NoSuchMethodException

Οι 6 αυτές εξαιρέσεις αφορούν στα σφάλματα που προέκυψαν από λάθη προγραμματισμού και όχι από εσωτερικά γεγονότα όπως για παράδειγμα το κοινότυπο σφάλμα *OutOfMemoryError* που μπορεί να προκληθεί σε οποιοδήποτε σημείο του προγράμματος.

Γενικά, η βιβλιοθήκη κλάσης της Java χρησιμοποιεί εξαιρέσεις οπουδήποτε και στο API υλικό τεκμηρίωσης που συνοδεύει το πακέτο JDK της Java παρέχεται αναλυτική καταγραφή των μεθόδων που υποστηρίζει σχετικά με τις

εξαιρέσεις. Γενικά η μέθοδος που διαχειρίζεται τις εξαιρέσεις έχει την παρακάτω μορφή :

```
public void Exairesi( ) {

    miaExairesiClass exair = new miaExairesiClass();

    try {
        exair.methodosExairesis();
    } catch (prwtiExairesi e) {
    . . . // διατύπωση υπεύθυνου και σημαντικού μέρους σφάλματος
      . . .
    }
}
```

Στην παραπάνω μέθοδο ο κώδικας προσπαθεί να εκτελέσει τη μέθοδο *exair.methodosExairesis()* και εάν παρουσιαστούν εξαιρέσεις ο έλεγχος μεταφέρεται στον κώδικα που ακολουθεί την *catch*. Μπορεί να υπάρχουν περισσότερες catch, μία για κάθε διαφορετική περίπτωση σφάλματος που διαχειρίζεται η εφαρμογή. Γενικότερα, η λέξη-κλειδί *throw* αποτελεί αντίστοιχη της εντολής *break*, δηλαδή τίποτα μετά από αυτήν δεν εκτελείται. Οι εξαιρέσεις αποτελούν ένα ισχυρό μηχανισμό διαχείρισης και αντιμετώπισης διαφορετικών κομματιών σφαλμάτων. Ετσι υπάρχει η δυνατότητα χρήσης αλυσιδωτών όρων catch που να διασπούν τις περιπτώσεις λαθών σε επιμέρους μέρη τα οποία μπορούν στη συνέχεια να έχουν ευκολότερη αντιμετώπιση όπως παρουσιάζεται στο παράδειγμα που ακολουθεί :

```
try {
    methodos();
} catch (IOException io) {
. . .
} catch (RuntimeException r) {
. . .
} catch (Exception e) {
. . .
} catch (NeedsRebbot n) {
    systemClass.reboot();
}
```

Στο παραπάνω παράδειγμα υπσοτηρίζεται η αντιμετώπιση ενός συνόλου από διαφορετικά είδη σφαλμάτων. Γενικά, τα λάθη που προκύπτουν κατά την εκτέλεση (Runtime errors) παράγονται κατά την κανονική εκτέλεση του Java κώδικα και συνήθως προκύπτουν από κάποιο προγραμματιστικό λάθος. Οι εξαιρέσεις (Exceptions) δεν αποτελούν είδος Runtime error αλλά αποτελούν συνθήκες που πρέπει να τις διαχειριστούν ανθεκτικά και καλογραμμένα μέρη κώδικα. Η κλάση βιβλιοθήκης της Java διαθέτει υποστήριξη για τα κυριότερα από αυτά τα σφάλματα, έτσι ώστε να επιτρέπεται η ασφαλής και ομαλή λειτουργία του συστήματος. Ο έλεγχος φεύγει από ένα σύνολο εντολών

ελέγχου (try-catch block) μόλις το πρόγραμμα εκτελέσει μία επιστροφή ή μεταβιβαστεί μία εξαίρεση. Επιπλέον η Java επιτρέπει στον χρήστη να ορίσει ένα κομμάτι κώδικα το οποίο να εκτελεστεί υποχρεωτικά, πριν ο έλεγχος φύγει από τη μέθοδο. Αυτό το κομμάτι ονομάζεται finally block γιατί ορίζεται με τη λέξη-κλειδί finally. Η γενική σύνταξη του κώδικα που απαιτείται για την υποστήριξη των εξαιρέσεων είναι :

```
public class dokimi {

  public static void main(String[] args) {
      try { // κανονική επεξεργασία κώδικα η οποία εγκαταλείπεται μόλις
          μεταβιβαστεί η εξαίρεση ή συναντηθεί επιστροφή.
      }
      catch (Exception e) { // εκτελείται μόνο όταν μεταβιβαστεί ένα σφάλμα
      }
      finally { // εκτελείται πριν φύγει ο έλεγχος από τη μέθοδο, ότι κι αν
          γίνει, ακόμα κι αν έχουμε εξαίρεση ή επιστροφή
      }
  }
}
```

## 7.5. Είσοδος - Εξοδος Αρχείων

Η είσοδος-έξοδος αρχείων στην Java ακολουθεί το ίδιο μοντέλο που χρησιμοποιεί και η C και η C++, δηλαδή δεν υποστηρίζονται πρωτογενείς εντολές γλώσσας είσοδου-έξοδου αρχείων. Η διασωλήνωση (pipe) αποτελεί ένα μη προκαθορισμένο ρεύμα από bytes που χρησιμοποιείται για την επικοινωνία μεταξύ των προγραμμάτων για ανάγνωση και εγγραφή σε τυχαίες περιφερειακές συσκευές και αρχεία. Ενα ρεύμα ροής (stream) αποτελεί τη διαδρομή επικοινωνίας μεταξύ της πηγής και του προορισμού κάποιας πληροφορίας. Στην Java έχουν δημιουργηθεί τρία μέλη της κλάσης System (System.in, System.out, και System.err) που επιτελούν χρήσιμες λειτουργίες σχετικά με την είσοδο-έξοδο. Η System.in είναι τύπου BufferedInputStream και επιτελεί την καθιερωμένη είσοδο (συνήθως από το πληκτρολόγιο), η System.out είναι τύπου PrintStream και επιτελεί την καθιερωμένη έξοδο (συνήθως στην οθόνη) καθώς και System.err (εκφράζει την περίπτωση standard error) είναι τύπου PrintStream και επιτελεί την έξοδο των σφαλμάτων (συνήθως στην οθόνη).

Γενικά, όλες οι λειτουργίες είσοδου-έξοδου παρέχονται από κλάσεις που περιέχονται στο πακέτο java.io. Οι λειτουργίες είσοδου-έξοδου παρέχουν ασφάλεια τύπου, που σημαίνει ότι δεν μπορεί να πραγματοποιηθεί λανθασμένη λειτουργία είσοδου-έξοδου για κάποιο δεδομένο τύπο δεδομένων. Η Java διαθέτει δύο abstract κλάσεις την InputStream() και την OutputStream() του πακέτου java.io, οι οποίες διαθέτουν ένα σύνολο από υπο-κλάσεις και μεθόδους που υποστηρίζουν όλο το πλαίσιο της ροής δεδομένων στην Java.

### 7.5.1. Η abstract κλάση InputStream()

Η κλάση InputStream είναι τύπου abstract και καθορίζει τους βασικούς τρόπους με τους οποίους μία θέση προορισμού διαβάζει μία ροή από δεδομένα που έχουν σταλεί από κάποια πηγή. Δε χρειάζεται να υπάρχει συμβατότητα μεταξύ της πηγής και του προορισμού, απλά με τη χρήση της InputStream η εφαρμογή γίνεται αποδέκτης των bytes που είναι κατευθυνόμενα από την πηγή. Οι κυριότερες μέθοδοι που αφορούν στην εισροή δεδομένων είναι :

- *read( ):* πραγματοποιεί την ανάγνωση των bytes από την πηγή των δεδομένων. Κάθε read( ) μέθοδος αναμένει έως τη στιγμή που μπορεί να ξεκινήσει η ροή των δεδομένων. Μία γενική μορφή της read( ) μπορεί να είναι :

```
InputStream eis = pareDedomena();
byte[ ]  buffer = new byte[1024];

if (eis.read(buffer) != buffer.length)
      System.out.println("Παραλαβή μικρότερου όγκου δεδομένων");
```

η οποία προσπαθεί να παραλάβει ένα συγκεκριμένο όγκο δεδομένων και όταν δεν τον παραλάβει εκτυπώνει σχετικό μήνυμα. Γενικά τα σφάλματα που προέρχονται από διαδικασίες εισόδου-εξόδου προκαλούν εξαίρεση τύπου IOException. Μία άλλη μορφή της read μπορεί να είναι η :

*eis.read(buffer, 200, 400);*

όπου γίνεται παραλαβή των bytes 200 έως και 599 και έχει παρόμοια συμπεριφορά. Γενικά η μέθοδος read() επιστρέφει έναν ακέραιο τύπου int

- *skip( ):* όταν χρειάζεται να παραλειφθούν κάποια bytes από κάποια ροή δεδομένων ή να γίνει ανάγνωση από κάποιο ενδιάμεσο σημείο της ροής.

- *available( ):* όταν χρειάζεται η πληροφορία των διαθέσιμων bytes που βρίσκονται στη ροή δεδομένων κάποια χρονική στιγμή. Η μέθοδος αυτή επιστρέφει τον αριθμό των bytes που μπορούν να παραληφθούν με κλήση της read() χωρίς διακοπή.

- *mark( ) και reset( ):* κάποια είδη ροής δεδομένων επιτρέπουν την έννοια της σημείωσης κάποιων θέσεων της ροής δεδομένων στα οποία υπάρχει κάποιος δείκτης έτσι ώστε να υπάρχει δυνατότητα επιστροφής στο συγκεκριμένο αυτό σημείο μέσω της μεθόδου reset( ). Επίσης υπάρχει η μέθοδος markSupported() η οποία δίνει την πληροφορία σχετικά με το εάν μία ροή δεδομένων δέχεται τις μεθόδους mark( ) και reset( ) ή όχι.

