# ΚΕΦΑΛΑΙΟ 1

# ΕΙΣΑΓΩΓΗ ΣΤΗΝ JAVA

## 1.1 Εισαγωγή

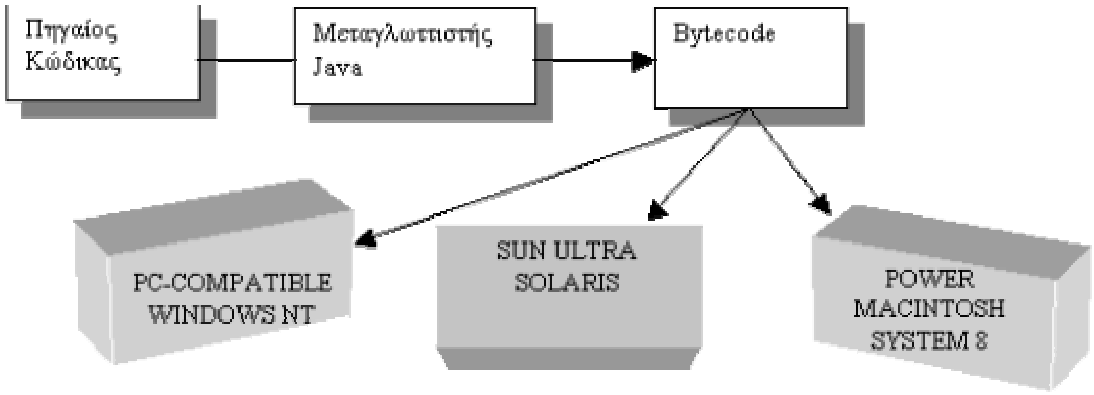
Η Java είναι δύο πράγματα μαζί, γλώσσα προγραμματισμού και πλατφόρμα. Η Java είναι υψηλού επιπέδου γλώσσα με τα ακόλουθα χαρακτηριστικά :

- **Αντικειμενοστρεφής**
- **Ασφαλής (secure)**: Απαγορεύει στους εισβολείς να γράφουν προγράμματα που μπορούν να χαλάσουν τα συστήματα των χρηστών.
- **Μεταφερτή (portable)**: Μπορεί να εκτελείται σε διαφορετικές πλατφόρμες χωρίς τροποποίηση όπως σε Windows, σε Macintosh.
- **Κατανεμημένη (distributed)**: Δηλαδή, διαφορετικά κομμάτια ενός προγράμματος μπορούν να προέλθουν από διαφορετικές πηγές.
- **Πολυνηματική (multithreaded)**: Δηλαδή ένα απλό πρόγραμμα σε Java μπορεί να κάνει πολλά προγράμματα ανεξάρτητα και αλληλεπιδρώντα.

## 1.2 Η Java δεν εξαρτάται από πλατφόρμα

***Ανεξαρτησία από πλατφόρμα*** είναι η δυνατότητα το ίδιο πρόγραμμα να εκτελείται σε διαφορετικές πλατφόρμες και λειτουργικά συστήματα .

Τα προγράμματα Java επιτυγχάνουν αυτή την ανεξαρτησία μέσω της χρήσης μιας ***εικονικής μηχανής*** (*Virtual Machine*) η οποία είναι κάτι σαν ένας υπολογιστής μέσα στον υπολογιστή. Ο πηγαίος κώδικας της Java μεταγλωττίζεται μέσω του ***compiler*** (μεταγλωττιστή). Η εικονική μηχανή παίρνει τα μεταγλωττισμένα προγράμματα Java και μετατρέπει τις εντολές τους σε εντολές που μπορεί να χειριστεί ένα λειτουργικό σύστημα. Το ίδιο μεταγλωττισμένο πρόγραμμα, που υπάρχει σε μια μορφή που καλείται ***bytecode***, μπορεί να εκτελεσθεί σε οποιαδήποτε άλλη πλατφόρμα και λειτουργικό σύστημα που διαθέτει μια εικονική μηχανή της Java. Ο bytecode ενός προγράμματος Java μπορεί να τρέξει σε Windows NT, Solaris, Macintosh χωρίς καμιά αλλαγή όπως φαίνεται στο παρακάτω σχήμα.




*Σχήμα 1.1- Διαδικασίες ενός προγράμματος σε Java*

***Bytecode*** είναι η έκδοση κώδικα μηχανής της εικονικής μηχανής (VM) της Java, δηλαδή οι εντολές που καταλαβαίνει άμεσα. Η ***εικονική μηχανή*** (VM) είναι επίσης γνωστή σαν ***διερμηνευτής Java*** *(Java interpreter).*

Η Java είναι επίσης ανεξάρτητη πλατφόρμας σε επίπεδο πηγαίου κώδικα. Τα προγράμματα της Java αποθηκεύονται σαν αρχεία κειμένου πριν να μεταγλωττιστούν και τα αρχεία αυτά μπορούν να δημιουργηθούν σε οποιαδήποτε πλατφόρμα που υποστηρίζει την Java. Π.χ. μπορείτε να γράψετε ένα πρόγραμμα της Java σε ένα Macintosh και να το μεταγλωττίσετε σε bytecode σε ένα μηχάνημα με Windows 95.

Η δυνατότητα ένα αρχείο bytecode να εκτελείται σε διάφορες πλατφόρμες είναι αυτή που κάνει την Java να εργάζεται στο World Wide Web, επειδή το ίδιο το Web είναι ανεξάρτητο από πλατφόρμα. Όπως τα αρχεία HTML μπορούν να διαβαστούν σε οποιαδήποτε πλατφόρμα, οι ***μικροεφαρμογές*** (applets) της Java μπορούν να εκτελεστούν σε οποιαδήποτε πλατφόρμα που διαθέτει ένα εξεταστή (browser – φυλλομετρητή) με δυνατότητες Java.

---

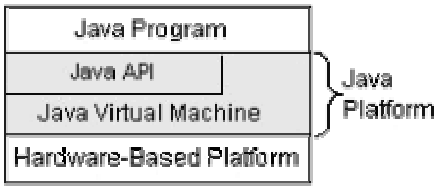
# 1.3 Η πλατφόρμα Java

---

Η πλατφόρμα Java περιλαμβάνει δύο συστατικά:

- Την εικονική μηχανή (VM)
- Το Application programming interface (API)

Το παρακάτω σχήμα απεικονίζει ένα πρόγραμμα σε Java το οποίο τρέχει σε πλατφόρμα της Java. Όπως φαίνεται, το API και η εικονική μηχανή της Java απομονώνουν το πρόγραμμα από εξαρτήσεις του hardware .



*Σχήμα 1.2 - Πλατφόρμα Java*

Το ***Java API*** είναι μια μεγάλη συλλογή από έτοιμα προγράμματα της Java τα οποία παρέχουν χρήσιμες δυνατότητες σε ένα υπό κατασκευή πρόγραμμα .Το Java API είναι οργανωμένο σε βιβλιοθήκες (packages) κώδικα από συστατικά που σχετίζονται μεταξύ τους.

---

# 1.4 Περιβάλλον και εργαλεία της Java

---

Υπάρχουν πολλά πακέτα λογισμικού που μπορείτε να χρησιμοποιήσετε για να δημιουργήσετε προγράμματα σε Java. Ένα από τα πιο δημοφιλή πακέτα είναι το Java Development Kit (JDK) της Sun Microsystems το οποίο χρησιμοποιούν τα προγράμματα αυτού του εγχειριδίου. Το JDK χρησιμοποιεί την γραμμή εντολών (DOS prompt) στα συστήματα Windows 95/98/NT/2000. Αναλυτικές πληροφορίες για το JDK υπάρχουν στο παράρτημα Α.

Όπως με τις περισσότερες γλώσσες προγραμματισμού, τα πηγαία αρχεία Java αποθηκεύονται σαν αρχεία καθαρού κειμένου. Μπορείτε να τα δημιουργήσετε με οποιοδήποτε κειμενογράφο ή επεξεργαστή κειμένου που μπορεί να αποθηκεύσει καθαρό κείμενο, μια μορφοποίηση που καλείται επίσης ASCII ή DOS text.

Τα πηγαία αρχεία Java πρέπει να αποθηκευτούν με επέκταση ***.java***. Τα πηγαία αρχεία Java μεταγλωττίζονται σε bytecode με επέκταση ονόματος αρχείου ***.class***. Τα αρχεία αυτά ονομάζονται ***κλάσεις***. Κατά κάποιο τρόπο, ο όρος ***κλάση*** *(*class*)* στην Java είναι συνώνυμος με τον όρο πρόγραμμα.

Τα τρία πιο χρήσιμα εργαλεία της Java είναι ο μεταγλωττιστής ***javac***, ο διερμηνευτής ***java*** και ο εξεταστής ***appletviewer*** .

Ο μεταγλωττιστής **javac**, μετατρέπει τον πηγαίο κώδικα Java σε ένα ή περισσότερα αρχεία κλάσεων bytecode που μπορούν να εκτελεστούν από ένα διερμηνευτή **java**. Για την μεταγλώττιση ενός αρχείου πηγαίου κώδικα Java σε bytecode, το εργαλείο **javac** εκτελείται με το όνομα του αρχείου σαν όρισμα .

Π.χ.

```
javac MyProgram.java
```

Το παραγώμενο αρχείο από την παραπάνω εντολή είναι το ‘MyProgram.class’ το οποίο είναι ένα αρχείο κλάσης και είναι σε μορφή bytecode.

Ο διερμηνευτής **java**, χρησιμοποιείται για εκτέλεση εφαρμογών Java από την γραμμή εντολών. Δέχεται σαν όρισμα το όνομα ενός αρχείου κλάσης προς εκτέλεση.

Π.χ.

```
java MyProgram
```

Αν και τα αρχεία κλάσης της Java έχουν επέκταση .class η επέκταση αυτή δεν καθορίζεται στην εντολή εκτελεσής τους. Το αρχείο κλάσης που φορτώνεται από τον διερμηνευτή **java** πρέπει να περιέχει μια μέθοδο (συνάρτηση) `main(...)` που έχει την παρακάτω μορφή διαφορετικά θα εμφανισθεί μια ένδειξη λάθους. Αναλυτική αναφορά για την μέθοδο `main(...)` υπάρχει στο επόμενο κεφάλαιο.

Π.χ.

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

{

// Κώδικας μεθόδου

}
```

Ο προβολέας μικροεφαρμογών της Java, ***appletviewer*** χρησιμοποιείται για εκτέλεση προγραμμάτων της Java που απαιτούν έναν browser για να εκτελεστούν και παρουσιάζονται σαν μέρος μιας ιστοσελίδας. Ο `appletviewer` παίρνει μια ιστοσελίδα σαν όρισμα γραμμής εντολών .

Π.χ.

```
appletviewer MyProgram.html
```

# ΚΕΦΑΛΑΙΟ 2

# ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΕΦΗΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ

## 2.1 Αντικειμενοστρεφής προγραμματισμός

Στον ***αντικειμενοστρεφή προγραμματισμό*** (object oriented programming, OOP*)* ένα πρόγραμμα υπολογιστή είναι ένα σύνολο αλληλεπιδρώντων αντικειμένων. Μπορεί να ειπωθεί ότι αντικειμενοστρεφής προγραμματισμός είναι ένα τρόπος καλύτερης οργάνωσης προγραμμάτων.

Στον αντικειμενοστρεφή προγραμματισμό, ένα πρόγραμμα δεν αποτελείται από τα δεδομένα και τον κώδικα του, αλλά από ***αντικείμενα*** (objects) τα οποία εμπεριέχουν τα δεδομένα και τα οποία (αντικείμενα) ανταλλάσσουν μεταξύ τους πληροφορίες και μηνύματα προκειμένου να επιτευχθεί ο στόχος του προγράμματος.

### 2.1.1 Κλάσεις και αντικείμενα

Η ***κλάση*** (class) είναι ένα πρότυπο που χρησιμοποιείται για δημιουργία πολλαπλών αντικειμένων με παρόμοια χαρακτηριστικά .

Οι κλάσεις περιέχουν όλα τα χαρακτηριστικά ενός συγκεκριμένου συνόλου αντικειμένων. Όταν γράφετε ένα πρόγραμμα σε μια αντικειμενοστρεφή γλώσσα προγραμματισμού, δεν ορίζετε μεμονωμένα αντικείμενα αλλά κλάσεις αντικειμένων.

Στον πραγματικό κόσμο υπάρχουν πολλά αντικείμενα από το ίδιο είδος. Για παράδειγμα το ποδήλατο κάποιου είναι ένα από τα πολλά ποδήλατα που κυκλοφορούν. Στον αντικειμενοστρεφή προγραμματισμό το προαναφερθέν ποδήλατο είναι ένα αντικείμενο (instance - ομότυπο) της κλάσης των ποδηλάτων. Γενικά τα ποδήλατα έχουν μερικά στοιχεία που χαρακτηρίζουν το είδος τους (π.χ. έχουν φρένα, έχουν ταχύτητες) και μερικά χαρακτηριστικά που είναι διαφορετικά σε κάθε ποδήλατο (π.χ. τρέχουσα ταχύτητα, τρέχων ρυθμός, μέγεθος ρόδας). Όταν ένα εργοστάσιο φτιάχνει ποδήλατα τα φτιάχνει με τα ίδια χαρακτηριστικά . Θα ήταν πολύ αναποτελεσματικό αν έφτιαχνε για κάθε ποδήλατο ένα καινούργιο πατρόν. Στο συγκεκριμένο παράδειγμα μια κλάση είναι το πατρόν από ένα είδος ποδηλάτου και τα αντικείμενα της είναι όλα τα παραγόμενα ποδήλατα από αυτό το πατρόν.

Τα παραπάνω ισχύουν και στον αντικειμενοστρεφή προγραμματισμό. Ο προγραμματιστής έχει πλεονεκτήματα δημιουργώντας ένα προσχέδιο για αντικείμενα με ίδιες ιδιότητες. Δηλαδή μπορεί να λεχθεί σε πιο ελεύθερη μορφή ότι κλάση είναι το προσχέδιο ή το πρωτότυπο, το οποίο ορίζει κοινές μεταβλητές και κοινές μεθόδους (συναρτήσεις) για τα αντικείμενα του ίδιου είδους.

Σαν συμπέρασμα μπορεί να λεχθεί ότι ένα ***αντικείμενο*** (object) αναπαριστά μια ομάδα χαρακτηριστικών που εκτελούν κάποιες εργασίες και είναι αυτόνομο στοιχείο ενός προγράμματος. Τα αντικείμενα καλούνται επίσης και ***ομότυπα*** (instances).

---

### 2.1.2 Μεταβλητές και μέθοδοι

---

Ένα σημαντικό στοιχείο των κλάσεων είναι οι ιδιότητες τους που διαφοροποιούν ένα αντικείμενο από ένα άλλο της ίδιας κλάσης. Οι ιδιότητες ορίζονται με τις ***μεταβλητές***.

Υπάρχουν μεταβλητές ομότυπου και κλάσης. Η διαφορά τους είναι ότι ενώ η μεταβλητή κλάσης επιδρά σε όλα τα ομότυπα της κλάσης, η μεταβλητή ομότυπου επιδρά μόνο στο συγκεκριμένο αντικείμενο της κλάσης που την κάλεσε. Στο προηγούμενο παράδειγμα (στο κεφάλαιο 2.1.1) έστω ότι ορίζεται μια μεταβλητή που αναφέρεται στην κατάσταση των φρένων των ποδηλάτων και έχει τιμή **true** όταν δουλεύουν και **false** όταν δεν δουλεύουν. Έστω ότι η μεταβλητή έχει τιμή **false**. Αν οριστεί ως μεταβλητή κλάσης, όλα τα ποδήλατα (αντικείμενα) του συγκεκριμένου πατρόν (κλάσης) θα έχουν φρένα χαλασμένα. Στην περίπτωση που η κατάσταση των φρένων οριστεί ως μεταβλητή ομότυπου μόνο το συγκεκριμένο ποδήλατο (αντικείμενο) από όλα του προσχεδίου (κλάσης) θα έχει τα φρένα χαλασμένα.

Οι ***μέθοδοι*** *(methods)* είναι ομάδες προτάσεων που ορίζουν την συμπεριφορά των αντικειμένων της κλάσης. Χρησιμοποιούνται για να κάνουν συγκεκριμένες εργασίες, με τον ίδιο τρόπο που οι συναρτήσεις χρησιμοποιούνται σε άλλες γλώσσες προγραμματισμού.

Όπως υπάρχουν μεταβλητές ομότυπου και κλάσης, υπάρχουν επίσης μέθοδοι ομότυπου και κλάσης. Οι μέθοδοι ομότυπου που είναι πολύ συνηθισμένες, ονομάζονται συνήθως απλώς μέθοδοι, και εφαρμόζονται σε ένα αντικείμενο της κλάσης. Αν η μέθοδος κάνει μια αλλαγή σε ένα μεμονωμένο αντικείμενο, είναι μια μέθοδος ομότυπου. Οι μέθοδοι κλάσης έχουν επιρροή σε όλα τα αντικείμενα της ίδιας κλάσης. Οι μεταβλητές και οι μέθοδοι μπορούν να δηλωθούν με όποια σειρά επιθυμούμε. Εκτενής αναφορά για τις μεταβλητές και τις μεθόδους ομότυπου και κλάσεις υπάρχουν στα κεφάλαιο 5.4 και 5.5 αντίστοιχα.

---

### 2.1.3 Δημιουργία κλάσης

---

Ο βασικός ορισμός της κλάσης φαίνεται παρακάτω :