- *close( ):* μετά το τέλος της εισόδου των δεδομένων, κλείνει η ροή δεδομένων έτσι ώστε να ελευθερώνονται οι πόροι του συστήματος. Κάποια είδη ροής δεδομένων επιτρέπουν την έννοια της σημείωσης κάποιων θέσεων σε θέσεις της ροής στα οποία μπορεί να φυλαχτεί η πληροφορία των διαθέσιμων bytes που βρίσκονται στη ροή κάποια χρονική στιγμή.

Γενικά υπάρχουν διάφορα είδη ρευμάτων εισροής δεδομένων, όπως ByteArrayInputStream, FileInputStream, FilterInputStream, BufferedInputStream, DataInputStream, LineNumberInputStream, PushBackInputStream, PipedInputStream και StringBufferInputStream. Αναλυτικά περιγράφονται στο υποστηρικτικό πακέτο API της Java. Στη συνέχεια παρατίθενται κάποια γενικά στοιχεία για τη πλέον συνήθη από αυτές την FileInputStream.

FileInputStream : Αποτελεί τη συνηθέστερη περίπτωση χρήσης ροής δεδομένων κατά την οποία συνδέεται κάποια ροή δεδομένων με κάποιο αρχείο. Για παράδειγμα σε ένα σύστημα UNIX θα είχαμε την εντολή :

```
InputStream eisodos = new FileInputStream("/users/csd/arheio");
```

και χρειάζεται ιδιαίτερη προσοχή κατά τη χρήση της γιατί η ανάγνωση στοιχείων από το αρχείο μπορεί να προκαλέσει παραβιάσεις στην ασφάλεια του συστήματος αρχείων. Είναι πιο χρήσιμο να αναπτύσσεται η εφαρμογή ανεξάρτητα από κάποια αρχεία αλλά να βασίζεται σε κάποιους εξυπηρέτες που διαφυλάσσουν διαμοιραζόμενη πληροφορία.

### 7.5.2. Η abstract κλάση OutputStream()

Η κλάση OutputStream είναι η αντίστοιχη της InputStream για την περίπτωση της εξόδου ροής κάποιων bytes από μία πηγή προς μία κατεύθυνση. Επομένως η OutputStream επιτελεί τις αντίθετες λειτουργίες της InputStream. Η κυριότερη μέθοδος που αφορά στην εκροή δεδομένων είναι :

- *write( ):* πραγματοποιεί την εγγραφή των bytes προς τη θέση προορισμού και αναμένει έως ότου η ροή των δεδομένων έχει περατωθεί.

Αντίστοιχα, υπάρχουν διάφορα είδη ρευμάτων εκροής δεδομένων, όπως ByteArrayOutputStream, FileOutputStream, FilterOutputStream, BufferedOutputStream, DataOutputStream, PipedOutputStream και PrintStream. Σε όλα τα προγράμματα που παρατέθηκαν έως τώρα ήδη χρησιμοποιήθηκαν οι δύο μέθοδοι της που καλούνται όταν υπάρχει στον κώδικα η εντολή System.out.print ή η εντολή System.out.println. Αναλυτικά περιγράφονται στο υποστηρικτικό πακέτο API της Java.

## 7.6. Η ιδεατή μηχανή της Java

Το σύστημα της Java βασίζεται σε μία ιδεατή μηχανή(virtual machine).Η βασική ιδέα της ιδεατής μηχανής είναι ότι καθορίζεται ένας ιδεατός υπολογιστής και η γλώσσα προγραμματισμού περιλαμβάνει κώδικα που έχει πρόσβαση στις λογικές συναρτήσεις και συσκευές αυτής της ιδεατής μηχανής. Στη συνέχεια, κάθε πλατφόρμα που υλοποιεί τις προδιαγραφές της ιδεατής μηχανής είναι υπεύθυνη για τη μετάφραση των συναρτήσεων και των συσκευών της ιδεατής μηχανής σε εγγενή κώδικα υποστήριξης. Για παράδειγμα, η ιδεατή μηχανή της Java για το λειτουργικό σύστημα Sun Solaris έχει την ευθύνη της αντιστοίχισης των πρωταρχικών τύπων γραφικών της κλάσης Graphics, στις τοπικές βιβλιοθήκες συστήματος.

Η ιδέα της ιδεατής μηχανής είναι παλαιότερη με ενδεικτικότερη περίπτωση την ανάπτυξη του συστήματος USCD της Pascal. Το πλεονέκτημα από τη χρησιμοποίηση μίας ιδεατής μηχανής είναι η δυνατότητα προσαρμογής της Java σε ένα σημαντικό αριθμό από διαφορετικές πλατφόρμες. Η μηχανή της Java υλοποιεί ένα σύνολο από λογικές ή ιδεατές συσκευές, ώστε να μην απαιτείται η υποστήριξη βιβλιοθηκών συγκεκριμένων τύπων συσκευών (π.χ. κάρτες δικτύου). Η χρήση ιδεατής μηχανής παρέχει ένα κοινό περιβάλλον ανάπτυξης που είναι συνεπές και σταθερό μεταξύ διαφόρων συστημάτων λογισμικού και υλικού. Με αυτόν τον τρόπο δίνεται η δυνατότητα τόσο στον χρήστη όσο και στον συντάκτη του κώδικα να έχουν κοινό σημείο αναφοράς ενός ενιαίου συστήματος αποφεύγοντας τις περιπλοκές και διαφορές των δια-πλατφορμικών συστημάτων.

Η Java περιέχει πλήρεις προδιαγραφές για την ιδεατή μηχανή που είναι ιδιαίτερα ακριβείς και προδιαγράφουν τις συναρτήσεις υποστήριξης της υλοποίηση της Java. Ακόμα, καθορίζεται με πλήρη λεπτομέρεια ο ακριβής τρόπος με τον οποίο θα υλοποιηθούν αυτές οι συναρτήσεις. Η ελευθερία αυτή στην ανάπτυξη του κώδικα επιτρέπει στα Java συστήματα να εκμεταλλευτούν τη διαφορετικότητα των Java πλατφορμών.

Υπάρχουν βέβαια και αρνητικά στοιχεία από την ύπαρξη της ιδεατής μηχανής. Το κυριότερο κόστος αφορά στο ότι η δια-πλατφορμική υποστήριξη επιβάλλει την ανάγκη υποστήριξης τέτοιων προδιαγραφών που να ικανοποιούν και το σύστημα χαμηλότερων επιδόσεων. Ετσι η λειτουργικότητα των συστημάτων απαιτήσεων προσαρμόζεται στα συστήματα χαμηλότερης ποιότητας και επίδοσης. Η Java επιχειρεί να επιλύσει αυτό το πρόβλημα με την ανάπτυξη εγγενών μεθόδων σε κάθε σύστημα. Αυτές οι εγγενείς μέθοδοι αναπτύσσονται σε άλλη γλώσσα όπως η C ή η C++, δηλώνονται στην Java και φορτώνονται στο περιβάλλον της Java με χρήση δυναμικών βιβλιοθηκών. Αυτός ο "εξωτερικός" κώδικας εκμεταλλεύεται τις δυνατότητες κάθε συστήματος και συμπληρώνει τις Java προδιαγραφές. Ακόμα οι εγγενείς μέθοδοι προσφέρουν δυνατότητα πρόσβασης σε βιβλιοθήκες του συστήματος που δεν ήταν διαθέσιμες στο σύστημα της Java και συντελούν στη βελτίωση της επίδοσης, ειδικά για προγράμματα απαιτήσεων και πολλαπλών επαναληπτικών βρόγχων.

## 7.7. Διαχείριση μνήμης στην Java

Στις συμβατικές γλώσσες προγραμματισμού (π.χ. C, C++) οι συντάκτες του κώδικα είναι υπεύθυνοι για τη διαχείριση της μνήμης, δηλαδή για την τοποθέτηση, την παρακολούθηση και την ελευθέρωση κομματιών (blocks) μνήμης. Ενα από τα ισχυρά χαρακτηριστικά της Java είναι η μη-απαίτηση κλήσεων όπως η *malloc* ή η *free* που χρειάζονται στις άλλες γλώσσες. Από τη στιγμή που θα δημιουργηθεί ένα αντικείμενο το σύστημα της Java είναι υπεύθυνο για την ευθύνη παρακολούθησης και ανάθεσης της απαραίτητης μνήμης, καθώς και για την απελευθέρωση αυτής της μνήμης σε περίπτωση που το αντικείμενο είναι πλέον περιττό. Υπάρχει μία ξεχωριστή αυτόματη διαδικασία συλλογής "απορριμμάτων"(garbage collector) που συλλέγει τα αντικείμενα που είναι πλέον περιττά και έχει την ευθύνη της αποκατάστασης του χώρου μνήμης που αυτά καταλαμβάνουν. Η διαδικασία συλλογής "απορριμμάτων" ξεκινάει όταν έχουν εξαντληθεί οι πόροι μνήμης του Java συστήματος. Επιπλέον δίνεται η δυνατότητα ανάπτυξης κώδικα που να περιλαμβάνει μεθόδους "καταστροφής" πριν από την αυτόματη κλήση του garbage collector. Η Java υποστηρίζει την κλάση Vector που επιτρέπει την αυτόματη ρύθμιση μεγέθους των αντικειμένων γενικού τύπου. Οι δυνατότητες αυτές δεν είναι ίσως τόσο σημαντικές για την περίπτωση μεγαλύτερων συστημάτων(workstations) αλλά αποτελεί σημαντική δυνατότητα εκμετάλλευσης της μνήμης για τα συστήματα μικρότερων απαιτήσεων όπως τα PCs.