```
class <όνομα κλάσης> {
    ...
    // μέθοδοι
    // μεταβλητές
    ...
}

Π.χ
class MyProgram {
int a;
float b;
  void mymethod(void)
  {
        a=3;
        b=4.5f;
        System.out.println("O a="+a+" b="+b);
   }
}
```

*Σύνταξη 2.1 – Δημιουργία κλάσης*

Στο παραπάνω παράδειγμα ορίζονται δυο μεταβλητές οι `a,b` και μία μέθοδος η `mymethod`. Το αποτέλεσμα της μεταγλώττισης δίνει το αρχείο `MyProgram.class`.

Όταν εκτελείτε ένα δικό σας πρόγραμμα, τα αντικείμενα δημιουργούνται από τις κλάσεις του προγράμματός σας ανάλογα με τις ανάγκες σας. Στην επιτυχία ενός προγράμματος παίζει ρόλο το σύνολο των κλάσεων που απαρτίζουν το πρόγραμμα. Το ευτύχημα είναι ότι η `Java` περιλαμβάνει μια ομάδα κλάσεων που υλοποιούν το μεγαλύτερο μέρος της βασικής λειτουργικότητας που θα χρειαστείτε. Αυτές οι ομάδες καλούνται ***βιβλιοθήκες***. Η βιβλιοθήκη της `Java` περιλαμβάνει, μαθηματικές συναρτήσεις, χειρισμό κειμένου, γραφικά, αλληλεπίδραση με χρήστη κ.α.

---

## 2.2 Η πρώτη εφαρμογή σε Java

---

Εισάγετε σε έναν κειμενογράφο το παρακάτω πρόγραμμα και σώστε το με το όνομα `HelloJava.java`.

Π.χ.

```
class HelloJava{
```

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

{

System.out.println("This is my first Java program");

}

}
```

Όπως φαίνεται το όνομα του πηγαίου αρχείου πρέπει να είναι ίδιο με το όνομα της κλάσης αλλιώς το αρχείο δεν θα μεταγλωττιστεί. Για να μεταγλωττίσετε το πρόγραμμα από την γραμμή εντολών του DOS πληκτρολογείστε '`javac HelloJava.java`'. Το αποτέλεσμα θα είναι να δημιουργηθεί ένα αρχείο κλάσης με όνομα '`HelloJava.class`'. Αυτό είναι το αρχείο `Java bytecode` που μπορεί να εκτελεστεί από την εικονική μηχανή. Για την εκτέλεση του `HelloJava` πληκτρολογείστε '`java HelloJava`'. Το αποτέλεσμα είναι να εμφανιστεί στην γραμμή εντολών το μήνυμα '`This is my first Java program`'. Απαραίτητη προϋπόθεση είναι να βρίσκεστε στον ίδιο κατάλογο με το πρόγραμμα όταν το μεταγλωττίζετε και το εκτελείτε .

Η `Java` στις προηγούμενες εκδόσεις (πριν το JDK έκδοση 1.3) έκανε διακρίσεις μεταξύ μικρών και κεφαλαίων, κατά την προσπάθεια μεταγλώττισης και εκτέλεσης του προγράμματος. Στην τρέχουσα έκδοση (JDK 1.3) συνεχίζει να υπάρχει διάκριση μικρών και κεφαλαίων μόνο κατά την εκτέλεση του προγράμματος. Για αυτό θα εμφανιστεί μια ένδειξη λάθους στις προσπάθειες εκτέλεσης του παραπάνω προγράμματος σαν τις παρακάτω :

Π.χ.

```
java Hellojava.java

java helloJava.java
```

Το 'HelloJava.class ' περιλαμβάνει μία μέθοδο, τη main(...). Όπως και στη C, η μέθοδος main() μας δείχνει από που ξεκινά να εκτελείται μία εφαρμογή. Οι εντολές που περικλείονται μέσα στην main() είναι υπεύθυνες για την εκκίνηση ενός προγράμματος όπως η αρχικοποίηση μεταβλητών και η δημιουργία ομότυπων κλάσεων. Όταν εκτελείται ένα πρόγραμμα, η μέθοδος main() είναι το πρώτο πράγμα που καλείται εκτός της περίπτωσης των μικροεφαρμογών. Γι' αυτές (τις μικροεφαρμογές) υπάρχει εκτενής αναφορά στο έκτο κεφάλαιο.

Η μέθοδος main()δηλώνεται δημόσια (**public**), δηλαδή μπορούν να την καλέσουν από παντού. Δηλώνεται στατική (**static**), δηλαδή όλα τα παραδείγματα της κλάσης μοιράζονται την ίδια μέθοδο. Δηλώνεται κενή (**void**), που σημαίνει ότι αυτή η μέθοδος δεν επιστρέφει τιμή, όπως και στη C. Οι λέξεις **public**, **static**, **void** δεν είναι μεταβλητές αλλά δεσμευμένες λέξεις . Οι δεσμευμένες λέξεις δεν μπορούν να χρησιμοποιηθούν από τον προγραμματιστή σαν ονόματα μεταβλητών, κλάσεων ή μεθόδων. Τις έχει ορίσει η Java και έχουν μια ιδιαίτερη σημασία γιατί προσδιορίζουν ένα χαρακτηριστικό της γλώσσας . Δεσμευμένη λέξη είναι και η λέξη **class** .

Όταν καλείται η μέθοδος main(), τυπώνει το 'This is my first program' στην έξοδο. Αυτό επιτυγχάνεται με μέθοδο System.out.println(…). Για να είμαστε πιο ακριβείς, αυτό επιτυγχάνεται

φωνάζοντας την println() του πεδίου out που ανήκει στην κλάση System. η μέθοδος System.out.println() προσθέτει μια καινούρια γραμμή στο τέλος της εξόδου. Το όρισμα της είναι τύπου **String**.

Παρατηρήσεις: Μόνο οι κλάσεις που εκτελούνται με τον διερμηνευτή **java** χρειάζεται να έχουν την μέθοδο main(…). Οι βοηθητικές κλάσεις (δηλαδή οι κλάσεις που απλά μεταγλωττίζονται και δεν εκτελούνται) δεν χρειάζεται να έχουν την μέθοδο main(…).

---

## 2.3 Κληρονομικότητα

---

Η ***κληρονομικότητα*** (inheritance) είναι ένας μηχανισμός που επιτρέπει σε μια κλάση να κληρονομεί τις μεθόδους και τις μεταβλητές μιας άλλης κλάσης. Δηλαδή όταν μια κλάση `B` κληρονομεί από την κλάση `A` τότε τα αντικείμενα της κλάσης `B` έχουν πρόσβαση στα δεδομένα και τις μεθόδους της κλάσης `A` χωρίς να χρειάζεται να τα ξαναδηλώσουμε. Η αναφορά ότι η Β κλάση κληρονομεί από την `A` γίνεται με την δεσμευμένη λέξη **extends**.

Π.χ.

```
class A {

...

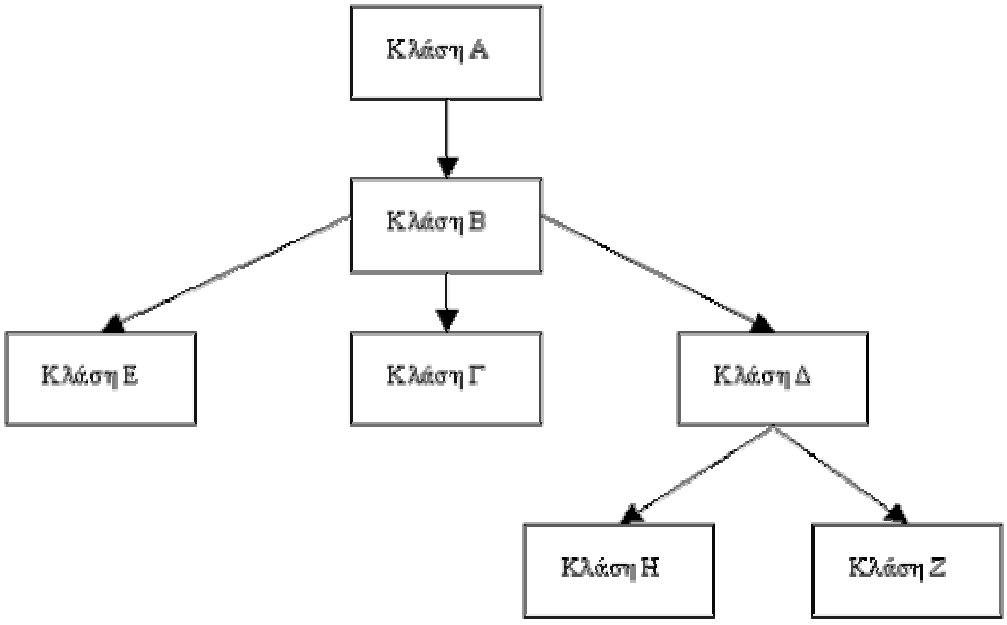
}


class B extends A {

...

}
```

Η κληρονομικότητα διατάσει τις κλάσεις σε μια αυστηρή ιεραρχία. Η κλάση που κληρονομεί από μια άλλη κλάση λέγεται ***δευτερεύουσα*** (subclass) ή ***υποκλάση*** και η κλάση που κληρονομείται λέγεται ***υπερκλάση*** (superclass). Μια κλάση μπορεί να έχει απεριόριστο αριθμό δευτερεύουσων (υποκλάσεων) κλάσεων αλλά μόνο μια υπερκλάση. Στο παρακάτω παράδειγμα γίνονται πιο κατανοητές οι παραπάνω έννοιες.




*Σχήμα 2.1- Παράδειγμα κληρονομικότητας*

Η κλάση `A` είναι υπερκλάση όλων των κλάσεων του σχήματος 2.1 διότι είναι ψηλότερα στην ιεραρχία. Η κλάση `B` έχει δευτερεύουσες κλάσεις την `E`, `Γ` και `Δ` ενώ οι `E`, `Γ`, `Δ` έχουν την κλάση `B` ως υπερκλάση. Η κλάση `Δ` είναι υπερκλάση των `H` και *Z* επομένως και η `B` είναι υπερκλάση των `H`, `Z` (μεταβατική ιδιότητα ).

Η `Java` χρησιμοποιεί την μονή μορφή κληρονομικότητας δηλαδή κάθε κλάση μπορεί να έχει μόνο μια υπερκλάση. Αντίθετα η `C++` χρησιμοποιεί την πολλαπλή κληρονομικότητα, δηλαδή μία κλάση μπορεί να έχει πολλές υπερκλάσεις. Αυτό όμως κάνει τους ορισμούς των κλάσεων και τον κώδικα παραγωγής τους πιο περίπλοκο.

# ΚΕΦΑΛΑΙΟ 3

## ΤΥΠΟΙ ΔΕΔΟΜΕΝΩΝ - ΤΕΛΕΣΤΕΣ

**Κατεβάστε το Κεφάλαιο 3 σε αρχείο μορφής pdf**



## 3.1 Σχόλια (Comments)

Τα ***σχόλια*** μπορούν να εμφανιστούν οπουδήποτε σε ένα αρχείο και ο σκοπός τους είναι να κάνουν κατανοητό το πρόγραμμα σε αυτούς που το μελετάνε. Ο μεταγλωττιστής της `Java` αγνοεί τελείως τα σχόλια, όταν προετοιμάζει μια εκτελέσιμη έκδοση (bytecode) του πηγαίου αρχείου `Java`. ***Σχόλιο*** θεωρείται ότι υπάρχει σε μια γραμμή μετά από δύο συνεχόμενα '/'. Επίσης μπορεί το σχόλιο να επεκταθεί σε περισσότερες από μια γραμμές αν συμπεριληφθεί ανάμεσα από το σύμβολο '/*' στην αρχή και "*/" στο τέλος όπως φαίνεται στο παρακάτω παράδειγμα. Ένας συμβολισμός με τα ίδια αποτελέσματα με τον '/*... */' είναι και ο '/** ... */'. Η διαφορά είναι πως ο τελευταίος χρησιμοποιείται από το εργαλείο javadoc για την δημιουργία documentation.

Π.χ.

```
// This is my first program in Java
class HelloJava{

     public static void main(String args[])
     {
      /* Print the line it is my first program */
     System.out.println("It is my first program");
     }
}
```

## 3.2 Μπλοκ

Η αρχή ενός ***μπλοκ*** δηλώνεται με το αριστερό άγκιστρο ‘{‘ και το τέλος ενός μπλοκ με το δεξί άγκιστρο ‘}’. Τα μπλοκ μπορούν να χρησιμοποιηθούν για να περιέχουν μεταβλητές και μεθόδους σε ένα ορισμό κλάσης. Επίσης για να ορίσουν τις προτάσεις που ανήκουν σε μια μέθοδο. Μπορούν ακόμα να χρησιμοποιηθούν μαζί με προτάσεις επανάληψης και προτάσεις υπό συνθήκη. Τα μπλοκ μπορούν να τοποθετηθούν μέσα σε άλλα μπλοκ .

Π.χ.

```
class A{

      firstmethod(){
      // σώμα μεθόδου
      }

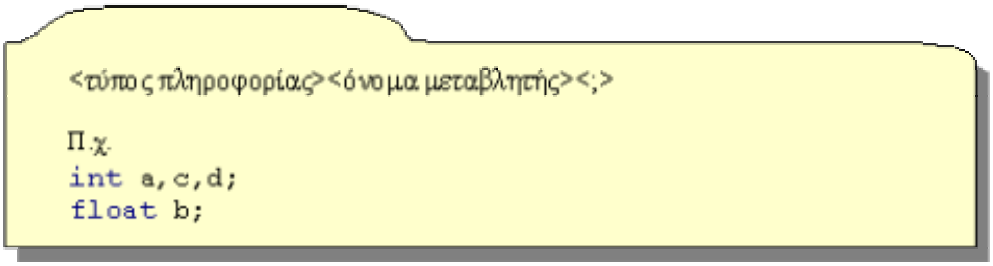
      secondmethod(){
      // σώμα μεθόδου
      }

}
```

# 3.3 Μεταβλητές

Οι εργασίες που γίνονται σε ένα πρόγραμμα Java μπορούν να διαιρεθούν σε επιμέρους σειρές προτάσεων. Μια ***πρόταση*** είναι μια εντολή που προκαλεί ένα συμβάν. Μια πρόταση καθορίζεται από την τοποθέτηση του ελληνικού ερωτηματικού στο τέλος της.

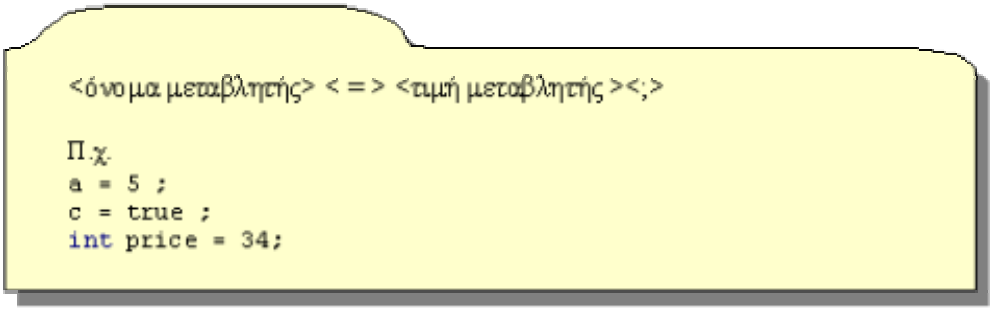
Οι μεταβλητές ορίζουν θέσεις όπου μπορούν να αποθηκευτούν πληροφορίες, όταν εκτελείται ένα πρόγραμμα. Η τιμή μπορεί να αλλάξει ανά πάσα στιγμή μέσα στο πρόγραμμα. Μια μεταβλητή για να δημιουργηθεί χρειάζεται δύο πράγματα: ένα όνομα γι’ αυτήν και τον τύπο της πληροφορίας που θα αποθηκεύσει. Ο τύπος πληροφοριών αναφέρεται πρώτος, ακολουθούμενος από το όνομα της μεταβλητής ή τα ονόματα των μεταβλητών που χωρίζονται με το κόμμα. Δηλαδή :




*Σύνταξη 3.1 – Σχηματισμός μεταβλητής (μη πλήρης μορφή)*

Η σύνταξη 3.1 είναι σωστή αλλά δεν είναι πλήρης. Ολοκληρωμένη σύνταξη των μεταβλητών υπάρχει στο κεφάλαιο 5.4.

Η ανάθεση τιμών σε μεταβλητές γίνεται με το σύμβολο της ισότητας όπως γίνεται στην σύνταξη 3.2.




*Σύνταξη 3.2 – Ανάθεση τιμής σε μεταβλητή*

Οι μεταβλητές της Java μπορούν να αρχίζουν με ένα γράμμα, ένα χαρακτήρα υπογράμμισης( _ ) ή ένα σύμβολο δολαρίου ($). Ο πρώτος χαρακτήρας απαγορεύεται να είναι αριθμός ενώ μετά από αυτόν επιτρέπεται οποιοσδήποτε συνδυασμός γραμμάτων ή χαρακτήρων. Μεταβλητές μπορεί να είναι εκτός από τους βασικούς τύπους δεδομένων που αναλύονται αμέσως μετά, και ονόματα κλάσεων και πίνακες.

---

## 3.4 Τύποι δεδομένων

---

Οι βασικοί τύποι ***δεδομένων*** της Java είναι οκτώ και έχουν το ίδιο μέγεθος και τα χαρακτηριστικά ανεξάρτητα από το λειτουργικό σύστημα και την πλατφόρμα στην οποία βρίσκονται. Στον πίνακα 3.1 φαίνεται το μέγεθος (σε bits) του κάθε τύπου δεδομένων.

| Τύπος | Μέγεθος (σε bits) |
|---|---|
| boolean | 8 |

| | |
|---|---|
| byte | 8 |
| char | 16 |
| short | 16 |
| int | 32 |
| long | 64 |
| float | 32 |
| double | 64 |

*Πίνακας 3.1- Μέγεθος βασικών τύπων δεδομένων*

---

## 3.5 Κυριολεκτικά

---

Τα ***κυριολεκτικά*** είναι οι τιμές που αναθέτονται στις μεταβλητές. Τα κυριολεκτικά που αρχίζουν με 0 είναι οκταδικά ενώ εκείνα που αρχίζουν με 0x ή 0X είναι δεκαεξαδικά. Οι χαρακτήρες `A` μέχρι `F` σε ένα δεκαεξαδικό αριθμό μπορούν να είναι ή πεζοί ή κεφαλαίοι.

Π.χ.

```
05675
```

```
0x34fad
```

```
255
```

Ένας αριθμός που περιέχει υποδιαστολή ή εκθέτη θεωρείται τύπου **double**. Τέτοιου είδους αριθμοί είναι οι παρακάτω :

Π.χ.

```
1.234
```

```
1.05e-21
```

Η μετατροπή ενός **double** αριθμού σε **float** γίνεται με την τοποθέτηση του ‘F’ ή του ‘f ‘ στο τέλος του αριθμού .

Π.χ.

```
3.534F
```

```
0.1233f
```

Οι ακέραιοι (**int**) μπορούν να γίνουν **long** βάζοντας στο τέλος τους ένα `L` ή `l`. Ο τύπος δεδομένων **boolean** μπορεί να πάρει δύο τιμές **true** η **false**. Τέλος τα κυριολεκτικά χαρακτήρων (**char**) εκφράζονται από ένα χαρακτήρα μέσα σε αποστρόφους ‘ ‘. Υπάρχουν και ορισμένοι μη εκτυπώσιμοι χαρακτήρες που δίνονται στον πίνακα 3.2.

| Κυριολεκτικό | Περιγραφή |
|---|---|
| \n | Αλλαγή γραμμής (newline) |
| \t | Στηλοθέτης (tab) |
| \b | Οπισθοδρόμηση(backspace) |
| \r | Επαναφορά |
| \f | Αλλαγή σελίδας |
| \’ | Απλό εισαγωγικό |
| \’’ | Διπλό εισαγωγικό |

*Πίνακας 3.2 – Μη εκτυπώσιμοι χαρακτήρες*

---

## 3.6 Πίνακες

---

Οι πίνακες στην `Java` δηλώνονται ως εξής :

<τύπος[]><μεταβλητή><;>
ή
<τύπος><μεταβλητή[]><;>

Π.χ.
```
int[] num;
char charact[];
```

*Σύνταξη 3.3 – Σχηματισμός πίνακα*

Ο τελευταίος πίνακας του παραδείγματος είναι διδιάστατος. Στους παραπάνω πίνακες δεν έχει ορίσθει το μέγεθος τους . Αυτό γίνεται με την σύνταξη 3.4.

<όνομα μεταβλητής>< = new><τύπος><[μέγεθος]>

Π.χ.
```
float b[];
b=new float[50];
int a[][];
a=new int[10][20];
String c[]=new String[10];
```

*Σύνταξη 3.4 - Ορισμός μεγέθους πίνακα*

Η αρχικοποίηση και η αναφορά στους πίνακες γίνεται όπως στα παρακάτω παραδείγματα .

<όνομα μεταβλητής[αριθμός στοιχειου]>

Π.χ.
```
int a[];
a=new int[2];
a[1]=10;
a[0]=3;
String[] str={"one","two","three"};
```

*Σύνταξη 3.5 – Αναφορά σε στοιχεία πινάκων*

Στην `Java` ένας πίνακας είναι ένα αντικείμενο. Η μεταβλητή `length` υπάρχει σε όλες τις κλάσεις των πινάκων και λαμβάνει προεπιλεγμένα το μέγεθος του πίνακα .

Π.χ.

```
String[] str=new String[23];

int i=str.length  // i=23
```

Όταν ένας πίνακας ορίζεται με μέγεθος N αυτό σημαίνει ότι έχει στοιχεία από 0 έως N-1.

---

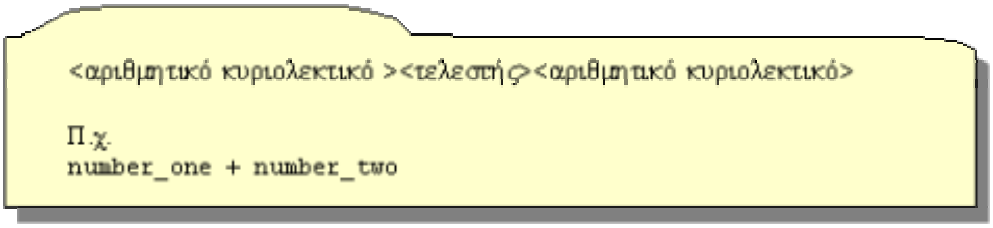
# 3.7 Τελεστές

---

### 3.7.1 Αριθμητικοί τελεστές

---

Η Java υποστηρίζει ποικιλία από αριθμητικούς τελεστές που ισχύουν και για ακέραιους αλλά και για πραγματικούς αριθμούς. Αυτοί οι τελεστές περιλαμβάνουν την πρόσθεση (+), αφαίρεση (-), πολλαπλασιασμό (*), διαίρεση (/) και το υπόλοιπο (%). Η σύνταξη των αριθμητικών τελεστών είναι:




*Σύνταξη 3.6 – Αριθμητικοί τελεστές*

Ο πίνακας 3.3 συνοψίζει τους αριθμητικούς τελεστές :

| Τελεστής | Παράδειγμα | Περιγραφή |
|---|---|---|

| | | |
|---|---|---|
| + | num1 + num2 | Πρόσθεση |
| - | num1 - num2 | Αφαίρεση |
| * | num1 * num2 | Πολλαπλασιασμός |
| / | num1 / num2 | Διαίρεση |
| % | num1 % num2 | Υπόλοιπο |

*Πίνακας 3.3- Αριθμητικοί τελεστές*

Τα ακόλουθα παραδείγματα πάνω στους τελεστές βοηθούν στην κατανόηση του θέματος .

Π.χ.

```
int b,c;      // Ορίζονται οι μεταβλητές b, c σαν ακέραιες μεταβλητές

c=5 ; b=21;  // Δίνονται αρχικές τιμές στις δύο μεταβλητές

c=c+5;        // Στην μεταβλητή c προστίθεται η τιμή 5 άρα  c=10

b=b+c;        // Τώρα η c έχει την τιμή 10 συν 21, άρα b=31
```

Στην ανάθεση πάντα υπολογίζεται πρώτα η τιμή της παράστασης δεξιά του "=" και μετά αντιστοιχίζεται στην μεταβλητή που βρίσκεται αριστερά του "=".

Υπάρχουν ορισμένες εκφράσεις στην ανάθεση τιμών (πίνακας 3.4) που μπορούν να έχουν και συντμημένη μορφή.

| Έκφραση | Ερμηνεία |
|---|---|
| num1 += num2 | Num1 = num1 + num2 |
| num1 -= num2 | num1 = num1 - num2 |
| num1 *= num2 | num1 = num1 * num2 |
| num1 /= num2 | num1 = num1 / num2 |

*Πίνακας 3.4 – Συντμημένη μορφή εκφράσεων*

Συνηθισμένες είναι οι εντολές που αυξάνουν την τιμή των δεδομένων κατά ένα ή την μειώνουν κατά ένα . Ένας τρόπος να γίνουν οι εντολές αυτές είναι με τους τελεστές προσαύξησης (++) και μείωσης (--).

Π.χ.

```
num=num+1; ⇔ num+=1; ⇔ num ++;
```

```
num=num-1; ⇔ num-=1; ⇔ num--;
```

Οι τελεστές προσαύξησης και μείωσης αποκαλούνται τελεστές προθέματος (prefix ) όταν αναφέρονται πριν από το όνομα μια μεταβλητής και επιθέματος όταν αναφέρονται μετά το όνομα (postfix).

Π.χ.

```
-- num   ++num   // τελεστές  προθέματος

num++     num--   // τελεστές επιθέματος
```

```
int a,b,c;

a=23;

b=a++;  // b=23 α=24

c=++a;  // c=25
```

Στο παραπάνω παράδειγμα θα περίμενε κανείς η μεταβλητή `b` να έχει την τιμή 24, άλλα αυτό δεν συμβαίνει διότι στην έκφραση '`b=a++`' υπάρχει τελεστής επιθέματος, δηλαδή πρώτα αντιστοιχεί η τιμή του `a` στην μεταβλητή `b` και μετά αυξάνεται η τιμή του `a`. Στην επόμενη έκφραση, στον τελεστή προθέματος, πρώτα αυξάνεται η τιμή του `a` και μετά αντιστοιχεί στην μεταβλητή `c`.