## 7.8. Θέματα ασφάλειας στην Java

Η κυριότερη και πολλά υποσχόμενη δυνατότητα των Java συστημάτων αφορά στην εισαγωγή, μεταφορά κώδικα από απομακρυσμένους υπολογιστές-εξυπηρέτες και η εκτέλεση τους μέσω των προγραμμάτων πλοήγησης. Επομένως, ένα πολύ σημαντικό ζήτημα είναι η ασφάλεια και η ακεραιότητα των δεδομένων που μεταφέρονται. Η ασφάλεια του συστήματος πραγματοποιείται σε διαφορετικά επίπεδα και δίνει πραγματική ισχύ και εγκυρότητα στο σύστημα της Java. Τα χαρακτηριστικά της ασφάλειας αφορούν σε τέσσερα επίπεδα :

- Μεταφραστής και η Java γλώσσα : Κάποια από τα χαρακτηριστικά της γλώσσας προσθέτουν επιπλέον ασφάλεια στο σύστημα. Ο μεταφραστής της Java ελέγχει τον κώδικα που χρειάζεται να μεταφραστεί και αρνείται να μεταφράσει κώδικα που δεν πληρεί τους αυστηρούς κανόνες της Java.

- Επιβεβαίωση των byte-code Τα προγράμματα πλοήγησης εφαρμόζουν ένα σύνολο από ελέγχους στα μέρη κώδικα (byte-codes) πριν ελεγχθούν. Το σύνολο των ελέγχων περιλαμβάνει επιβεβαίωση εσφαλμένων δεικτών έτσι ώστε να αποφεύγεται η "παράνομη" πρόσβαση στο τοπικό σύστημα.

- Η κλάση loader Μετά τη διάβαση του κώδικα από την επιβεβαίωση των byte-codes ο έλεγχος μεταβιβάζεται στην κλάση loader. Η κλάση αυτή ελέγχει εάν τον πυρήνα του Java κώδικα και διασφαλίζει τη μοναδική διαδρομή πρόσβασης σε μία κλάση.

- Προστασία του τοπικού συστήματος Το τελικό στάδιο του ελέγχου ασφάλειας περιλαμβάνει ένα σύνολο από ελέγχους πρόσβασης. Το πρόγραμμα πλοήγησης της Java εξετάζει τις κλήσεις από κλάσεις του τοπικού συστήματος και ελέγχει εάν επιχειρείται μέσω της Java κλήση σε αρχεία εκτός του χώρου πρόσβασης της Java. Ετσι διασφαλίζεται η ασφάλεια των τοπικών αρχείων και αποφεύγεται εσφαλμένη πρόσβαση σε αυτά.

## *ΒΙΒΛΙΟΓΡΑΦΙΑ*

1. *Active Java*, A. Freeman and D.Ince, Addison Wesley Publishing Company Inc, 1997.

2. *Δουλέψτε με την Java*, S. R. Davis, Microsoft Press, 1996.

3. *Teach yourself Java in 21 days*, L. Lemay and C. Perkins, Sams net, 1996.

4. *Late Night Advanced Java*, J. Wehling, V. Bharat et al, Ziff-Davis Press, 1996.

5. *JAVA How to Program*, Deitel & Deitel, Prentice Hall International Inc, 1997.

6. *Using Java*, 2nd edition, Joe Weber et al, QUE Corporation, 1996.

7. *JavaScript*, The Definite Guide, David Flanagan, O'Reilly & Associates, Inc. 1997.

8. *Exploring Java*, P. Niemeyer & J. Peck, O'Reilly & Associates, Inc. 1997

9. *Object-Oriented Programming in Java*, S.Gilbert, B.Mc.Carty, Waite Group Press, 1997.

10. *Java 1.1. Certification Study Guide*, S.Roberts, P.Heller, *Sybex Inc. 1997.*

11. *Visual J++ Unleashed*, B. Morgan et al, *Sams net, 1997.*

## *ΗΛΕΚΤΡΟΝΙΚΗ ΒΙΒΛΙΟΓΡΑΦΙΑ*

Ενας μεγάλος αριθμός κόμβων του Internet περιέχει πληροφορίες σχετικά με την Java, τα προϊόντα και τις εφαρμογές της. Ακολουθεί μία ενδεικτική κατάσταση των κυριότερων Web κόμβων για την αρχική ενασχόληση με την Java.

| | |
|---|---|
| http://java.sun.com/ | Κεντρικό σημείο αναφοράς για την Java. Αποτελεί τη συλλογική και ενοποιημένη πληροφορία σχετικά με την Java από την εταιρία Sun. |
| http://www.javalobby.org/ | Ομάδα εργασίας ειδικών για ανταλλαγή ενδιαφερόντων και απόψεων σχετικά με την Java. |
| http://www.javaworld.com/ | Ηλεκτρονική έκδοση περιοδικού για την Java. |

| | |
|---|---|
| http://www.jars.com/ | JARS : Java Applet Rating Service Υπηρεσία κατάταξης και ελέγχου των Java μικρο-εφαρμογών. |
| http://www.javasoft.com/ | Κεντρικός κόμβος υποστήριξης της Java, ταυτόσημος με τον κόμβο http://java.sun.com/ της Sun. |
| http://www.javasoft.com/products/ | Προϊόντα και υπηρεσίες της Java |
| http://www.javasoft.com/products/jdk/ | Κόμβος παροχής του JDK πακέτου της Java, καθώς και της σχετικής τεκμηρίωσης του. |
| http://www.javasoft.com/beans/ | Αρχιτεκτονικό δια-πλατφορμικό περιβάλλον ανάπτυξης της Java. |
| http://www.sys-con.com/java/ | Ηλεκτρονική έκδοση εφημερίδας για την Java. |
| http://www.teamjava.com/ | Φορέας προώθησης και παροχής συμβουλευτικών υπηρεσιών για επαγγελματική ενασχόληση με την Java. |
| http://www.bulletproof.com/ | Το Java σύστημα ανάπτυξης JDesigner-Pro της εταιρίας BulletProof για ανάπτυξη Intranet εφαρμογών. |
| http://www.cupojoe.com/ | Ηλεκρονικό "κατάστημα" με Java προϊόντα. |
| http://www.javameansbusiness.com/ | Συνέδριο που διοργανώνει η Sun και άλλοι συνεργαζόμενοι φορείς σχετικά με τις επιχειρηματικές εφαρμογές της Java. |

Οι παραπάνω κόμβοι περιέχουν συνδέσεις προς άλλους σχετικούς κόμβους και καλύπτουν ένα ευρύ φάσμα θεματικών αντικειμένων σχετικών με την Java.

# ΠΕΡΙΕΧΟΜΕΝΑ ΠΑΡΑΡΤΗΜΑΤΟΣ

**Α) ΠΑΡΑΔΕΙΓΜΑΤΑ ΑΠΛΩΝ JAVA ΠΡΟΓΡΑΜΜΑΤΩΝ**

```
// Ένα πρόγραμμα ανάλυσης της βαθμολογίας
// των αποτελεσμάτων εξετάσεων

import java.io.*;

public class Analysis {
   public static void main( String args[] ) throws IOException
   {
      // initializing variables in declarations
      int passes = 0, failures = 0, student = 1, result;

      // process 10 students; counter-controlled loop
      while ( student <= 10 ) {
         System.out.print( "Enter result (1=pass,2=fail): " );
         System.out.flush();
         result = System.in.read();

         if ( result == '1' )      // if/else nested in while
            passes = passes + 1;
         else
            failures = failures + 1;

         student = student + 1;
         System.in.skip( 1 );
      }

      System.out.println( "Passed " + passes );
      System.out.println( "Failed " + failures );

      if ( passes > 8 )
         System.out.println( "Raise tuition " );
   }
}
```

---

```
// Αύξηση « πριν « και « μετά»

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

public class Increment extends Applet {
   public void paint( Graphics g )
   {
      int c;

      c = 5;
     g.drawString( Integer.toString( c ), 25, 25 );
      g.drawString( Integer.toString( c++ ), 25, 40 ); //
postincrement
      g.drawString( Integer.toString( c ), 25, 55 );

      c = 5;
      g.drawString( Integer.toString( c ), 25, 85 );
      g.drawString( Integer.toString( ++c ), 25, 100 ); //
preincrement
      g.drawString( Integer.toString( c ), 25, 115 );
   }
}
```

---

```
//  Πρόγραμμα ανεύρεση Μέσου όρου με βρόγχους επανάληψης

import java.io.*;
public class Average {
   public static void main( String args[] ) throws IOException
   {
      int counter, grade, total, average;

      // initialization phase
      total = 0;
      counter = 1;

      // processing phase
      while ( counter <= 10 ) {
         System.out.print( "Enter letter grade: " );
         System.out.flush();
         grade = System.in.read();

         if ( grade == 'A' )
            total = total + 4;
         else if ( grade == 'B' )
            total = total + 3;
         else if ( grade == 'C' )
            total = total + 2;
         else if ( grade == 'D' )
            total = total + 1;
         else if ( grade == 'F' )
            total = total + 0;

         System.in.skip( 1 );    // skip the newline character
         counter = counter + 1;
      }

      // termination phase
      average = total / 10;      // integer division
      System.out.println( "Class average is " + average );
   }
}
```

**Β) ΠΑΡΑΔΕΙΓΜΑΤΑ ΔΟΜΩΝ ΕΛΕΓΧΟΥ ΠΡΟΓΡΑΜΜΑΤΩΝ**

**// Υπολογισμός τοκιζόμενου κεφαλαίου σε μία τράπεζα**

```
import java.awt.Graphics;
import java.applet.Applet;
public class Interest extends Applet {
   public void paint( Graphics g )
   {
      double amount, principal = 1000.0, rate = .05;
      int yPos = 40;

      g.drawString( "Year", 25, 25 );
      g.drawString( "Amount on deposit", 100, 25 );

      for ( int year = 1; year <= 10; year++ ) {
         amount = principal * Math.pow( 1.0 + rate, year );
         g.drawString( Integer.toString( year ), 25, yPos );
         g.drawString( Double.toString( amount ), 100, yPos );
         yPos += 15;
      }
   }
}
```

**// Χρήση της continue σε ένα πρόγραμμα**

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

public class ContinueTest extends Applet {
   public void paint( Graphics g )
   {
      int xPos = 25;
      for ( int count = 1; count <= 10; count++ ) {
         if ( count == 5 )
            continue;  // skip remaining code in loop
                       // only if count == 5

         g.drawString( Integer.toString( count ), xPos, 25 );
         xPos += 10;
      }

      g.drawString( "Used continue to skip printing 5",
                    25, 40 );
   }
}
```

```
// Χρήση λογικών τελεστών

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

public class LogicalOperators extends Applet {
   public void paint( Graphics g )
   {
      g.drawString( "Logical AND (&&)", 10, 25 );
      g.drawString( "F && F: " + ( false && false ), 10, 40 );
      g.drawString( "F && T: " + ( false && true ), 10, 55 );
      g.drawString( "T && F: " + ( true && false ), 10, 70 );
      g.drawString( "T && T: " + ( true && true ), 10, 85 );
      g.drawString( "Logical OR (||)", 215, 25 );
      g.drawString( "F || F: " + ( false || false ), 215, 40 );
      g.drawString( "F || T: " + ( false || true ), 215, 55 );
      g.drawString( "T || F: " + ( true || false ), 215, 70 );
      g.drawString( "T || T: " + ( true || true ), 215, 85 );

      g.drawString( "Boolean logical AND (&)", 10, 115 );
      g.drawString( "F & F: " + ( false & false ), 10, 130 );
      g.drawString( "F & T: " + ( false & true ), 10, 145 );
      g.drawString( "T & F: " + ( true & false ), 10, 160 );
      g.drawString( "T & T: " + ( true & true ), 10, 175 );

      g.drawString( "Boolean logical inclusive OR (|)",
                    215, 115 );
      g.drawString( "F | F: " + ( false | false ), 215, 130 );
      g.drawString( "F | T: " + ( false | true ), 215, 145 );
      g.drawString( "T | F: " + ( true | false ), 215, 160 );
      g.drawString( "T | T: " + ( true | true ), 215, 175 );

      g.drawString( "Boolean logical exclusive OR (^)",
                    10, 205 );
      g.drawString( "F ^ F: " + ( false ^ false ), 10, 220 );
      g.drawString( "F ^ T: " + ( false ^ true ), 10, 235 );
      g.drawString( "T ^ F: " + ( true ^ false ), 10, 250 );
      g.drawString( "T ^ T: " + ( true ^ true ), 10, 265 );

      g.drawString( "Logical NOT (!)",
                    215, 205 );
      g.drawString( "!F: " + ( !false ), 215, 220 );
      g.drawString( "!T: " + ( !true ), 215, 235 );
   }
}
```

**Γ) ΠΡΟΓΡΑΜΜΑΤΑ ΓΙΑ ΤΗ ΧΡΗΣΗ ΤΩΝ ΜΕΘΟΔΩΝ**

```
//  Παραγωγή τυχαίων αριθμών
import java.awt.Graphics;
import java.applet.Applet;

public class RandomInt extends Applet {
   public void paint( Graphics g )
   {
      int xPosition = 25;
      int yPosition = 25;
      int value;

      for ( int i = 1; i <= 20; i++ ) {
         value = 1 + (int) ( Math.random() * 6 );
         g.drawString( Integer.toString( value ),
                       xPosition, yPosition );

         if ( i % 5 != 0 )
            xPosition += 40;
         else {
            xPosition = 25;
            yPosition += 15;
         }
      }
   }
}
```

---

```
//  Ρίψη ζαριού 6000 φορές

import java.awt.Graphics;
import java.applet.Applet;
public class RollDie extends Applet {
   int frequency1 = 0, frequency2 = 0,
       frequency3 = 0, frequency4 = 0,
       frequency5 = 0, frequency6 = 0;

   public void start()
   {
      for ( int roll = 1; roll <= 6000; roll++ ) {
         int face = 1 + (int) ( Math.random() * 6 );

         switch ( face ) {
            case 1:
               ++frequency1;
               break;
            case 2:
               ++frequency2;
               break;
            case 3:
               ++frequency3;
               break;
            case 4:
               ++frequency4;
               break;
            case 5:
               ++frequency5;
               break;
            case 6:
               ++frequency6;
               break;
         }
      }
   }

   // display results
```

```
   public void paint( Graphics g )
   {
      g.drawString( "Face", 25, 25 );
      g.drawString( "Frequency", 100, 25 );
      g.drawString( "1", 25, 40 );
      g.drawString( Integer.toString( frequency1 ), 100, 40 );
      g.drawString( "2", 25, 55 );
      g.drawString( Integer.toString( frequency2 ), 100, 55 );
      g.drawString( "3", 25, 70 );
      g.drawString( Integer.toString( frequency3 ), 100, 70 );
      g.drawString( "4", 25, 85 );
      g.drawString( Integer.toString( frequency4 ), 100, 85 );
      g.drawString( "5", 25, 100 );
      g.drawString( Integer.toString( frequency5 ),
                    100, 100 );
      g.drawString( "6", 25, 115 );
      g.drawString( Integer.toString( frequency6 ),
                    100, 115 );
   }
}
```

---

```
// Περιοχή ενέργειας μεταβλητών (scoping)
import java.awt.Graphics;
import java.applet.Applet;

public class Scoping extends Applet {
   int x = 1;      // instance variable
   public void paint( Graphics g )
   {
      g.drawString( "See command line for output", 25, 25 );
      int x = 5;   // local variable to paint
      System.out.println( "local x in paint is " + x );
      a();         // a has automatic local x
      b();         // b uses instance variable x
      a();         // a reinitializes automatic local x
      b();         // instance variable x retains its value

      System.out.println( "\nlocal x in paint is " + x );
   }
   void a()
   {
      int x = 25;  // initialized each time a is called
      System.out.println( "\nlocal x in a is " + x +
                          " after entering a" );
      ++x;
      System.out.println( "local x in a is " + x +
                          " before exiting a" );
   }

   void b()
   {
      System.out.println( "\ninstance variable x is " + x +
                          " on entering b" );
      x *= 10;
      System.out.println( "instance variable x is " + x +
                          " on exiting b" );
   }
}
```

**// Υπολογισμός παραγοντικού**

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

public class FactorialTest extends Applet {
   public void paint( Graphics g )
   {
      int yPosition = 25;

      for ( long i = 0; i <= 10; i++ )  {
         g.drawString( i + "! = " + factorial( i ),
                       25, yPosition );
         yPosition += 15;
      }
   }
   public long factorial( long number )
   {
      if ( number <= 1 )  // base case
         return 1;
      else
         return number * factorial( number - 1 );
   }
}
```

---

**// Υπολογισμός σειράς Fibonacci**

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

public class FibonacciTest extends Applet {
   Label numLabel, resultLabel;
   TextField num, result;

   public void init()
   {
      numLabel = new Label( "Enter an integer and press return" );
      num = new TextField( 10 );
      resultLabel = new Label( "Fibonacci value is" );
      result = new TextField( 15 );
      result.setEditable( false );
      add( numLabel );
      add( num );
      add( resultLabel );
      add( result );
   }
   public boolean action( Event e, Object o )
   {
      long number, fibonacciVal;
      number = Long.parseLong( num.getText() );
      showStatus( "Calculating ..." );
      fibonacciVal = fibonacci( number );
      showStatus( "Done." );
      result.setText( Long.toString( fibonacciVal ) );
      return true;
   }
   // Recursive definition of method fibonacci
   long fibonacci( long n )
   {
      if ( n == 0 || n == 1 )  // base case
         return n;
      else
         return fibonacci( n - 1 ) + fibonacci( n - 2 );
   }
}
```

**// Υπερ-φόρτωση (**overload**) μεθόδων (Παράδειγμα 1)**

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

public class MethodOverload extends Applet {
   public void paint( Graphics g )
   {
      g.drawString( "The square of integer 7 is " + square( 7 ),
                    25, 25 );
      g.drawString( "The square of double 7.5 is " +
                    square( 7.5 ), 25, 40 );
   }

   int square( int x )
   {
      return x * x;
   }

   double square( double y )
   {
      return y * y;
   }
}
```

**// Υπερ-φόρτωση (**overload**) μεθόδων (Παράδειγμα 2 ** error **)**

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

public class MethodOverload extends Applet {
   int square( double x )
   {
      return x * x;
   }

   double square( double y )
   {
      return y * y;
   }
}
```

**Δ) ΠΡΟΓΡΑΜΜΑΤΑ ΓΙΑ ΤΗ ΧΡΗΣΗ ΠΙΝΑΚΩΝ**

```
// αρχικοποίηση πίνακα
import java.awt.Graphics;
import java.applet.Applet;
public class InitArray extends Applet {
   int n[];        // declare an array of integers
   public void init()
   {
      n = new int[ 10 ];  // dynamically allocate array
   }
   public void paint( Graphics g )
   {
      int yPosition = 25;  // starting y position on applet

      g.drawString( "Element", 25, yPosition );
      g.drawString( "Value", 100, yPosition );

      for ( int i = 0; i < n.length; i++ ) {
         yPosition += 15;
         g.drawString( String.valueOf( i ), 25, yPosition );
         g.drawString( String.valueOf( n[ i ] ),
                       100, yPosition );
      }
   }
}
```

---

```
// Πρόγραμμα τύπου «γκάλοπ
import java.awt.Graphics;
import java.applet.Applet;

public class StudentPoll extends Applet {
   int responses[] = { 1, 2, 6, 4, 8, 5, 9, 7, 8, 10,
                       1, 6, 3, 8, 6, 10, 3, 8, 2, 7,
                       6, 5, 7, 6, 8, 6, 7, 5, 6, 6,
                       5, 6, 7, 5, 6, 4, 8, 6, 8, 10 };
   int frequency[];

   public void init()
   {
      frequency = new int[ 11 ];
      for ( int answer = 0; answer < responses.length; answer++ )
         ++frequency[ responses[ answer ] ];
   }