---

### 3.7.2 Τελεστές σύγκρισης

---

Οι ***τελεστές σύγκρισης*** χρησιμοποιούνται σε προτάσεις που επιστρέφουν τιμές **boolean** δηλαδή **true** αν η σύγκριση είναι αληθής και **false** αν η σύγκριση είναι ψευδής. Στον πίνακα 3.5 φαίνονται οι τελεστές σύγκρισης.

| Τελεστής σύγκρισης | Ερμηνεία |
|---|---|
| = = | Ισότητα |
| != | Ανισότητα |
| < | Μικρότερο |
| > | Μεγαλύτερο |
| <= | Μικρότερο ή ίσο |
| >= | Μεγαλύτερο ή ίσο |

*Πίνακας 3.5 – Τελεστές σύγκρισης*

Π.χ.

```
boolean c;

int a,b;

b=3; a=5;

c=b>a;      //  c=false

b+=3;

c=b>a;      //  c=true
```

---

### 3.7.3 Λογικοί τελεστές

---

Οι ***λογικοί τελεστές*** είναι οι **&**, |, **^, !** και εφαρμόζονται σε ορίσματα **boolean** για να εκτελέσουν τις λογικές πράξεις and, or, exclusive or και not αντίστοιχα.

Η λογική πράξη AND δίνει αποτέλεσμα **true** μόνο αν οι λογικές εκφράσεις που συνδέονται μαζί της είναι **true** .

Π.χ.

```
int a=6;

boolean c= (a<78) & (a>2); // a<78 αληθής και a >2 αληθής άρα c=true
```

Η λογική πράξη OR δίνει αποτέλεσμα true αν τουλάχιστον μια από τις λογικές εκφράσεις που συνδέονται μαζί της είναι **true .**

Π.χ.

```
int b=7;

boolean c= (b>23) | (b<1) | (b == 7); /* b>23 ψευδής , b<1 ψευδής αλλά b==7 αληθής άρα c=true */
```

Η λογική πράξη exclusive or δίνει αποτέλεσμα **true** αν οι δύο λογικές εκφράσεις έχουν αντίθετες τιμές .

Π.χ.

```
int a=12;

boolean c= (a>1) ^ (a = = 10);  // a>1 αληθής , a==10 ψευδής άρα c=true
```

Η λογική πράξη NOT χρησιμοποιεί μία λογική έκφραση που έπεται του τελεστή και σαν αποτέλεσμα έχει την αντιστροφή της τιμής της έκφρασης .

Π.χ.

```
boolean c= !(12 > 34); // 12>34 ψευδής άρα c=true
```

Οι παραπάνω τελεστές μπορούν και να συνδυαστούν μεταξύ τους σε περίπλοκες εκφράσεις. Τέλος υπάρχουν δύο τελεστές οι && και || οι οποίοι έχουν το ίδιο αποτέλεσμα με τους τελεστές & και | αντίστοιχα. Απλώς μειώνουν τις πράξεις που εκτελεί η Java. Στο παρακάτω παράδειγμα χρησιμοποιείται ο τελεστής && και έχει σαν αποτέλεσμα αφού η πρώτη έκφραση ‘a>5 ’ είναι ψευδής να μην εξετάζεται η δεύτερη έκφραση δηλαδή αντί για δύο συγκρίσεις πραγματοποιείται μόνο μία.

Π.χ.

```
int a=3;

boolean c=(a>5) && (a<23); // αφού α>5 ψευδής c=false
```

---

### 3.7.4 Δυαδικοί τελεστές

---

Οι πράξεις με ***δυαδικούς τελεστές*** πραγματοποιούνται με ακέραιους αριθμούς στο δυαδικό σύστημα. Στον πίνακα 3.6 φαίνονται οι δυαδικοί τελεστές και οι λειτουργίες τους .

| Τελεστής | Παράδειγμα | Λειτουργία |
|---|---|---|
| >> | num1 >> num2 | Ολίσθηση του num1 κατά num2 θέσεις δεξιά |
| << | num1 << num2 | Ολίσθηση του num1 κατά num2 θέσεις αριστερά |
| & | num1 & num2 | Δυαδικό and |
| \| | num1 \| num2 | Δυαδικό \| |
| ^ | num1 ^ num2 | Δυαδικό ^ |
| ~ | ~num1 | Συμπλήρωμα του num1 |

*Πίνακας 3.6 – Δυαδικοί τελεστές*

Το παρακάτω παράδειγμα θα κάνει πιο κατανοητές τις έννοιες των δυαδικών τελεστών .

Π.χ.

```
int a=12,b=13,c=2;

System.out.println(a>>c +"  "+a&b+" "+a^c); /* a>>c δίνει 3 , a&b δίνει 12 , a^c δίνει 14 */
```

a=12 στο δυαδικό  1100

b=13 στο δυαδικό &1101

1100 = 12

1100

c=2 στο δυαδικό ^ 0010

1110 = 14

Το αποτέλεσμα που θα δούμε στην έξοδο είναι:




---

### 3.7.5 Προτεραιότητα τελεστών

---

Σε μια έκφραση της Java μπορούν να χρησιμοποιούνται πολλοί τελεστές ταυτόχρονα. Η Java υπολογίζει αυτές τις εκφράσεις από αριστερά προς τα δεξιά της έκφρασης και τηρώντας την σειρά προτεραιότητας κάθε τελεστή. Η σειρά προτεραιότητας προσδιορίζεται στον πίνακα 3.7.

| Τελεστές | Σχόλια |
|---|---|
| . [ ] ( ) | |
| ++ -- ! ~ | Τελεστές προσαύξησης μείωσης, NOT, δυαδικό συμπλήρωμα |
| New (type) expression | Το new δημιουργεί νέα ομότυπα κλάσεων, η έκφραση type αλλάζει τον τύπο μιας μεταβλητή σε άλλο τύπο |
| * / % | Πολλαπλασιασμός ,διαίρεση , υπόλοιπο |
| + - | Πρόσθεση, αφαίρεση |
| << >> | Τελεστές ολίσθησης |
| < > <= >= | Συγκρίσεις |
| = = ! = | Ισότητα |
| & | AND |
| ^ | XOR |

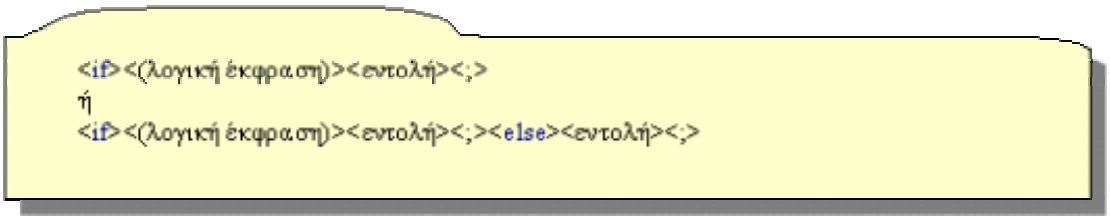
| | |
|---|---|
| \| | OR |
| && | AND |
| \|\| | OR |
| ? : | Χρησιμοποιείται όπως η συνθήκη ‘if’ |

*Πίνακας 3.7- Σειρά προτεραιότητας τελεστών*

# ΚΕΦΑΛΑΙΟ 4

# ΕΛΕΓΧΟΣ ΡΟΗΣ

Η συνθήκη **if** παρέχει την δυνατότητα στο πρόγραμμα να αποφασίσει με ορισμένα κριτήρια με ποιες εντολές θα συνεχίσει το πρόγραμμα. Η σύνταξή της είναι η παρακάτω .




*Σύνταξη 4.1 – Σχηματισμός συνθήκης if*

Όπου στις παραπάνω εκφράσεις υπάρχει η λέξη εντολή μπορούν να υπάρξουν περισσότερες από μια προτάσεις ανάμεσα σε μπλοκ. Η λογική έκφραση μπορεί να πάρει μόνο δύο τιμές **true** και **false**. Αν η λογική έκφραση στο **if** είναι **true** εκτελούνται οι εντολές μετά από αυτό. Αν όμως η λογική έκφραση είναι **false** τότε εκτελούνται οι εντολές μετά το **else**.

Π.χ

```
int a;
if(a>7){
..
}
else {
..
}
```

Υπάρχει μια άλλη μορφή της **else** η **else if** η οποία εκτελεί καταστάσεις βασισμένες σε εναλλακτικές εκφράσεις. Για παράδειγμα, ένας προγραμματιστής γράφει ένα πρόγραμμα στο οποίο εκχωρεί βαθμούς βασισμένος σε ένα τεστ επιδόσεων, δηλαδή το βαθμό A για επίδοση 90% και πάνω, το βαθμό B για επίδοση 80% και πάνω κ.ο.κ. Το παραπάνω πρόγραμμα υλοποιείται με μια σειρά από **else if**.

Π.χ

```
int score;
char grade;
if (score>=95){
grade='A';
}else if(score>=85){
grade='B';
}else if(score>=75){
grade='C';
}else if(score>=65){
grade='D';
}else{
grade='F';
}
```

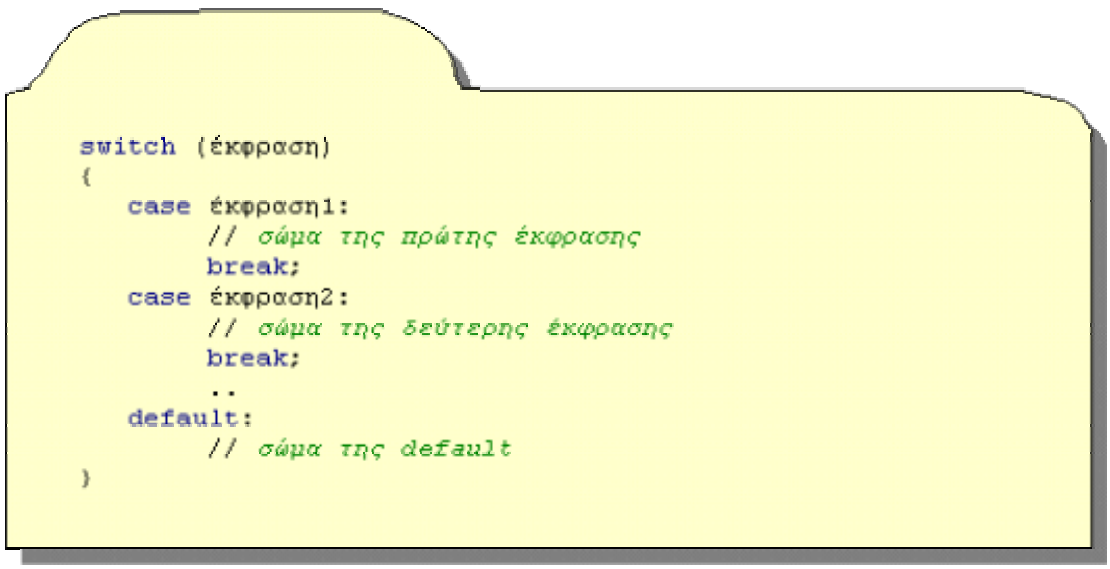
Η συνθήκη **if** μπορεί να έχει περισσότερα από ένα **else if** στην σύνθεση της αλλά μόνο ένα **else**. Όταν μια συνθήκη ισχύει τότε εκτελούνται οι εντολές στο σώμα της συνθήκης που ισχύει και μετά ο έλεγχος του προγράμματος βγαίνει από την συνθήκη **if**.

---

## 4.2 Συνθήκη switch

---

Η σύνταξη της εντολής **switch** είναι η παρακάτω.




*Σύνταξη 4.2 – Σχηματισμός συνθήκης switch*

Αρχικά υπολογίζεται η έκφραση του **switch** η οποία συγκρίνεται με όλες τις άλλες εκφράσεις μέχρι να βρεθεί κάποια ίδια. Αν βρεθεί κάποια όμοια εκτελούνται οι προτάσεις μετά από αυτή, αν δεν βρεθεί εκτελούνται οι προτάσεις μετά την λέξη **default**. Η **default** έκφραση είναι προαιρετική στο σχηματισμό της συνθήκης switch. Όταν εκτελείται η πρόταση **break** ο έλεγχος περνάει στο κύριο πρόγραμμα. Αν δεν βρεθεί η πρόταση **break** η ροή του προγράμματος συνεχίζει με τις υπόλοιπες προτάσεις της **switch**.

Η **switch** συνθήκη θέτει πολλούς περιορισμούς. Δηλαδή η έκφραση πρέπει να είναι μόνο τύπου **byte**, **char**, **short** ή **int**. Επίσης οι συγκρίσεις γίνονται μόνο για σχέσεις ισότητας και όχι ανισότητας όπως φαίνεται και στο παρακάτω παράδειγμα. Αντίθετα από την **if** συνθήκη η **switch** δεν χρειάζεται να περιλαμβάνει με αγκύλες τις προτάσεις από κάθε **case**.

Π.χ.

```
int day;
switch (day){
    case 1:System.out.println("Sunday");break;
    case 2:System.out.println("Monday");break;
    case 3:System.out.println("Tuesday");break;
    case 4:System.out.println("Wednesday");break;
    case 5:System.out.println("Thursday");break;
    case 6:System.out.println("Friday");break;
    case 7:System.out.println("Saturday");break;
    default: System.out.println("Such day does not exist");
}
```

Στο προηγηθέν πρόγραμμα ανάλογα με το αριθμό που έχει η μεταβλητή `day` τυπώνεται και η ανάλογη μέρα .

Παρατήρηση: Η εντολή **break** χρησιμοποιείται και σε άλλες περιπτώσεις και η λειτουργία της είναι να σταματά την εκτέλεση του προγράμματος στο τρέχον σημείο και να την συνεχίζει έξω από το επόμενο δεξιό άγκιστρο ‘ **}** ’.

---

# 4.3 Βρόχοι επανάληψης

---

### 4.3.1 Βρόχος for

---

Οι ***βρόχοι επανάληψης***, όπως συμπεραίνεται και από το όνομα, χρησιμοποιούνται για την επανάληψη μιας πρότασης ή ενός μπλοκ προτάσεων. Ο βρόχος **for** χρησιμοποιείται για συγκεκριμένο αριθμό επαναλήψεων μέχρι να ικανοποιηθεί μια συνθήκη. Η σύνταξή του είναι η ακόλουθη.

```
for (αρχικοποιήσεις ; έλεγχος ; αύξηση){
     // σώμα επαναλαμβανόμενων προτάσεων
}
```

*Συνθήκη 4.3 – Σχηματισμός βρόχου for*

- Αρχικοποιήσεις: Χρησιμοποιείται για να δώσει αρχικές τιμές στις μεταβλητές. Μπορούν να αρχικοποιηθούν περισσότερες από μία μεταβλητές οι οποίες θα χωρίζονται με κόμμα μεταξύ τους.
- Έλεγχος: Ο έλεγχος γίνεται σε κάθε επανάληψη του βρόχου με μια λογική έκφραση. Αν η λογική έκφραση δίνει την τιμή **true** ο βρόχος συνεχίζεται διαφορετικά διακόπτεται.
- Αύξηση: Χρησιμοποιείται για να αλλάξει την τιμή της μεταβλητής η μεταβλητών του βρόχου ώστε στις επόμενες επαναλήψεις να προκύψει η τιμή **false** στον έλεγχο και να τερματιστεί ο βρόχος. Μπορούν να τοποθετηθούν περισσότερες εκφράσεις σε αυτό το πεδίο οι οποίες θα χωρίζονται με κόμμα.