   // paint the applet
   public void paint( Graphics g )
   {
      int yPosition = 25;   // starting y position on applet

      g.drawString( "Rating", 25, yPosition );
      g.drawString( "Frequency", 100, yPosition );

      for ( int rating = 1;
            rating < frequency.length; rating++ ) {
         yPosition += 15;
         g.drawString( String.valueOf( rating ),
                       25, yPosition );
         g.drawString( String.valueOf( frequency[ rating ] ),
                       100, yPosition );
      }
   }
}
```

```
// Παράδειγμα δι-διάστατου πίνακα
```

```
import java.awt.Graphics;
import java.applet.Applet;
public class DoubleArray extends Applet {
   int grades[][] = { { 77, 68, 86, 73 }, { 96, 87, 89, 81 },{ 70,
90, 86, 81 } };
   int students, exams;
   int xPosition, yPosition;
   public void init()
   {
      students = grades.length;
      exams = grades[ 0 ].length;
   }
   public void paint( Graphics g )
   {
      xPosition = 25;
      yPosition = 25;
      g.drawString( "The array is:", xPosition, yPosition );
      yPosition += 15;
      printArray( g );
      xPosition = 25;
      yPosition += 30;
      g.drawString( "Lowest grade:", xPosition, yPosition );
      int min = minimum();
      g.drawString( String.valueOf( min ),xPosition + 85, yPosition
);
      yPosition += 15;
      g.drawString( "Highest grade:", xPosition, yPosition );
      int max = maximum();
      g.drawString( String.valueOf( max ),xPosition + 85, yPosition
);
      yPosition += 15;
      for ( int i = 0; i < students; i++ ) {
         g.drawString( "Average for student " + i + " is ", 25,
yPosition );
         double ave = average( grades[ i ] );
         g.drawString( String.valueOf( ave ), 165, yPosition );
         yPosition += 15;
      }
   }
   public int minimum()
   {  int lowGrade = 100;
      for ( int i = 0; i < students; i++ )
         for ( int j = 0; j < exams; j++ )
            if ( grades[ i ][ j ] < lowGrade )
              lowGrade = grades[ i ][ j ];
      return lowGrade;
   }
   public int maximum()
   {  int highGrade = 0;
      for ( int i = 0; i < students; i++ )
         for ( int j = 0; j < exams; j++ )
            if ( grades[ i ][ j ] > highGrade )
               highGrade = grades[ i ][ j ];
      return highGrade;
   }

   // determine the average grade for a particular
   // student (or set of grades)
   public double average( int setOfGrades[] )
   {
      int total = 0;

      for ( int i = 0; i < setOfGrades.length; i++ )
         total += setOfGrades[ i ];
      return (double) total / setOfGrades.length;
   }
```

```
   // print the array
   public void printArray( Graphics g )
   {
      xPosition = 80;

      for ( int i = 0; i < exams; i++ ) {
         g.drawString( "[" + i + "]", xPosition, yPosition );
         xPosition += 30;
      }

      for ( int i = 0; i < students; i++ ) {
         xPosition = 25;
         yPosition += 15;
         g.drawString( "grades[" + i + "]",
                       xPosition, yPosition );
         xPosition = 80;

         for ( int j = 0; j < exams; j++ )  {
            g.drawString( String.valueOf( grades[ i ][ j ] ),
                          xPosition, yPosition );
            xPosition += 30;
         }
      }
   }
}
```

**Ε) ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΑΦΗΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ**

```
//  ΠΑΡΆΔΕΙΓΜΑ πωλητή και μεθόδων του

import java.awt.*;
import java.applet.Applet;
public class SalesPersonTest extends Applet {
   private SalesPerson s;
   private double salesFigure;
   private int month;
   private boolean moreInput = true;
   private String months[] = { "January", "February",
                       "March", "April", "May", "June",
                       "July", "August", "September",
                       "October", "November", "December" };

   // GUI components
   private Label enterLabel;
   private TextField currentMonth, enter, total;

   public void init()
   {
      s = new SalesPerson();
      month = 1;

      enterLabel = new Label( "Enter sales for" );
      currentMonth = new TextField( "January", 10 );
      currentMonth.setEditable( false );
      enter = new TextField( 10 );
      total = new TextField( 30 );
      total.setEditable( false );

      add( enterLabel );
      add( currentMonth );
      add( enter );
      add( total );
   }

   public boolean action( Event e, Object o )
   {
      if ( e.target == enter ) {
         if ( moreInput ) {
            Double val = Double.valueOf( e.arg.toString() );
            s.setSales( month, val.doubleValue() );

            if ( month == 12 ) {
               total.setText( s.toString() );

               // hide GUI components no longer needed
               enterLabel.hide();
               currentMonth.hide();
               enter.hide();
               moreInput = false;
            }
            else {
               month++;
               currentMonth.setText( months[ month - 1 ] );
               enter.setText( "" );  // clear the text field
            }
         }
      }
      return true;
   }
}

// πρόσβαση δεδομένων άλλης κλάσης στο ίδιο πακέτο
```

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

public class FriendlyDataTest extends Applet {
   private FriendlyData d;

   public void init()
   {
      d = new FriendlyData();
   }

   public void paint( Graphics g )
   {
      g.drawString( "After instantiation: ", 25, 25 );
      g.drawString( d.toString(), 40, 40 );

      d.x = 77;
      d.s = new String( "Good bye" );
      g.drawString( "After changing values: ", 25, 55 );
      g.drawString( d.toString(), 40, 70 );
   }
}

class FriendlyData {
   int x;     // friendly instance variable
   String s;  // friendly instance variable

   // constructor
   public FriendlyData()
   {
      x = 0;
      s = new String( "Hello" );
   }
   public String toString()
   {
      return "x: " + x + "    s: " + s;
   }
}
```

**// χρήση του ' This ' στο πρόγραμμα**

```
import java.awt.Graphics;
import java.applet.Applet;
public class ThisTest extends Applet {
   private int x = 12;
   public void paint( Graphics g )
   {
      g.drawString( this.toString(), 25, 25 );
   }

   public String toString()
   {
      return "x: " + x + "   this.x = " + this.x;
   }
}
```

```
// ΠΑΡΆΔΕΙΓΜΑ δομών κατασκευής (constructors και finalizers) υπερ-
και υπο-κλάσεων
import java.awt.Graphics;
import java.applet.Applet;

public class Test extends Applet {
   private Circle circle1, circle2;
   public void init()
   {
      circle1 = new Circle( 4.5, 7.2, 2.9 );
      circle2 = new Circle( 10, 5, 5 );
   }
   public void start()
   {
      circle2 = null; // Circle can now be garbage collected
      circle1 = null; // Circle can now be garbage collected

      System.gc();    // call the garbage collector
   }
   public void paint( Graphics g )
   {
      g.drawString(
         "See command line or Java Console for output",
         25, 25 );
   }
}
```

---

```
// ΠΑΡΆΔΕΙΓΜΑ ιεράρχισης κλάσης υπαλλήλων
import java.awt.Graphics;
″
import java.applet.Applet;
″

″
public class Test extends Applet {
″
   private Employee ref;  // base-class reference
   private Boss b;
   private CommissionWorker c;
   private PieceWorker p;
   private HourlyWorker h;
   public void init()
   {
      b = new Boss( "John", "Smith", 800.00 );
      c = new CommissionWorker( "Sue", "Jones", 400.0, 3.0, 150);
      p = new PieceWorker( "Bob", "Lewis", 2.5, 200 );
      h = new HourlyWorker( "Karen", "Price", 13.75, 40 );
   }
   public void paint( Graphics g )
   {
      ref = b;  // superclass reference to subclass object
      g.drawString( ref.toString() + " earned $" +ref.earnings(), 25,
25 );
      g.drawString( b.toString() + " earned $" +   b.earnings(), 25,
40 );
      ref = c;  // superclass reference to subclass object
      g.drawString( ref.toString() + " earned $" + ref.earnings(),
25, 55 );
      g.drawString( c.toString() + " earned $" + c.earnings(), 25, 70
);
      ref = p;  // superclass reference to subclass object
      g.drawString( ref.toString() + " earned $" + ref.earnings(),
25, 85 );
```

```
      g.drawString( p.toString() + " earned $" + p.earnings(), 25,
100 );

      ref = h;  // superclass reference to subclass object
      g.drawString( ref.toString() + " earned $" +
                    ref.earnings(), 25, 115 );
      g.drawString( h.toString() + " earned $" +
                    h.earnings(), 25, 130 );
   }
}
```

**Z) ΧΡΗΣΗ ΓΡΑΦΙΚΩΝ**

```
// παρουσίαση των μεθόδων drawString, drawChars και drawBytes
″
import java.applet.Applet;
″
import java.awt.Graphics;
″

″
public class DrawSCB extends Applet {
″
   private String s = "Using drawString!";
″
   private char c[] = { 'c', 'h', 'a', 'r', 's', ' ', '8' };
   private byte b[] = { 'b', 'y', 't', 'e', 1, 2, 3 };

   public void paint( Graphics g )
   {
      // draw a string at location (100, 25) on the applet
      g.drawString( s, 100, 25 );

      // draw a series of characters at location (100, 50)
      g.drawChars( c, 2, 3,  100, 50 );

      // draw a series of bytes at location (100, 75)
      g.drawBytes( b, 0, 5, 100, 75 );
   }
}
```

---

```
// Παρουσίαση του κατασκευαστή Color
import java.applet.Applet;
import java.awt.Graphics;
import java.awt.Color;

public class ShowColors2 extends Applet {
   private float red, green, blue;

   public void init()
   {
      red = 0.1f;
      green = 0.21f;
      blue = 0.33f;
   }

   public void paint( Graphics g )
   {
       g.setColor( new Color( red, green, blue ) );
       g.drawString( "ABCDEFGHIJKLMNOPQRSTUVWXYZ", 60, 33 );
       showStatus( "Current RGB: " + g.getColor().toString() );
   }
}
```

```
// Παρουσίαση της μεθόδου
// Demonstrating the methods for
// retrieving font information
import java.applet.Applet;
import java.awt.Graphics;
import java.awt.Font;

public class DemoFont2 extends Applet {
   private Font f;

   public void init()
   {
      // create a font object: 24-point bold italic courier
      f = new Font( "Courier", Font.ITALIC + Font.BOLD, 24 );
   }

   public void paint( Graphics g )
   {
      int style, size;
      String s, name;

      g.setFont( f );          // set the current font to f
      style = f.getStyle();    // determine current font style

      if ( style == Font.PLAIN )
         s = "Plain ";
      else if ( style == Font.BOLD )
         s = "Bold ";
      else if ( style == Font.ITALIC )
         s = "Italic ";
      else    // bold + italic
         s = "Bold italic ";

      size = f.getSize();    // determine current font size
      s += size + " point ";
      name = f.getName();    // determine current font name
      s += name;
      g.drawString( s, 20, 40 );

      // display font family
      g.drawString( "Font family is " + f.getFamily(), 20, 60 );
  }
}
```

```
// Παρουσίαση isPlain, isBold και isItalic
import java.applet.Applet;
import java.awt.Graphics;
import java.awt.Font;

public class DemoFont3 extends Applet {
   private Font f;

   public void init()
   {
      // create a font object: 24-point bold italic courier
      f = new Font( "Courier", Font.ITALIC + Font.BOLD, 24 );
   }

   public void paint( Graphics g )
   {
      String s;
```

```
      g.setFont( f );          // set the current font to f

      if ( f.isPlain() == true )
         s = "Font is plain.";
      else if ( f.isBold() == true && f.isItalic() == false )
         s = "Font is bold.";
      else if ( f.isItalic() == true && f.isBold() == false )
         s = "Font is italic.";
      else    // bold + italic
         s = "Font is bold italic.";

      g.drawString( s, 20, 40 );
   }
}
```

**Η) ΓΡΑΦΙΚΑ ΕΝΔΙΑΜΕΣΑ ΧΡΗΣΤΩΝ (GUI : Graphics User Interfaces)**

```
// Δημιουργία πλήκτρων επιλογής (buttons)
″
import java.applet.Applet;
″
import java.awt.*;
″

″
public class MyButtons extends Applet {
   private Button pushButton1, pushButton2, pushButton3;

   public void init()
   {
      pushButton1 = new Button( "Click here" );
      pushButton2 = new Button( "Sorry I don't do anything" );
      pushButton3 = new Button();  // no button label

      // add buttons
      add( pushButton1 );
      add( pushButton2 );
      add( pushButton3 );
   }

   // handle the button events
   public boolean action( Event e,  Object o )
   {
      // check to see if a button triggered the event
      if ( e.target instanceof Button ) {
         // check to see if pushButton1 or pushButton3 was pressed.
         // Nothing will be done if  pushButton2 was pressed.
      ″
         if ( e.target == pushButton1 )
″
            showStatus( "You pressed: " + o.toString() );
         else if ( e.target == pushButton3 )
            showStatus( "You pressed: " + e.arg );
         return true;   // event was handled here
      }
      return true;
   }
}
```

---

```
// Αποτύπωση των κινήσεων ποντικιού  με χρήση «καμβά»
import java.applet.Applet;
import java.awt.*;

class NewCanvas extends Canvas {
   private int width, height;
   private int x1, y1;
```

```
   public void paint( Graphics g )
   {
      g.drawOval( x1, y1, width, height );
   }
   public boolean mouseDown( Event e, int x, int y )
   {
      x1 = x;
      y1 = y;
      return true;
   }
   public boolean mouseUp( Event e, int x, int y )
   {
      width = Math.abs( x - x1 );
      height = Math.abs( y - y1 );
      // determine the upper left point of the bounding rectangle
      x1 = Math.min( x1, x );
      y1 = Math.min( y1, y );
      repaint();
      return true;
   }
}

public class MyCanvas extends Applet {
   private NewCanvas c;

   public void init()
   {
      c = new NewCanvas();
      c.resize( 185, 70 );  // resize canvas
      c.setBackground( Color.yellow );

      add( c );    // add canvas to applet
   }
}
```

---

```
// ΧΡΗΣΗ ΜΕΝΟΥ ΣΕ ΜΙΑ ΕΦΑΡΜΟΓΗ βασισμένη σε πλαίσια
import java.awt.*;

public class ScratchPad extends Frame {
   private TextArea t;
   private Font f;

   public ScratchPad()
   {
      super( "ScratchPad Application" );

      t = new TextArea();
      add( "Center", t );

      f = new Font( "TimesRoman", Font.PLAIN, 12 );
      setFont( f );

      // create menubar
      MenuBar bar = new MenuBar();

      // create font menu
      Menu fontMenu = new Menu( "Font" );

      // add some items to the menu
      fontMenu.add( "Times Roman" );
      fontMenu.add( "Courier" );
      fontMenu.add( "Helvetica" );

      // add menu to menu bar
      bar.add( fontMenu );
```

```
      // set the menubar for the frame
      setMenuBar( bar );

      resize( 300, 200 );
      show();
   }

   public boolean handleEvent( Event e )
   {
      if ( e.id == Event.WINDOW_DESTROY ) {
         hide();      // hide frame
         dispose();   // free resources
         System.exit( 0 );  // terminate
         return true;
      }

      return super.handleEvent( e );
   }
   public boolean action( Event e, Object o )
   {
      if ( e.target instanceof MenuItem ) {
         if ( e.arg.equals( "Times Roman" ) )
            f = new Font( "TimesRoman", Font.PLAIN, 12 );
         else if ( e.arg.equals( "Courier" ) )
            f = new Font( "Courier", Font.PLAIN, 12 );
         else  // Helvetica
            f = new Font( "Helvetica", Font.PLAIN, 12 );
         t.setFont( f );
      }
      return true;
   }
   public static void main( String args[] )
   {
      ScratchPad e;
      e = new ScratchPad();
   }
}
```

---

```
// Δημιουργία πλήκτρων επικύρωσης
import java.applet.Applet;
import java.awt.*;

public class MyCheckbox extends Applet {
   private Font f;
   private TextField t;
   private Checkbox checkBold, checkItalic;
   public void init()
   {
      t = new TextField( "Sample Text", 30 );
      // instantiate checkbox objects
      checkBold = new Checkbox( "Bold" );
      checkItalic = new Checkbox();
      checkItalic.setLabel( "Italic" );  // set checkbox label
      f = new Font( "TimesRoman", Font.PLAIN, 14 );
      t.setFont( f );
      add( t );
      add( checkBold );     // unchecked (false) by default
      add( checkItalic );   // unchecked (false) by default
   }
   public boolean action( Event e, Object o )
   {
      int b, i;
      // Check for Checkbox event
      if ( e.target instanceof Checkbox ) {
         if ( checkBold.getState() == true  ) // test state of bold
checkbox
```

```
            b = Font.BOLD;
         else
            b = Font.PLAIN;   // value of 0

         if ( checkItalic.getState() == true ) // test state of
italic checkbox
            i = Font.ITALIC;
         else
            i = Font.PLAIN;   // value of 0

         f = new Font( "TimesRoman", b + i, 14 );
         t.setFont( f );
      }

      return true;
   }
}
```

**θ) ΔΙΑΧΕΙΡΙΣΗ ΕΞΑΙΡΕΣΕΩΝ**

```
// Παρουσίαση του «try-catch-finally» μηχανισμού εξαίρεσης
″
public class UsingExceptions {
″
   public static void main( String args[] )
″
   {  try {
″
         throwException();
″
      }
″
      catch ( Exception e )
      {
         System.err.println( "Exception handled in main" );
      }
      doesNotThrowException();
   }
   public static void throwException() throws Exception
   {  try {      // Throw an exception and immediately catch it.
         System.out.println( "Method throwException" );
         throw new Exception();  // generate exception
      }
      catch( Exception e )
      {  System.err.println( "Exception handled in " +"method
throwException" );
         throw e;  // rethrow exception for further processing
      }
      finally {System.err.println( "Finally is always executed" );
      }
   }
   public static void doesNotThrowException()
   {  try { System.out.println( "Method doesNotThrowException" );
      }
      catch( Exception e )
      { System.err.println( e.toString() ); }
      finally {
         System.err.println( "Finally is always executed." );
      }
   }
}
```

---

```
// Εξαιρέσεις σε μία ανέλιξη ουράς
public class UsingExceptions {
   public static void main( String args[] )
   {  try { throwException();    }
```

```
      catch ( Exception e )
      { System.err.println( "Exception handled in main" ); }
   }
   public static void throwException() throws Exception
   {
.      try {      // Throw an exception and catch it in main
         System.out.println( "Method throwException" );
         throw new Exception();  // generate exception
      }
      catch( OtherException e )
      { System.err.println( "Exception handled in " + "method
throwException" );
      }
      finally { System.err.println( "Finally is always executed" );
      }
   }
}

class OtherException extends Exception {
   public OtherException()
   {
      super( "Another exception type" );
   }
}
```