Π.χ.

```
for(int a=0;a<12;a++)

  System.out.println("A="+a);   // Ο βρόχος for εκτελείται 12 φορές
```

---

```
int a;
for(a=0;a>12;a++)
   /* Ο βρόχος for δεν επαναλαμβάνεται διότι η συνθήκη ελέγχου στην πρώτη επανάληψη δίνει false */
   System.out.println("A="+a);
```

Ένα πρόγραμμα που αντιμεταθέτει τα στοιχεία ενός πίνακα δηλαδή κάνει το πρώτο τελευταίο και το τελευταίο πρώτο είναι το παρακάτω.

Π.χ.

```
int temp;  // Η μεταβλητή αυτή χρησιμοποιείται για προσωρινή αποθήκευση
int a,b;
for(a=0,b=100;a<b;a++,b--)
{
    temp=pinakas[a];
```

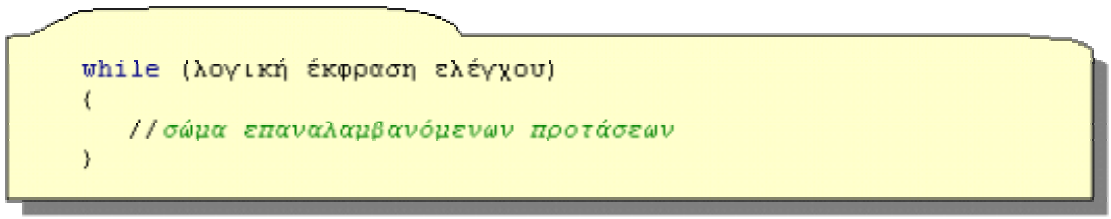
```
    pinakas[a]=pinakas[b];
    pinakas[b]=temp;
}
```

---

### 4.3.2 Βρόχος while

---

Ο βρόχος **while** έχει την ακόλουθη σύνταξη:




*Συνθήκη 4.4 – Σχηματισμός βρόχου while*

Οι προτάσεις μέσα στο σώμα της **while** επαναλαμβάνονται για όσες επαναλήψεις η λογική έκφραση είναι αληθής. Ο έλεγχος όπως και στην **if** γίνεται πριν την κάθε επανάληψη.

Π.χ.

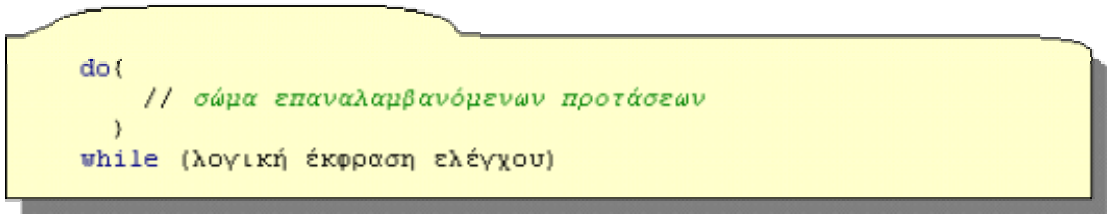
```
while(true)
    /* Επειδή η λογική έκφραση είναι πάντα true δημιουργείται ένας ατέρμων βρόχος  */
    System.out.println("Hello Java");
```

---

### 4.3.3 Βρόχος do-while

---

Η σύνταξή του βρόχου do-while είναι:




*Συνθήκη 4.5 – Σχηματισμός βρόχου do-while*

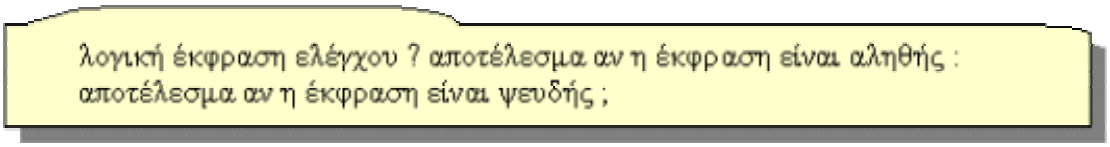
Οι εντολές μέσα στον βρόχο **do while** εκτελούνται μέχρι η λογική έκφραση να γίνει **false**. Η διαφορά από τις δύο προηγούμενες συνθήκες είναι ότι οι εντολές προς επανάληψη εκτελούνται τουλάχιστον μία φορά αφού ο έλεγχος του βρόγχου βρίσκεται στο τέλος της συνθήκης.

---

### 4.3.4 Τελεστής εκτέλεσης υπό συνθήκη

---

Μια εναλλακτική λύση στην χρησιμοποίηση της συνθήκης **if** είναι ο τελεστής εκτέλεσης υπό συνθήκη ο οποίος έχει σύνταξη.




*Συνθήκη 4.6 – Σχηματισμός τελεστή εκτέλεσης υπό συνθήκη*

Ο τελεστής εκτέλεσης υπό συνθήκη επιστρέφει μια τιμή ανάλογα αν η λογική έκφραση έχει αποτέλεσμα **true** τότε επιστρέφει το πρώτο αποτέλεσμα διαφορετικά επιστρέφει το δεύτερο. Στο παρακάτω παράδειγμα μια μεταβλητή παίρνει τιμή από τον τελεστή υπό συνθήκη .

Π.χ.

```
int a=23;

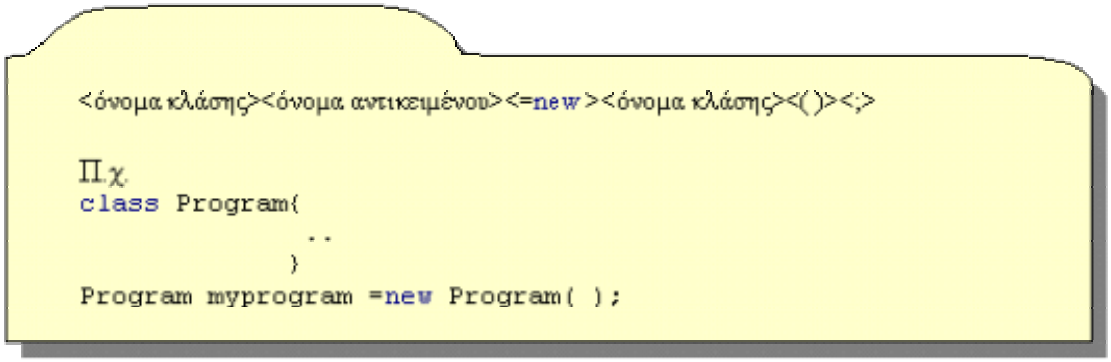
int b=a>12?45:32;  // το 23>12 άρα b=45
```

# ΚΕΦΑΛΑΙΟ 5

# ΑΝΤΙΚΕΙΜΕΝΑ - ΚΛΑΣΕΙΣ

## 5.1 Δημιουργία αντικειμένων

Για την δημιουργία ομότυπου μιας κλάση χρησιμοποιείται η παρακάτω σύνταξη.




*Σύνταξη 5.1 – Δημιουργία αντικειμένου*

Η παραπάνω έκφραση πραγματοποιεί τρεις ενέργειες.

- Με την δήλωση '`Program myprogram`' πληροφορεί τον μεταγλωττιστή ότι το όνομα `myprogram` θα είναι αντικείμενο της κλάσης `Program`. Να σημειωθεί ότι το όνομα της κλάσης χρησιμοποιείται σαν τύπος μεταβλητής .
- Με τον τελεστή **new** δημιουργεί ένα νέο αντικείμενο.
- Με την δήλωση `Program()` αρχικοποιεί το αντικείμενο .

## 5.2 Δήλωση - δημιουργία - αρχικοποίηση αντικειμένων

Η δήλωση αντικειμένων δεν είναι απαραίτητα μέρος της δημιουργίας αντικειμένων, αν και συχνά εμφανίζονται στην ίδια γραμμή. Όπως όλες οι δηλώσεις μεταβλητών έτσι και οι δηλώσεις αντικειμένων μπορούν να εμφανιστούν μόνες τους όπως στο παρακάτω παράδειγμα .

Π.χ.

```
Program myprogram;
```

Στο παραπάνω παράδειγμα ορίζουμε ένα αντικείμενο ονόματος `myprogram` της υπάρχουσας κλάσης `program`. Με αυτήν την δήλωση δεν δημιουργούμε ούτε αρχικοποιούμε το ομότυπο που ορίζουμε.

Για την δημιουργία αντικειμένων χρησιμοποιείται ο τελεστής **new** ο οποίος δεσμεύει για το καινούργιο αντικείμενο την απαραίτητη μνήμη . Ο τελεστής **new** απαιτεί την ύπαρξη ενός επιθεματικού ορίσματος που ονομάζεται δημιουργός (constructor). Δηλαδή ο τελεστής **new** δημιουργεί ένα αντικείμενο και ο δημιουργός το αρχικοποιεί .

Π.χ.

```
Program = myprogram; // Δημιουργία αντικειμένου

myprogram=new Program(100,200); // Αρχικοποίηση του αντικειμένου
```

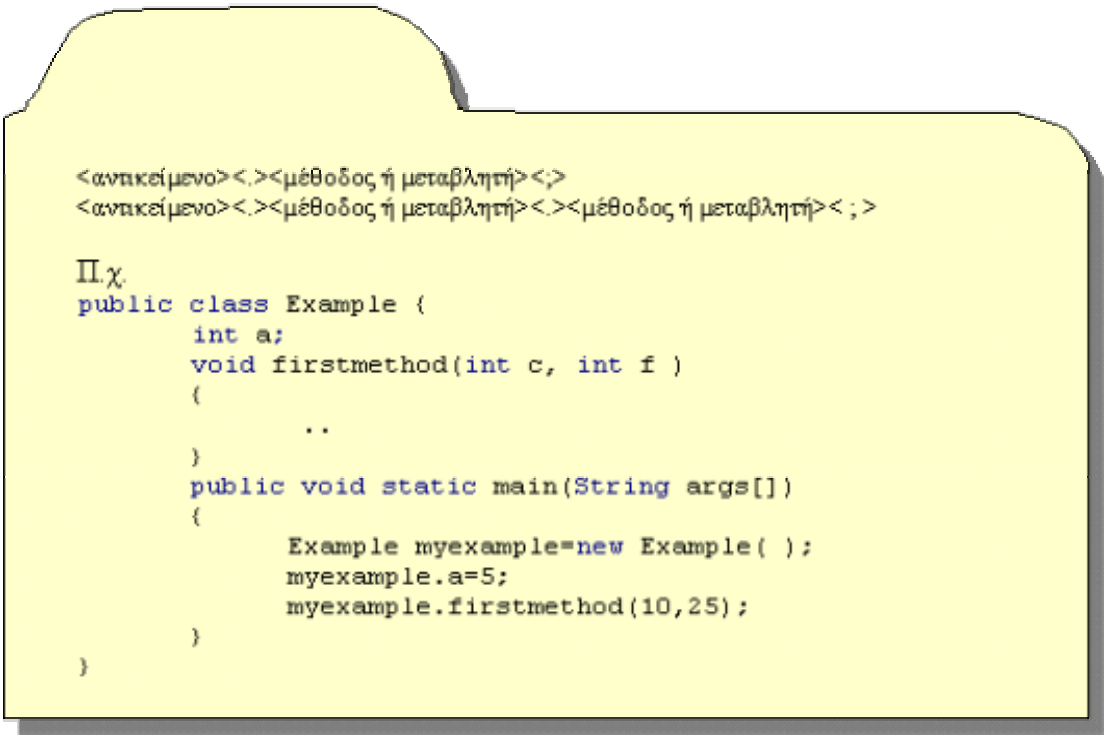
Το αντικείμενο `myprogram` παίρνει δύο ορίσματα τις τιμές 100, 200. Στη σύνταξη 5.1 είχε οριστεί ξανά το αντικείμενο `myprogram` αλλά χωρίς ορίσματα. Αυτό γίνεται διότι μια κλάση μπορεί να έχει περισσότερες από μία μεθόδους δημιουργούς. Οι μέθοδοι δημιουργοί αρχικοποιούν μια κλάση. Για αυτές τις μεθόδους θα υπάρχει εκτενής αναφορά στο κεφάλαιο 5.8.1.

---

## 5.3 Προσπέλαση των μελών αντικειμένου

---

Η προσπέλαση μεταβλητών και μεθόδων ενός αντικειμένου πραγματοποιείται με τον τελεστή τελείας όπως φαίνεται στην παρακάτω σύνταξη :




*Σύνταξη 5.2 – Προσπέλαση των μελών αντικειμένου*

Στο παραπάνω παράδειγμα το αντικείμενο `myexample` της κλάσης `Example` καλεί αρχικά την μεταβλητή `a`, της δίνει την τιμή πέντε και στην συνέχεια καλεί την μέθοδο `firstmethod(...)`.

Ένα πιο σύνθετο παράδειγμα προσπέλασης μεθόδου αντικειμένου είναι η μέθοδος που εκτυπώνει κείμενο στην οθόνη του υπολογιστή, η `System.out.println( )`. Η κλάση `System` υπάρχει στο πακέτο `java.lang`. Η `out` είναι μια μεταβλητή κλάσης που περιέχει ένα ομότυπο της κλάσης `PrintStream`. Η κλάση `PrintStream` παριστάνει την πρότυπη έξοδο που είναι η οθόνη. Τέλος η μέθοδος `println()` στέλνει ένα **String** χαρακτήρων στην έξοδο.

---

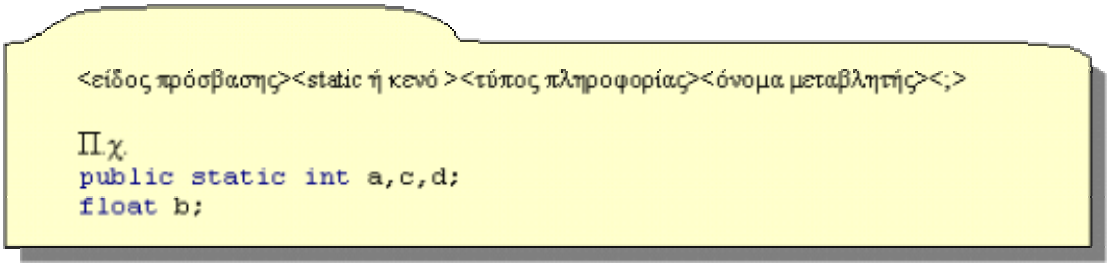
## 5.4 Μεταβλητές κλάσης και ομότυπου

---

Οι μεταβλητές κλάσης δηλώνονται με την δεσμευμένη λέξη **static** την οποία έχουν ως πρόθεμα. Αυτές οι μεταβλητές αποθηκεύονται στις κλάσεις και είναι ίδιες για όλα τα ομότυπα της συγκεκριμένης κλάσης . Δηλαδή όταν ένα ομότυπο αλλάζει την τιμή μιας μεταβλητής αυτή η αλλαγή επιδρά και για τα άλλα ομότυπα της ίδιας κλάσης.

Οι μεταβλητές ομοτύπου δεν χρησιμοποιούν καμία δεσμευμένη λέξη για να δηλωθούν. Κάθε νέο ομότυπο μιας κλάσης παίρνει ένα καινούργιο αντίγραφο των μεταβλητών ομότυπου που ορίζει η κλάση . Το αντικείμενο μιας κλάσης αλλάζει τις μεταβλητές ομότυπου της κλάσης χωρίς να επηρεάζονται οι τιμές των άλλων ομότυπων.

Η πλήρης μορφή της σύνταξης των μεταβλητών είναι:




*Σύνταξη 5.3 – Σχηματισμός μεταβλητής ( πλήρης μορφή)*

Το είδος πρόσβασης αναλύεται στο κεφάλαιο 5.7.

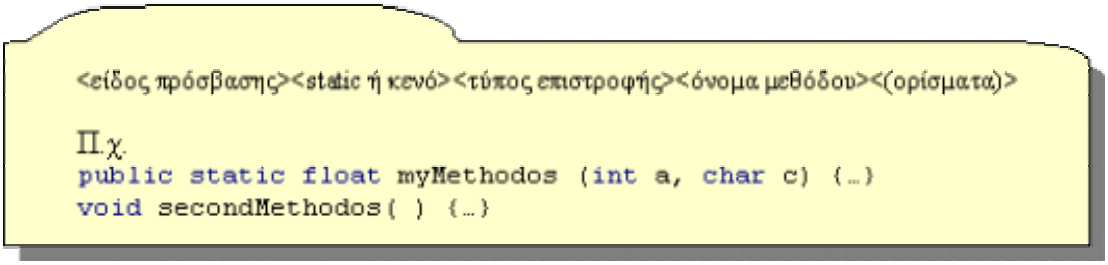
Παρατήρηση: Δεν μπορεί να υπάρχει μεταβλητή κλάσης στο σώμα μιας μεθόδου.

---

## 5.5 Μέθοδοι κλάσης και ομότυπου

---

Γενικά οι μέθοδοι ορίζονται ως εξής: Στην αρχή υπάρχει ο προσδιοριστής πρόσβασης της μεθόδου, ακολούθως ορίζεται αν η μέθοδος είναι κλάσης ή ομοτύπου, μετά υπάρχει ο τύπος επιστροφής της μεθόδου και τέλος προαιρετικά μπορεί η μέθοδος να έχει και ορίσματα μέσα σε παρενθέσεις. Οι προσδιοριστές πρόσβασης μεθόδων και μεταβλητών αναλύονται στο κεφάλαιο 5.7. Ο τύπος επιστροφής μιας μεθόδου μπορεί να είναι οποιοδήποτε τύπος μεταβλητής και όνομα κλάσης. Σε περίπτωση που η μέθοδος δεν επιστρέφει τιμή τότε στην θέση του τύπου επιστροφής μπαίνει η λέξη **void**. Τα ορίσματα είναι προαιρετικά. Συνοπτικά τα παραπάνω είναι:




*Σύνταξη 5.4 – Σχηματισμός μεθόδου*

Μια μέθοδος κλάσης είναι κοινή για όλα τα αντικείμενα που μπορούν να την καλέσουν δηλαδή μία αλλαγή της μεθόδου επηρεάζει όλα τα ομότυπα. Οι μέθοδοι κλάσεων δηλώνονται με την δεσμευμένη λέξη **static** και δεν είναι υποχρεωτικό να κληθούν από ένα ομότυπο της κλάσης τους. Μπορούν να κληθούν και με το όνομα της ίδιας της κλάσης .

Μια μέθοδος ομότυπου είναι ξεχωριστή για κάθε ομότυπο, δηλαδή όταν καλείται η συγκεκριμένη μέθοδος ενός ομότυπου δεν επηρεάζεται η ίδια μέθοδος ενός άλλου ομότυπου. Το παρακάτω παράδειγμα διασαφηνίζει τις παραπάνω έννοιες .

Π.χ.

```
public class Program {
    int a=3; // μεταβλητή ομότυπου
    static int b=4; // μεταβλητή κλάσης

    void first ( ){ // μέθοδος ομότυπου
        System.out.println("Welcome to Java");
    }

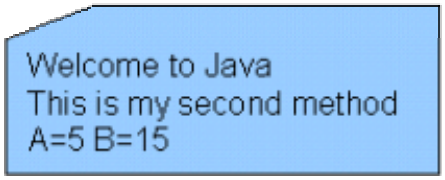
    static void second( ) // μέθοδος κλάσης
    {
        System.out.println("This is my second method");
    }

    public static void main(String args[])
    {
        Program myprogram,myprogramtwo;
        myprogram=new Program( );
        myprogramtwo=new Program( );
        myprogram.a=5;
        myprogram.b=10;
        myprogram.first( );
        Program.second( );
        myprogramtwo.a+=3;
        myprogramtwo.b +=5;
        System.out.println("A="+myprogram.a+" B="+myprogramtwo.b);
    }

}
```

*Παράδειγμα 5.1 – Program.java*

Στο παραπάνω παράδειγμα δημιουργούνται τα αντικείμενα `myprogram` και `myprogramtwo` της κλάσης `program` . Η μεταβλητή `a` είναι μεταβλητή ομότυπου δηλαδή ανεξάρτητη για κάθε ομότυπο της κλάσης ενώ η μεταβλητή `b` είναι μεταβλητή κλάσης δηλαδή η αλλαγή που γίνεται σε αυτή από οποιοδήποτε αντικείμενο συνεχίζει να ισχύει και στα άλλα αντικείμενα. Στην αρχή οι μεταβλητές `a,b` του αντικειμένου `myprogram` λαμβάνουν τις τιμές 5 και 10 αντίστοιχα. Στην συνέχεια καλείται η μέθοδος ομότυπου `first` του αντικείμενου `myprogram` η οποία προβάλει στην έξοδο το μήνυμα ‘`Welcome to Java`‘. Μετά καλείται η μέθοδος κλάσης `second( )` η οποία εμφανίζει το μήνυμα ‘`This is my second method`‘. Η μέθοδος second καλείται με το όνομα της κλάσης. Κάλιστα θα μπορούσε να κληθεί με το όνομα κάποιου ομοτύπου της κλάσης με τα ίδια ακριβώς αποτελέσματα. Στην συνέχεια του προγράμματος στις μεταβλητές `a, b` του αντικειμένου `myprogramtwo` προστίθενται οι τιμές 3 και 5 αντίστοιχα . Το αποτέλεσμα είναι η μεταβλητή `a` να πάρει την τιμή 5 και η μεταβλητή `b` την τιμή 15 διότι το αντικείμενο `myprogram` είχε θέσει στην μεταβλητή κλάσης `b` την τιμή 10 Δηλαδή το αποτέλεσμα που θα δούμε στην έξοδο είναι:




Μια άλλη ενδιαφέρουσα περίπτωση είναι όταν το ίδιο όνομα μεταβλητής ορίζεται και στο σώμα της κλάσης και στο σώμα μιας μεθόδου της κλάσης αυτής. Η μεταβλητή που υπάρχει στην μέθοδο έχει εμβέλεια μόνο μέσα στην μέθοδο .

Στο παρακάτω παράδειγμα ορίζονται δύο μεταβλητές στο σώμα της κλάσης η μεταβλητή `a` είναι μεταβλητή ομότυπου και η μεταβλητή `b` είναι μεταβλητή κλάσης . Στην μέθοδο `methos()` ορίζονται δύο μεταβλητές που έχουν το ίδια ονόματα με τις προηγούμενες δύο μεταβλητές. Οι μεταβλητές `a,b` που είναι έξω από την μέθοδο μπορούν να κληθούν μέσα στην μέθοδο μόνο μέσω κάποιου αντικειμένου. Η μεταβλητή `b` επειδή είναι μεταβλητή κλάσης μπορεί να κληθεί και με το όνομα της κλάσης .

Π.χ.

```
public class Classmethodos{
    int a=23;
    static int b=25;

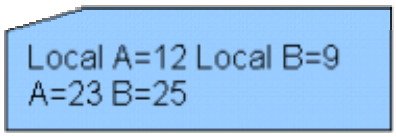
    void methos(){
        int a=12;
        int b=9;
        Classmethodos cl=new Classmethodos();
        System.out.println("Local A="+a+" Local B="+b);
        System.out.println("A="+cl.a+" B="+Classmethodos.b);
    }

    public static void main(String[] args1){
        Classmethodos c=new Classmethodos();
        c.methos();
    }

}
```

*Παράδειγμα 5.2 – Classmethodos.java*

Η έξοδος του προγράμματος είναι:



Εκτός από τις μεταβλητές που ορίζονται μέσα στις μεθόδους και οι μεταβλητές που ορίζονται μέσα σε μπλοκ προτάσεων έχουν εμβέλεια μόνο μέσα σε αυτά.

Π.χ.

```
// Η μεταβλητή i έχει εμβέλεια μόνο μέσα στο μπλοκ της for

for(int i=0;i<10;i++)

{…}
```

Παρατηρήσεις: Τα ορίσματα μιας μεθόδου μπορεί να είναι ένας πίνακας ή ακόμα και ένα αντικείμενο μιας κλάσης.

---

# 5.6 Αλλαγή τύπων

---

Η αλλαγή τύπων περιλαμβάνει τρεις κατηγορίες :

- Μετατροπή τύπου μεταξύ τύπων δεδομένων, δηλαδή **int** σε **float** , **float** σε **double** κ.α
- Μετατροπή από ένα ομότυπο μιας κλάσης σε ένα ομότυπο άλλης .
- Μετατροπή τύπου δεδομένων σε αντικείμενα .

Στην πρώτη κατηγορία γίνεται η μετατροπή ενός τύπου δεδομένων σε ένα άλλο τύπο . Βέβαια υπάρχει και ο τύπος **boolean** στον οποίο δεν μπορεί να γίνει αλλαγή σε άλλον τύπο διότι αυτός ο τύπος μπορεί να πάρει μόνο δύο τιμές **true** και **false**. Η σύνταξη της αλλαγής τύπου είναι :

<(το όνομα του επιθυμητού τύπου δεδομένων)><η τιμή που θέλει αλλαγή>

Π.χ.

```
int a = 5,c;
c=(int)(a/3);
```

*Σύνταξη 5.5 – Αλλαγή τύπων δεδομένων*

Όταν γίνεται μετατροπή τύπου μιας μεταβλητής από τύπο δεδομένων μικρότερης ακρίβειας σε τύπο δεδομένων μεγαλύτερης ακρίβειας δεν υπάρχει απώλεια δεδομένων όπως **int** σε **float**, **int** σε **long**, **int** σε **double**. Στην αντίθετη περίπτωση δηλαδή μετατροπή τύπου δεδομένων μεγαλύτερης ακρίβειας σε μικρότερης ακρίβειας υπάρχει απώλεια δεδομένων όπως **long** σε **float**, **long** σε **double** .

Στην δεύτερη κατηγορία μετατροπών ένα ομότυπο μιας κλάσης αλλάζει σε ομότυπο άλλης κλάσης. Απαραίτητη προϋπόθεση είναι η μία κλάση να είναι δευτερεύουσα της άλλης. Η αλλαγή ενός αντικειμένου δευτερεύουσας κλάσης σε αντικείμενο της υπερκλάσης της δεν έχει νόημα διότι η δευτερεύουσα κλάσης κληρονομεί οτιδήποτε έχει σχέση με την υπερκλάση της. Αντίθετα η αλλαγή αντικειμένου υπερκλάσης σε αντικείμενο μιας δευτερεύουσας κλάσης έχει σαν αποτέλεσμα να κερδίσει το αντικείμενο της υπερκλάσης όλες τις μεταβλητές και τις μεθόδους της δευτερεύουσας κλάσης της. Η σύνταξη της αλλαγής τύπου είναι:

<(όνομα της επιθυμητής κλάσης)><το όνομα του αντικείμενου που θα αλλάξει>

Π.χ.

```
//Η κλάση First είναι υπερκλάση της Second
First  fir=new First( );
Second sec=new Second( );
sec=(Second)fir ;
```

*Σύνταξη 5.6 – Αλλαγή τύπου κλάσης ενός αντικειμένου*

Στην τρίτη κατηγορία τύποι δεδομένων μετατρέπονται σε αντικείμενα και αντίστροφα, τα οποία όμως αντικείμενα πρέπει να αντιστοιχούν σε ένα τύπο δεδομένων. Δηλαδή οι τύποι **int**, **float**, **boolean** μετατρέπονται στα αντικείμενα των κλάσεων `Integer`, `Float` και `Boolean` αντίστοιχα .

Ένας τύπος δεδομένων μετατρέπεται σε αντικείμενο της αντίστοιχης του κλάσης και μπορεί καλώντας την κατάλληλη μέθοδο της κλάσης να μετατραπεί πάλι στον αρχικό τύπο δεδομένων .