**Ι) ΔΙΑΧΕΙΡΙΣΗ ΠΟΛΛΑΠΛΩΝ ΝΗΜΑΤΩΝ (MULTI-THREADING)**

```
// ΠΟΛΛΑΠΛΑ ΝΗΜΑΤΑ ΕΚΤΥΠΩΣΗΣ ΣΕ  διαφορετικά χρονικά διαστήματα
″

″
public class PrintTest {
″
   public static void main( String args[] )
″
   {
″
      PrintThread thread1, thread2, thread3, thread4;

      thread1 = new PrintThread( "1" );
      thread2 = new PrintThread( "2" );
      thread3 = new PrintThread( "3" );
      thread4 = new PrintThread( "4" );

      thread1.start();
      thread2.start();
      thread3.start();
      thread4.start();
   }
}

class PrintThread extends Thread {
   int sleepTime;

   // PrintThread constructor assigns name to thread
   // by calling Thread constructor
   public PrintThread( String id )
   {
      super( id );

      // sleep between 0 and 5 seconds
      sleepTime = (int) ( Math.random() * 5000 );

      System.out.println( "Name: " + getName() +
```

```
                          ";  sleep: " + sleepTime );
   }

   // execute the thread
   public void run()
   {
      // put thread to sleep for a random interval
      try {
         sleep( sleepTime );
      }
      catch ( InterruptedException exception ) {
         System.err.println( "Exception: " +
                             exception.toString() );
      }

      // print thread name
      System.out.println( "Thread " + getName() );
   }
}
```

```
// ΠΟΛΛΑΠΛΑ ΝΗΜΑΤΑ ΤΡΟΠΟΠΟΙΗΣΗΣ ΔΙΑΜΟΙΡΑΖΟΜΕΝΩΝ ΑΝΤΙΚΕΙΜΕΝΩΝ

″
public class SharedCell {
″
   public static void main( String args[] )
   {
      HoldInteger h = new HoldInteger();
      ProduceInteger p = new ProduceInteger( h );
      ConsumeInteger c = new ConsumeInteger( h );
      p.start();
      c.start();
   }
}
class ProduceInteger extends Thread {
   private HoldInteger pHold;
   public ProduceInteger( HoldInteger h )
   {
      pHold = h;
   }
   public void run()
   {
      for ( int count = 0; count < 10; count++ ) {
         pHold.setSharedInt( count );
         System.out.println( "Producer set sharedInt to " +
                             count );
         // sleep for a random interval
         try {
            sleep( (int) ( Math.random() * 3000 ) );
         }
         catch( InterruptedException e ) {
            System.err.println( "Exception " + e.toString() );
         }
      }
   }
}
class ConsumeInteger extends Thread {
   private HoldInteger cHold;
   public ConsumeInteger( HoldInteger h )
   {
      cHold = h;
   }
   public void run()
   {
      int val;
      val = cHold.getSharedInt();
      System.out.println( "Consumer retrieved " + val );
      while ( val != 9 ) {
         // sleep for a random interval
         try {
            sleep( (int) ( Math.random() * 3000 ) );
         }
         catch( InterruptedException e ) {
            System.err.println( "Exception " + e.toString() );
         }

         val = cHold.getSharedInt();
         System.out.println( "Consumer retrieved " + val );
      }
   }
}
class HoldInteger {
   private int sharedInt;
   public void setSharedInt( int val ) { sharedInt = val; }
   public int getSharedInt() { return sharedInt; }
}
```

```
// ΠΟΛΛΑΠΛΑ ΝΗΜΑΤΑ ΤΡΟΠΟΠΟΙΗΣΗΣ ΔΙΑΜΟΙΡΑΖΟΜΕΝΩΝ ΑΝΤΙΚΕΙΜΕΝΩΝ
// ΜΕ ΚΑΤΑΛΛΗΛΟ ΣΥΓΧΡΟΝΙΣΜΟ

"
public class SharedCell {
   public static void main( String args[] )
   {  HoldInteger h = new HoldInteger();
      ProduceInteger p = new ProduceInteger( h );
      ConsumeInteger c = new ConsumeInteger( h );
      p.start();
      c.start();
   }
}
class ProduceInteger extends Thread {
   private HoldInteger pHold;
   public ProduceInteger( HoldInteger h )
   {     pHold = h;  }
   public void run()
   {  for ( int count = 0; count < 10; count++ ) {
         pHold.setSharedInt( count );
         System.out.println( "Producer set sharedInt to " + count );
         // sleep for a random interval
         try {sleep( (int) ( Math.random() * 3000 ) ); }
         catch( InterruptedException e ) {
            System.err.println( "Exception " + e.toString() );
         }
      }
   }
}
class ConsumeInteger extends Thread {
   private HoldInteger cHold;
   public ConsumeInteger( HoldInteger h )
   {     cHold = h;   }
   public void run()
   {  int val;
      val = cHold.getSharedInt();
      System.out.println( "Consumer retrieved " + val );
      while ( val != 9 ) {
         try {    // sleep for a random interval
            sleep( (int) ( Math.random() * 3000 ) );
         }
         catch( InterruptedException e ) {
            System.err.println( "Exception " + e.toString() );
         }
         val = cHold.getSharedInt();
         System.out.println( "Consumer retrieved " + val );
      }
   }
}
class HoldInteger {
   private int sharedInt;
   private boolean writeable = true;
   public synchronized void setSharedInt( int val )
   {  while ( !writeable ) {
         try {  wait(); }
         catch ( InterruptedException e ) {
            System.err.println( "Exception: " + e.toString() );
         }
      }
      sharedInt = val;
      writeable = false;
      notify();
   }
```

```
   public synchronized int getSharedInt()
″
   {
″
      while ( writeable ) {
″
         try {
″
            wait();
″
         }
″
         catch ( InterruptedException e ) {
″
            System.err.println( "Exception: " + e.toString() );
         }
      }

      writeable = true;
      notify();
      return sharedInt;
   }
}
```

---

```
// Παρουσίαση  του ενδιάμεσου νημάτων Runnable
″
import java.awt.*;
″
import java.applet.Applet;
″

″
public class RandomCharacters extends Applet
                              implements Runnable {

   String alphabet;
   TextField output1, output2, output3;
   Button button1, button2, button3;

   Thread thread1, thread2, thread3;

   boolean suspend1, suspend2, suspend3;
   public void init()
   {
      alphabet = new String( "ABCDEFGHIJKLMNOPQRSTUVWXYZ" );
      output1 = new TextField( 10 );
      output1.setEditable( false );
      output2 = new TextField( 10 );
      output2.setEditable( false );
      output3 = new TextField( 10 );
      output3.setEditable( false );
      button1 = new Button( "Suspend/Resume 1" );
      button2 = new Button( "Suspend/Resume 2" );
      button3 = new Button( "Suspend/Resume 3" );
      add( output1 );
      add( button1 );
      add( output2 );
      add( button2 );
      add( output3 );
      add( button3 );
   }
   public void start()
   {    // create threads and start every time start is called
      thread1 = new Thread( this, "Thread 1" );
```

```
      thread2 = new Thread( this, "Thread 2" );
      thread3 = new Thread( this, "Thread 3" );
      thread1.start();
      thread2.start();
      thread3.start();
   }

   public void stop()
   {
      // stop threads every time stop is called
      // as the user browses another Web page
      thread1.stop();
      thread2.stop();
      thread3.stop();
   }

   public boolean action( Event event, Object obj )
   {
      if ( event.target == button1 )
         if ( suspend1 ) {
            thread1.resume();
            suspend1 = false;
         }
         else {
            thread1.suspend();
            output1.setText( "suspended" );
            suspend1 = true;
         }
      else if ( event.target == button2 )
         if ( suspend2 ) {
            thread2.resume();
            suspend2 = false;
         }
         else {
            thread2.suspend();
            output2.setText( "suspended" );
            suspend2 = true;
         }
      else if ( event.target == button3 )
         if ( suspend3 ) {
            thread3.resume();
            suspend3 = false;
         }
         else {
            thread3.suspend();
            output3.setText( "suspended" );
            suspend3 = true;
         }
      return true;
   }
   public void run()
   {  int location;
      char display;
      String executingThread;
      while ( true ) {
         // sleep from 0 to 5 seconds
         try {
            Thread.sleep( (int) ( Math.random() * 5000 ) );
         }
         catch ( InterruptedException e ) {
            e.printStackTrace();
         }
         location = (int) ( Math.random() * 26 );
         display = alphabet.charAt( location );
         executingThread = Thread.currentThread().getName();
         if ( executingThread.equals( "Thread 1" ) )
```

```
            output1.setText( "Thread 1: " + display );
         else if ( executingThread.equals( "Thread 2" ) )
            output2.setText( "Thread 2: " + display );
         else if ( executingThread.equals( "Thread 3" ) )
            output3.setText( "Thread 3: " + display );
      }
   }
}
```

**Κ) ΠΡΟΓΡΑΜΜΑΤΑ ΕΦΑΡΜΟΓΩΝ ΔΙΚΤΥΩΝ**

```
// Χρήση ενός πλήκτρου για τη φόρτωση εγγράφου από κάποιο URL.
″
import java.awt.*;
″
import java.net.*;
″
import java.applet.Applet;

public class SiteSelector extends Applet {
   Site siteList[];  // array of sites
   public void init()
   {
      siteList = new Site[ 2 ];
      siteList[ 0 ] = new Site ( " Java Home ",
                      "http://www.javasoft.com/index.html" );
      siteList[ 1 ] = new Site ( "Deitel Home",
                      "http://www.deitel.com/index.html" );

      for ( int i = 0; i < siteList.length; i++ )
         add( new Button( siteList[ i ].getTitle() ) );
   }
   public boolean action( Event e, Object arg )
   {
      if ( e.target instanceof Button ) {
         String title;
         URL location;

         for ( int i = 0; i < siteList.length; i++ ) {
            title = siteList[ i ].getTitle();
            location =
               siteList[ i ].getLocation();

            if ( title.equals( arg.toString() ) ) {
               gotoSite( location );
               return true;  // event handled
            }
         }
      }

      return false;    // event not handled yet
   }
   public void gotoSite( URL loc )
   {
      // this must be executed in a broswer such as Netscape
      getAppletContext().showDocument( loc );
   }
}
class Site extends Button {
   private String title;
   private URL location;

   public Site( String siteTitle, String siteLocation )
   {
      title = siteTitle;

      try {
```

```
            location = new URL( siteLocation );
         }
         catch ( MalformedURLException e ) {
            System.err.println( "Invalid URL: " + siteLocation );
         }
      }
      public String getTitle() { return title; }
      public URL getLocation() { return location; }
}

// ΧΡΗΣΗ URL ΣΥΝΔΕΣΗΣ ΓΙΑ ΑΝΑΓΝΩΣΗ ΑΡΧΕΙΟΥ ΣΤΟΝ ΕΞΥΠΗΡΕΤΗ
″
import java.awt.*;
″
import java.net.*;
″
import java.io.*;
import java.applet.Applet;

public class ReadServerFile extends Applet {
   URL fileURL;
   TextArea contents;
   InputStream input;
   DataInputStream dataInput;

   public void init()
   {
      contents = new TextArea( "Please wait...", 10, 40 );
      add( contents );

      try {
         fileURL = new URL(
            "http://www.deitel.com/test/test.txt" );
      }
      catch ( MalformedURLException e ) {
         showStatus( "Exception: " + e.toString() );
      }
   }

   public void start()
   {
      String text;

      try {
         input = fileURL.openStream();
         dataInput = new DataInputStream( input );
         contents.setText( "The file contents are:\n" );

         while ( ( text = dataInput.readLine() ) != null )
            contents.appendText( text + "\n" );

         dataInput.close();
      }
      catch ( IOException e ) {
         showStatus( "Exception: " + e.toString() );
      }
   }
}
```