Π.χ.

```
Integer akeraios=new Integer(234); /* Ένας ακέραιος (234) μετατρέπετε σε αντικείμενο
της κλάσης Integer */
int a=akeraios.intValue(); /* Η μέθοδος intValue της κλάσης Integer δίνει ένα ακέραιο αριθμό α=234
 από το αντικείμενο akeraios */
```

---

# 5.7 Προσδιοριστές πρόσβασης (Access Specifiers)

---

### 5.7.1 Γενικά

---

Ένα από τα πλεονεκτήματα των κλάσεων είναι ότι μπορούν να προστατεύσουν τις μεθόδους και τις μεταβλητές τους από πρόσβαση αντικειμένων άλλων κλάσεων καθώς και την πρόσβαση των ίδιων στις μεταβλητές τους όταν καλούνται μέσα από άλλες κλάσεις. Η `Java` υποστηρίζει τέσσερα επίπεδα

πρόσβασης για τις μεταβλητές και τις μεθόδους τα οποία είναι **private**, **public**, **protected**, και **package** . Τα τρία πρώτα είδη δηλώνονται σαν πρόθεμα με τις προαναφερθείσες λέξεις στη μεταβλητή ή στη μέθοδο που προσδιορίζουν. Το τέταρτο είδος δεν δηλώνεται στις μεταβλητές και στις μεθόδους, απλώς εννοείται δηλαδή υπάρχει κενό στην δήλωση πρόσβασης. Στον πίνακα 4.1 δηλώνονται συνοπτικά οι άδειες του κάθε προσδιοριστή πρόσβασης.

| Προσδιοριστής | Κλάση | Υποκλάση έξω από το πακέτο της υπερκλάσης της | Κλάσεις που ανήκουν στο ίδιο πακέτο | Κάθε άλλη κλάση εκτός από αυτές που ανήκουν στο ίδιο πακέτο |
|---|---|---|---|---|
| private | ναι | όχι | όχι | όχι |
| protected | ναι | ναι | ναι | όχι |
| public | ναι | ναι | ναι | ναι |
| package | ναι | όχι | ναι | όχι |

*Πίνακας 4.1- Προσδιοριστές πρόσβασης*

Ο παραπάνω πίνακας αναφέρει την πρόσβαση που υπάρχει όταν τα μέλη μιας κλάσης καλούνται από αυτήν την κλάση, στην δεύτερη στήλη μέσα από αυτή την κλάση, στην τρίτη στήλη από μια υποκλάση της έξω από το πακέτο της κλάσης αυτής, στην τέταρτη στήλη μέσα από οποιαδήποτε κλάση που ανήκει στο πακέτο της κλάσης αυτής και στην πέμπτη στήλη οποιαδήποτε κλάση έξω από το πακέτο της κλάσης αυτής. Η λέξη ναι στον πίνακα σημαίνει ότι ο προσδιοριστής πρόσβασης είναι έγκυρος στην συγκεκριμένη περίπτωση.

---

### 5.7.2 Private πρόσβαση

---

Το πιο περιοριστικό επίπεδο πρόσβασης είναι το **private**(ιδιωτική). Τα **private** μέλη είναι προσπελάσιμα μόνο στις κλάσεις στις οποίες έχουν οριστεί . Για να οριστεί ένα **private** μέλος χρησιμοποιείται η δεσμευμένη λέξη **private** ως πρόθεμα. Η κλάση που ακολουθεί ορίζει την **private** μεταβλητή a και την **private** μέθοδο methodos().

Π.χ.

```
class Program {
      // ορισμός της private μεταβλητής
     private int a;

      // ορισμός της private μεθόδου
```

```
    private void methodos( ){
        System.out.println(" This is a private method");
    }

}
```

Στο παραπάνω παράδειγμα μόνο τα αντικείμενα της κλάσης `Program` που βρίσκονται μέσα σε αυτή μπορούν να προσπελάσουν και να επεξεργαστούν τα **private** μέλη της. Για παράδειγμα η κλάση `Prosbash` στην συνέχεια του παραδείγματος δεν μπορεί να προσπελάσει τα **private** μέλη της κλάσης `Program` .

Π.χ.

```
class Prosbash {

    void prosbash_method( ) {
        Program antikeimeno =new Program( );
        antikeimeno.a=10; // λάθος τρόπος πρόσβασης
        antikeimeno.methodos( ); // λάθος τρόπος πρόσβασης
    }

}
```

---

### 5.7.3 Protected πρόσβαση

---

Το επίπεδο πρόσβασης **protected** επιτρέπει στα μέλη μιας κλάσης (εννοείται μέσω της κλάσης αυτής) που έχουν δηλωθεί έτσι (ως protected) να μπορούν να προσπελαθούν από όλες τις δευτερεύουσες κλάσεις της, καθώς και όλες τις κλάσεις που ανήκουν στο ίδιο πακέτο. Για την δήλωση των **protected** μελών χρησιμοποιείται η δεσμευμένη λέξη **protected** ως πρόθεμα. Ακολουθεί ένα παράδειγμα στο οποίο υπάρχουν δυο κλάσεις του πακέτου `mypackage`, οι κλάσεις `First` και `Second` . Η κλάση `First` έχει μια **protected** μέθοδο και μια **protected** μεταβλητή .

Π.χ.

```
package mypackage;

class First{
    protected int a;

    protected void methodfirst( ){
        System.out.println("This is a protected method");
    }

}
```

---

```
package mypackage;

class Second{

    void methodsecond( ){
        First antikeimeno =new First( );
        antikeimeno.a=5; /* πρόσβαση σε protected μεταβλητή του ίδιου πακέτου */
        antikeimeno.methodfirst( ); /* πρόσβαση σε protected μέθοδο του ίδιου πακέτου */
    }

}
```

Επεκτείνοντας το παράδειγμα ορίζεται μια τρίτη κλάση με όνομα `Third` η οποία είναι δευτερεύουσα της κλάσης `First` και η οποία ανήκει στο πακέτο `mysecondpackage`. Όπως φαίνεται στον ακόλουθο κώδικα μπορεί η κλάση `Third` μέσο ενός αντικειμένου της να προσπελάσει τις **protected** μεταβλητή και μέθοδο της κλάσης `First`, διότι είναι δευτερεύουσα της κλάσης . Αντιθέτως το αντικείμενο της κλάσης First που ορίζεται στην κλάση Third δεν μπορεί να προσπελάσει τις **protected** μεταβλητή και μέθοδο της (της κλάσης `First`) διότι οι κλάσεις `First` και `Third` ανήκουν σε διαφορετικά πακέτα .

Π.χ.

```
import mypackage.*;

package mysecondpackage;

class Third extends First {

    void methodthird(First f, Third th)
    {
        f.a=10; // λάθος τρόπος πρόσβασης
        th.a=10; /* το αντικείμενο th κληρονομεί από την υπερκλάση του First την μεταβλητή a */
        f.methodfirst( ); // λάθος τρόπος πρόσβασης
        th.methodfirst( ); /* το αντικείμενο th κληρονομεί από την υπερκλάση του First την μέθοδο
methodfirst( ) */
    }

}
```

---

### 5.7.4 Public πρόσβαση

Ο προσδιοριστής πρόσβασης **public** δίνει την δυνατότητα της προσπέλασης από οποιαδήποτε κλάση των μελών μιας άλλης κλάσης (εννοείται μέσω της κλάσης που έχουν δηλωθεί τα μέλη αυτά). Δηλαδή ακόμα και μια κλάση ενός πακέτου μπορεί να έχει πρόσβαση στα **public** μέλη μιας κλάσης άλλου πακέτου .Τα **public** μέλη μιας κλάσης δηλώνονται με την δεσμευμένη λέξη **public**.

Π.χ.

```
public int x;
```

### 5.7.5 Package πρόσβαση

Αυτό το είδος της πρόσβασης επιτρέπει σε μια κλάση να έχει πρόσβαση στα μέλη μιας άλλης κλάσης του ίδιου πακέτου που έχουν δηλωθεί ως **package**. Δηλαδή μια κλάση που η υπερκλάση της ανήκει σε άλλο πακέτο δεν θα έχει πρόσβαση στα protected μέλη της υπερκλάσης της (εννοείται πρόσβαση μέσω της υπερκλάσης της διότι λόγω κληρονομικότητας με ένα αντικείμενό της η κλάση αυτή θα μπορεί να προσπελάσει τα μέλη της υπερκλάσης της). Καμία δεσμευμένη λέξη δεν χρησιμοποιείται για τον ορισμό των μελών μιας κλάσης ως **package**. Δηλαδή όταν δεν υπάρχει τίποτα ως πρόθεμα στις μεταβλητές και στις μεθόδους σημαίνει ότι ανήκουν στην κατηγορία πρόσβασης **package**. Η διαφορά μεταξύ της protected και της package πρόσβασης είναι ότι οι δευτερεύουσες κλάσεις πάντα έχουν πρόσβαση στα protected μέλη των υπερκλάσεων τους ενώ στα package μέλη έχουν πρόσβαση όταν ανήκουν (οι δευτερεύουσες κλάσεις) στο ίδιο πακέτο με τις υπερκλάσεις τους.

## 5.8 Χρήση μεθόδων

### 5.8.1 Μέθοδοι δημιουργοί(Constructors)

Η Java έχει μια κατηγορία μεθόδων που ονομάζονται μέθοδοι ***δημιουργοί*** (constructors) οι οποίες καλούνται όταν δημιουργείται ένα αντικείμενο της κλάσης στην οποία ανήκει η μέθοδος. Η μέθοδος δημιουργός έχει το ίδιο όνομα με το όνομα της κλάσης στην οποία ανήκει, δεν επιστρέφει κάποια τιμή αλλά μπορεί να δεχτεί ορίσματα. Δεν μπορεί ο προγραμματιστής να καλέσει την μέθοδο δημιουργό άμεσα. Παρακάτω δίνεται ένα παράδειγμα σχετικά με την μέθοδο δημιουργού.



```
public class First {
    int a;

    First(int b){
        a=b;
    }

    void method( ){
        System.out.println(a);
    }

    public static void main(String arg[])
    {
        First fir=new First(10); // Δημιουργία του αντικείμενου fir
        fir.method( ); // Καλείται η μέθοδος ομότυπου method
    }

}
```

*Παράδειγμα 5.3 – First.java*

Η έξοδος του προγράμματος είναι:




Στο παραπάνω παράδειγμα δημιουργείται ένα αντικείμενο με όνομα `fir`. Αυτόματα εκτελείται η μέθοδος δημιουργού και θέτει την μεταβλητή `a` ίση με την τιμή της μεταβλητής `b` δηλαδή με 10 .

---

### 5.8.2 Υπερφόρτωση απλών μεθόδων και δημιουργού

---

Σε μία κλάση υπάρχει η δυνατότητα της χρησιμοποίησης μιας μεθόδου με το ίδιο όνομα περισσότερες από μια φορές, αρκεί κάθε μέθοδος να έχει είτε διαφορετικό αριθμό ορισμάτων είτε τουλάχιστον ένα διαφορετικό τύπο ορισμάτων ή και τα δύο. Η παραπάνω δυνατότητα της `Java` ονομάζεται ***υπερφόρτωση μεθόδων***. Ο μεταγλωττιστής καταλαβαίνει ποια μέθοδο πρέπει να καλέσει κάθε φορά ή από τον διαφορετικό αριθμό των ορισμάτων ή τον διαφορετικό τύπο ορισμάτων που χρησιμοποιούνται σε κάθε μέθοδο. Το ίδιο συμβαίνει με τους κατασκευαστές δηλαδή μία κλάση μπορεί να έχει περισσότερες από μία μέθοδο δημιουργού.

Στο παράδειγμα AppOverLoad υπάρχουν δύο μέθοδοι δημιουργοί, η πρώτη έχει τέσσερα ορίσματα και η δεύτερη έχει τρία. Επιπλέον ορίζονται και δύο μέθοδοι με το ίδιο όνομα (με το όνομα methodos) οι οποίες έχουν διαφορετικό αριθμό ορισμάτων. Σε αυτήν την περίπτωση ο μεταγλωττιστής καταλαβαίνει ποια μέθοδος πρέπει να κληθεί κάθε φορά από τον διαφορετικό αριθμό ορισμάτων.




```
public class AppOverLoad{
    int x1=0;
    int x2=0;
    int x3=0;
    int x4=0;

     // Μέθοδος δημιουργός με 4 ορίσματα
    AppOverLoad(int y1,int y2,int y3,int y4)
    {
        x1=y1;
        x2=y2;
        x3=y3;
        x4=y4;
    }

     // Μέθοδος δημιουργός με 3 ορίσματα
    AppOverLoad(int y1, int y2,int y3)
    {
        x1=y1;
        x2=y2;
        x3=y3;
    }

    void methodos(int y1, int y2){
        x1+=y1+y2+x3;
        System.out.println("X1="+x1+" X2="+x2);
        System.out.println("X3="+x3+" X4="+x4);
    }

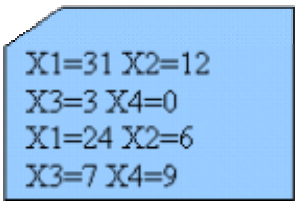
    void methodos(int y1, int y2, int y3){
        x1+=y1*y2+x4;
        x2=y3*x3;
        System.out.println("X1="+x1+" X2="+x2);
        System.out.println("X3="+x3+" X4="+x4);
    }

    public static void main(String[] args){
        AppOverLoad fir,sec;
        fir=new AppOverLoad(1,2,3);  // καλείται η δεύτερη δημιουργός
        sec=new AppOverLoad(4,6,7,9);  // καλείται η πρώτη δημιουργός
        fir.methodos(3,10,4);
        sec.methodos(4,9);
    }

}
```

*Παράδειγμα 5.4 – AppOverLoad.java*

Η έξοδος του προγράμματος στην dos prompt είναι:




---

### 5.8.3 Υπέρβαση μεθόδου

---

Τι ενέργειες κάνει η Java όταν καλείται μια μέθοδος αντικειμένου ;

Η Java στην αρχή αναζητεί την μέθοδο στην κλάση του αντικειμένου . Αν δεν βρει την μέθοδο συνεχίζει ψάχνοντας στις υπερκλάσεις της κλάσης του αντικειμένου. Στο ακόλουθο παράδειγμα ένα αντικείμενο δεν βρίσκει την μέθοδο που καλεί στην κλάση του αλλά στην υπερκλάση του .

Π.χ.

```
class AppFirst{
    int x1=12;

    void methodos( ){
        System.out.println("X1 ="+x1);
    }

}
```

```
------------------
```

```
class AppSecond extends AppFirst{

    public static void main(String[] args){
        AppSecond sec=new AppSecond();
        sec.methodos();
    }

}
```

Η έξοδος του προγράμματος είναι:




Το αντικείμενο `sec` δεν βρήκε την μέθοδο `methodos()` στην κλάση του για αυτό έψαξε στις υπερκλάσεις του και την βρήκε στην υπερκλάση του AppFirst. Αν σε μια κλάση δημιουργηθεί μια μέθοδος που υπάρχει στην υπερκλάση της τότε όταν καλέσει ένα αντικείμενο αυτή την μέθοδο θα εκτελεστεί η μέθοδος της δευτερεύουσας κλάσης. Αυτό ονομάζεται ***υπέρβαση μεθόδου***.

Το παρακάτω παράδειγμα επεξηγεί την υπέρβαση μεθόδου. Σε αυτό ορίζεται μια κλάση AppThird η οποία είναι δευτερεύουσα της AppFirst του προηγουμένου παραδείγματος και έχει την ίδια μέθοδο με αυτήν δηλαδή την μέθοδο `methodos()`. Ένα αντικείμενο της κλάσης `AppThird` καλεί την μέθοδο `methodos()`. Στην περίπτωση αυτή υπάρχει υπερβασή μεθόδου δηλαδή καλείται η μέθοδος `methodos()` της κλάσης `AppThird` αντί της μεθόδου `methdodos()` της υπερκλάσης `AppFirst`. Η μεταβλητή `x1` κληρονομείται στην κλάση `AppThird` από την υπερκλάση της.