---

```
// ΠΡΟΕΤΟΙΜΑΣΙΑ ΠΕΛΑΤΗ ΠΟΥ ΘΑ ΔΙΑΒΑΣΕΙ ΠΛΗΡΟΦΟΡΙΑ ΑΠΟ ΤΟΝ ΕΞΥΠΗΡΕΤΗ
import java.io.*;
import java.net.*;
import java.awt.*;
```

```
public class Client extends Frame {
   TextArea display;

   public Client()
   {
      super( "Client" );
      display = new TextArea( 20, 10 );
      add( "Center", display );
      resize( 300, 150 );
      show();
   }

   public void runClient()
   {
      Socket client;
      InputStream input;

      try {
         client = new Socket( InetAddress.getLocalHost(),
                              5000 );
         display.appendText( "Created Socket\n" );

         input = client.getInputStream();
         display.appendText( "Created input stream\n" );

         display.appendText(
            "The text from the server is:\n\t");
         char c;

         while ( ( c = (char) input.read() ) != '\n' )
            display.appendText( String.valueOf( c ) );

         display.appendText( "\n" );
         client.close();
      }
      catch ( IOException e ) {
         e.printStackTrace();
      }
   }

   public boolean handleEvent( Event e )
   {
      if ( e.id == Event.WINDOW_DESTROY ) {
         hide();
         dispose();
         System.exit( 0 );
      }

      return super.handleEvent( e );
   }

   public static void main( String args[] )
   {
      Client c = new Client();

      c.runClient();
   }
}
```

**// ΠΡΟΕΤΟΙΜΑΣΙΑ ΕΞΥΠΗΡΕΤΗ ΠΟΥ ΘΑ ΠΑΡΑΛΑΒΕΙ ΣΥΝΔΕΣΗ ΑΠΟ ΤΟΝ ΠΕΛΑΤΗ, ΘΑ ΑΠΟΣΤΕΙΛΛΕΙ ΜΙΑ // ΣΥΜΒΟΛΟΣΕΙΡΑ ΧΑΡΑΚΤΗΡΩΝ ΣΤΟΝ ΠΕΛΑΤΗ ΚΑΙ ΘΑ ΚΛΕΙΣΕΙ ΤΗ ΣΥΝΔΕΣΗ**

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

```
public class Server extends Frame {
   TextArea display;

   public Server()
   {
      super( "Server" );
      display = new TextArea( 20, 5 );
      add( "Center", display );
      resize( 300, 150 );
      show();
   }

   public void runServer()
   {
      ServerSocket server;
      Socket connection;
      OutputStream output;

      try {
         server = new ServerSocket( 5000, 100 );
         connection = server.accept();
         display.setText( "Connection received...\n" );
         display.appendText( "Sending data...\n" );
         output = connection.getOutputStream();
         String s = new String( "Connection successful\n");

         for ( int i = 0; i < s.length(); i++ )
            output.write( (int) s.charAt( i ) );

         display.appendText(
            "Transmission complete. Closing socket.\n" );
         connection.close();
      }
      catch ( IOException e ) {
         e.printStackTrace();
      }
   }

   public boolean handleEvent( Event e )
   {
      if ( e.id == Event.WINDOW_DESTROY ) {
         hide();
         dispose();
         System.exit( 0 );
      }

      return super.handleEvent( e );
   }

   public static void main( String args[] )
   {
      Server s = new Server();

      s.runServer();
   }
}
```

```
// ΠΡΟΕΤΟΙΜΑΣΙΑ ΠΕΛΑΤΗ ΠΟΥ ΘΑ ΑΠΟΣΤΕΙΛΛΕΙ ΠΑΚΕΤΑ ΣΤΟΝ ΕΞΥΠΗΡΕΤΗ ΚΑΙ
ΘΑ ΠΑΡΑΛΑΒΕΙ
// ΠΑΚΕΤΑ ΑΠΟ ΤΟΝ ΕΞΥΠΗΡΕΤΗ

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

```
public class Client extends Frame {
   TextField enter;
   TextArea display;
   Panel enterPanel;
   Label enterLabel;

   DatagramPacket sendPacket, receivePacket;
   DatagramSocket sendSocket, receiveSocket;

   public Client()
   {
      super( "Client" );
      enterPanel = new Panel();
      enterLabel = new Label( "Enter message:" );
      enter = new TextField( 20 );
      enterPanel.add( enterLabel );
      enterPanel.add( enter );
      add( "North", enterPanel );
      display = new TextArea( 20, 10 );
      add( "Center", display );
      resize( 400, 300 );
      show();

      try {
         sendSocket = new DatagramSocket();
         receiveSocket = new DatagramSocket( 5001 );
      }
      catch( SocketException se ) {
         se.printStackTrace();
         System.exit( 1 );
      }
   }

   public void waitForPackets()
   {
      while ( true ) {
         try {
            // set up packet
            byte array[] = new byte[ 100 ];
            receivePacket =
               new DatagramPacket( array, array.length );

            // wait for packet
            receiveSocket.receive( receivePacket );

            // process packet
            display.appendText( "\nPacket received:" +
               "\nFrom host: " + receivePacket.getAddress() +
               "\nHost port: " + receivePacket.getPort() +
               "\nLength: " + receivePacket.getLength() +
               "\nContaining:\n\t" );
            byte data[] = receivePacket.getData();
            String received = new String( data, 0 );
            display.appendText( received );
         }
         catch( IOException exception ) {
            display.appendText( exception.toString() + "\n" );
            exception.printStackTrace();
         }
      }
   }

   public boolean handleEvent( Event e )
   {
      if ( e.id == Event.WINDOW_DESTROY ) {
```

```
         hide();
         dispose();
         System.exit( 0 );
      }

      return super.handleEvent( e );
   }

   public boolean action( Event event, Object o )
   {
      try {
         display.appendText( "\nSending packet containing: " +
                             o.toString() + "\n" );
         String s = o.toString();
         byte data[] = new byte[ 100 ];
         s.getBytes( 0, s.length(), data, 0 );
         sendPacket = new DatagramPacket( data, s.length(),
            InetAddress.getLocalHost(), 5000 );
         sendSocket.send( sendPacket );
         display.appendText( "Packet sent\n" );
      }
      catch ( IOException exception ) {
         display.appendText( exception.toString() + "\n" );
         exception.printStackTrace();
      }

      return true;
   }

   public static void main( String args[] )
   {
      Client c = new Client();

      c.waitForPackets();
   }
}
```

**// ΠΡΟΕΤΟΙΜΑΣΙΑ ΕΞΥΠΗΡΕΤΗ ΠΟΥ ΘΑ ΠΑΡΑΛΑΒΕΙ ΠΑΚΕΤΑ ΑΠΟ ΕΝΑ ΠΕΛΑΤΗ ΚΑΙ ΘΑ ΑΠΟΣΤΕΙΛΛΕΙ**
**// ΠΑΚΕΤΑ ΣΤΟΝ ΠΕΛΑΤΗ**

```
//
// Set up a Server that will receive packets from a
//
// client and send packets to a client.
//
import java.io.*;
//
import java.net.*;
//
import java.awt.*;

public class Server extends Frame {
   TextArea display;

   DatagramPacket sendPacket, receivePacket;
   DatagramSocket sendSocket, receiveSocket;

   public Server()
   {
      super( "Server" );
      display = new TextArea( 20, 10 );
      add( "Center", display );
      resize( 400, 300 );
      show();
      try {
```

```
            sendSocket = new DatagramSocket();
            receiveSocket = new DatagramSocket( 5000 );
         }
         catch( SocketException se ) {
            se.printStackTrace();
            System.exit( 1 );
         }
      }
      public void waitForPackets()
      {
         while ( true ) {
            try {
               // set up packet
               byte array[] = new byte[ 100 ];
               receivePacket =
                  new DatagramPacket( array, array.length );

               // wait for packet
               receiveSocket.receive( receivePacket );

               // process packet
               display.appendText( "\nPacket received:" +
                  "\nFrom host: " + receivePacket.getAddress() +
                  "\nHost port: " + receivePacket.getPort() +
                  "\nLength: " + receivePacket.getLength() +
                  "\nContaining:\n\t" );
               byte data[] = receivePacket.getData();
               String received = new String( data, 0 );
               display.appendText( received );

               // echo information from packet back to client
               display.appendText( "\n\nEcho data to client...");
               sendPacket = new DatagramPacket( data, data.length,
                  receivePacket.getAddress(), 5001 );
               sendSocket.send( sendPacket );
               display.appendText( "Packet sent\n" );
            }
            catch( IOException exception ) {
               display.appendText( exception.toString() + "\n" );
               exception.printStackTrace();
            }
         }
      }
      public boolean handleEvent( Event e )
      {
         if ( e.id == Event.WINDOW_DESTROY ) {
            hide();
            dispose();
            System.exit( 0 );
         }
         return super.handleEvent( e );
      }
      public static void main( String args[] )
      {
         Server s = new Server();
         s.waitForPackets();
      }
}
```