Π.χ.

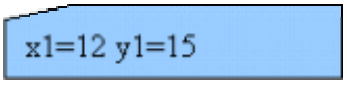
```
class AppThird extends AppFirst{
    int y1=15;

    void methodos(){
        System.out.println("x1="+x1+" y1="+y1);
    }

    public static void main(String[] args){
        AppThird th=new AppThird();
        th.methodos(); //κλήση της μεθόδου της κλάσης AppThird
    }

}
```

Η έξοδος του προγράμματος είναι:




Υπάρχει περίπτωση να μην θέλετε να υπερβείτε την μέθοδο της υπερκλάσης μιας κλάσης αλλά να προσθέσετε και άλλες πληροφορίες σε αυτήν (την μέθοδο). Αυτό γίνεται καλώντας μέσα από το σώμα της μεθόδου που κάνει την υπέρβαση την μέθοδο της υπερκλάσης και επιπλέον μπορείτε να προσθέσετε όποια πληροφορία νομίζετε σε αυτήν. Για να καλέσετε την αρχική μέθοδο χρησιμοποιείτε την λέξη ***super*** την τελεία και τέλος το όνομα της μεθόδου. Το παρακάτω παράδειγμα αποσαφηνίζει τις έννοιες αυτές .

Π.χ.

```
class Superclass{
     int x;

     void apotelesma(int a, int b){
          x+=a*b;
          System.out.println("X="+x);
     }
}



----------------

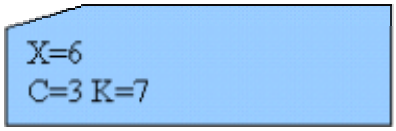
class Secondclass extends Superclass{
     int c=3;
     int k=7;

     void apotelesma(){
          super.apotelesma(10,15);//Καλείται η μέθοδος της υπερκλάσης
          System.out.println("C="+c+" K="+k);
     }

     public static void main(String[] args){
          Secondclass sec=new Secondclass();
          sec.apotelesma();
     }

}
```

Η έξοδος του προγράμματος στην dos prompt είναι:




Στο παραπάνω παράδειγμα η μέθοδος `apotelesma()` καλεί την ίδια μέθοδο από την υπερκλάση της και προσθέτει και μια γραμμή κώδικα στο σώμα της η οποία εκτυπώνει δύο μεταβλητές .

---

### 5.8.4 Υπέρβαση δημιουργών

---

Όταν καλείται ένα αντικείμενο μιας κλάσης τότε η μέθοδος δημιουργού της κλάσης καλείται αυτόματα. Επίσης καλούνται και όλες οι μέθοδοι δημιουργοί στις υπερκλάσεις της κλάσης αυτής . Αν ο προγραμματιστής επιθυμεί να καλέσει την μέθοδο δημιουργού της υπερκλάσης της κλάσης με τα δικά του ορίσματα τότε χρησιμοποιεί την λέξη `super()` βάζοντας στις παρενθέσεις τα επιθυμητά του ορίσματα . Με την λέξη `super` δεν χρησιμοποιείται το όνομα της μεθόδου δημιουργού διότι η μέθοδος δημιουργού δεν μπορεί να κληθεί από ένα αντικείμενο αλλά καλείται από μόνη της. Στο παρακάτω παράδειγμα η μέθοδος δημιουργός `Thirdclass` καλεί την μέθοδο δημιουργό της κλάσης `Myclass` δίνοντας της τα ορίσματα 4,5 . Αν παραληφθεί η εντολή `super(4,5)` το αποτέλεσμα θα είναι διαφορετικό διότι θα κληθεί η προεπιλεγμένη δημιουργός της κλάσης `Myclass` .

Π.χ.

```
class Myclass{
    int p,m;
    Myclass(){
        System.out.println("This method is constructor");
    }

    Myclass(int k,int l){
        System.out.println("This is myclass class");
        p=k;
        m=l;
    }

}
```

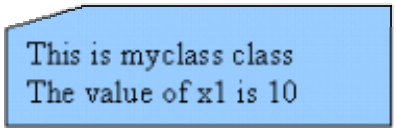
---

```
class Thirdclass extends Myclass{
    String str;
    int x1;
    Thirdclass(String str1,int x){
        super(4,5);
        str=str1;
        x1=x;
    }

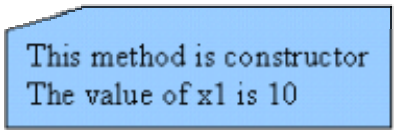
    public static void main(String[] args){
        Thirdclass th=new Thirdclass("The value of x1 is",10);
        System.out.println(th.str+th.x1);
    }

}
```

Η έξοδος του προγράμματος είναι:




Η έξοδος του προγράμματος χωρίς την εντολή `super(4,5)` θα είναι:




Παρατηρήσεις: Η εντολή `super(…)` πρέπει να είναι υποχρεωτικά η πρώτη εντολή στο σώμα της μεθόδου δημιουργού που την καλεί .

---

### 5.8.5 Η λέξη this

---

Η λέξη ***this*** αναφέρεται στο τρέχον αντικείμενο κάθε κλάσης και μπαίνει στην θέση του όνοματος του αντικειμένου. Μπορεί να χρησιμοποιηθεί οπουδήποτε καλείται ένα αντικείμενο δηλαδή στο προσδιορισμό μεταβλητής και μεθόδου, σαν όρισμα και σαν αντικείμενο επιστροφής από μια μέθοδο. Στο ακόλουθο παράδειγμα στην μέθοδο δημιουργού για να προσδιοριστούν τα ονόματα των μεταβλητών επειδή είναι τα ίδια (των μεταβλητών a, b) χρησιμοποιείται η λέξη **this**. Σε αυτή την μέθοδο τα ονόματα των μεταβλητών χωρίς την λέξη **this** στο σώμα της είναι της μεθόδου δημιουργού ενώ αυτά με την λέξη **this** ανήκουν στο κάθε αντικείμενο που καλείται. Στην μέθοδο `methodos()` καλείται η μέθοδος `methos()` με την λέξη **this** . Η λέξη **this** μπορεί να μην χρησιμοποιηθεί σε αυτήν την περίπτωση διότι και με την λέξη **this** και χωρίς αυτή το αποτέλεσμα θα είναι το ίδιο. Αυτό συμβαίνει γιατί η μέθοδος `methos()` είναι μέθοδος αντικείμενου και ήδη αναφέρεται στο τρέχον αντικείμενο. Το ίδιο συμβαίνει και με την μεταβλητή `b` που ορίζεται στην μέθοδο `methos()`. Σε αυτή την μέθοδο αφού δεν ορίζεται στο σώμα της άλλη μεταβλητή με το όνομα `b` η λέξη **this** μπορεί να παραληφθεί διότι είναι μεταβλητή αντικειμένου δηλαδή αναφέρεται στο κάθε τρέχον αντικείμενο.




```
public class AppThis{
    int a,b;
    AppThis(int a, int b){
        this.a=a;
        this.b=b;
    }
```

```
    void methodos(){
        this.methos();
    }

    void methos(){
        System.out.println("A="+a+" B="+this.b);
    }

    public static void main(String[] args1){
        AppThis cl=new AppThis(4,5);
        cl.methodos();
    }

}
```

*Παράδειγμα 5.5 – AppThis.java*

Η έξοδος του προγράμματος είναι:




---

## 5.9 Τροποποιητής final

---

Ο τροποποιητής **final** χρησιμοποιείται με μεταβλητές, μεθόδους και κλάσεις.

Όταν χρησιμοποιείται με μεταβλητές σημαίνει ότι αυτές έχουν σταθερή τιμή η οποία δεν αλλάζει.

Π.χ.

```
final int a=6;

static final char ch;

public static final int v=10;
```

Οι μέθοδοι που δηλώνονται ως **final** (ονομάζονται και τελικές μέθοδοι – αντίστοιχα υπάρχουν τελικές μεταβλητές και τελικές κλάσεις) έχουν το πλεονέκτημα ότι προστατεύονται από τυχόν υπερβάσεις. Δηλαδή μια δευτερεύουσα κλάση της κλάσης που περιέχει την **final** μέθοδο δεν μπορεί να περιέχει την ίδια μέθοδο με την υπερκλάση της. Ο λόγος για τον οποίο αξίζει να κάνετε μια μέθοδο τελική είναι για να εκτελείται αποδοτικότερα. Όταν ο διερμηνευτής **java** εκτελεί μια μέθοδο, πρώτα ελέγχει

την τρέχουσα κλάση για αυτήν την μέθοδο μετά την υπερκλάση της, στην συνέχεια την υπερκλάση της υπερκλάσης κ.ο.κ. Για να αποφευχθεί αυτή η διαδικασία που επιβραδύνει την ταχύτητα, χρησιμοποιείται ο τροποποιητής **final** (δηλαδή όταν βρεθεί μέθοδος ορισμένη ως **final** ο μεταγλωττιστής δεν θα ψάξει για αυτήν στις υπερκλάσεις της κλάσης που ανήκει η μέθοδος).

Π.χ.

```
final void methodos(){

// σώμα μεθόδου

}
```

Οι τελικές κλάσεις δεν έχουν δευτερεύουσες κλάσεις. Επίσης οι μέθοδοι που περιλαμβάνουν είναι και αυτές τελικές χωρίς να χρειάζεται να ορισθούν με τον τροποποιητή **final**.

Π.χ.

```
public final MyClass{

// σώμα κλάσης

}
```

---

# 5.10 Αφηρημένες (abstract) κλάσεις και μέθοδοι

---

Μερικές φορές μια κλάση που ορίζεται μπορεί να περιέχει αφηρημένες έννοιες οι οποίες δεν μπορούν να αρχικοποιηθούν. Για παράδειγμα το φαγητό στον πραγματικό κόσμο. Ποτέ δεν έχετε δει αντικείμενο (ομότυπο) από το φαγητό αλλά έχετε δει αντικείμενα από καρότο, μήλο, σοκολάτα τα οποία ανήκουν στο είδος των φαγητών. Δηλαδή τα φαγητά αντιπροσωπεύουν την αφηρημένη ιδέα των πραγμάτων που μπορούν να φαγωθούν και δεν υπάρχει αντικείμενό τους.

Αντίστοιχα στον αντικειμενοστρεφή προγραμματισμό υπάρχουν μοντέλα από αφηρημένες έννοιες χωρίς την δυνατότητα να μπορούν να αρχικοποιηθούν. Αυτές τα μοντέλα είναι οι ***αφηρημένες κλάσεις*** δηλαδή προσδιορίζουν ιδιότητες και συμπεριφορές που μπορούν να χρησιμοποιηθούν από δευτερεύουσες κλάσεις τους. Για παράδειγμα η κλάση `Number` στο πακέτο `java.lang`, που αντιπροσωπεύει την αφηρημένη έννοια των αριθμών δεν μπορεί να αρχικοποιηθεί αλλά χρησιμοποιείται από τις δευτερεύουσες κλάσεις της, όπως είναι οι κλάσεις Integer, Float.

Για την δήλωση μιας κλάσης ως αφηρημένης χρησιμοποιείται ο τροποποιητής ***abstract*** στην δηλωσή της.

Π.χ.

```
public abstract class AbstractClass{

// σώμα αφηρημένης κλάσης

}
```

Οι αφηρημένες κλάσεις μπορούν να περιέχουν οτιδήποτε περιέχει μια κανονική κλάση και επιπλέον αφηρημένες μεθόδους. Οι αφηρημένοι μέθοδοι είναι δηλώσεις μεθόδων που δεν έχουν σώμα (δεν έχουν υλοποίηση). Αυτές οι μέθοδοι υλοποιούνται σε δευτερεύουσες κλάσεις της αφηρημένης κλάσης.

Όπως φαίνεται στην παρακάτω δήλωση η αφηρημένη μέθοδος δεν περιέχει σώμα εντολών.

Π.χ.

```
abstract void astractMethodos();
```

---

# 5.11 Διασύνδεση (interface)

---

Όπως έγινε αναφορά στο κεφάλαιο 2.3 η Java υποστηρίζει την μονή κληρονομικότητα των κλάσεων της δηλαδή μια κλάση μπορεί να έχει μόνο μια υπερκλάση (στην αμέσως προηγούμενη βαθμίδα πάνω από αυτή). Η μονή κληρονομικότητα είναι περιοριστική επειδή δεν αφήνει μια κλάση να μπορεί να κληρονομεί στην ίδια βαθμίδα παραπάνω από μια κλάσει. Η λύση στο πρόβλημα αυτό είναι η χρησιμοποίηση των διασυνδέσεων.

Η ***διασύνδεση*** είναι μια συλλογή από ορισμούς αφηρημένων μεθόδων (δηλαδή μέθοδοι χωρίς υλοποίηση) που μπορούν να περιέχουν και σταθερές δηλώσεις. Δηλαδή πιο απλά η διασύνδεση είναι αφηρημένη συμπεριφορά που μπορεί να προστεθεί σε μια κλάση ώστε να της προσφέρει ότι δεν της παρέχεται από τις υπερκλάσεις της (εννοείται υπερκλάσεις σε όλες τις βαθμίδες).

---

### 5.11.1 Δημιουργία διασυνδέσεων

---

Οι διασυνδέσεις ορίζονται όμοια με τις κλάσεις, αντί όμως της λέξης **class** στις διασυνδέσεις υπάρχει η λέξη ***interface***. Έτσι η δήλωση μιας διασύνδεσης είναι:

Π.χ.

```
public interface AppInterface{

// σώμα διασύνδεσης

}
```

Οι διασυνδέσεις σώζονται ως πηγαία αρχεία με κατάληξη `.java` και μεταγλωττίζονται όπως και οι κλάσεις από το εργαλείο **javac**. Οι μέθοδοι που ορίζονται σε μια διασύνδεση είναι αφηρημένες (abstract) και δημόσιες (public). Αν δεν τοποθετηθούν αυτοί οι δύο τροποποιητές σε μια μέθοδο τότε εννοούνται. Φυσικά δεν μπορεί μια μέθοδος να δηλωθεί με τους τροποποιητές **private** ή **protected**. Και μεταβλητές μπορούν να δηλωθούν σε μια διασύνδεση αρκεί να συνοδεύονται από τους τροποποιητές **public**, **static**, **final**. Αυτοί (οι τροποποιητές) κάνουν μια μεταβλητή σταθερή. Αν δεν ορίζονται οι τροποποιητές **public**, **static**, **final** σε μια μεταβλητή εννοούνται.

Π.χ.

```
public interface AppInterface{
    public static final int x;
    int y; // εννοούνται οι τροποποιητές public, static, final
    public abstract void methodos();
    void secondmethodos(); // εννοούνται οι τροποποιητές public και abstract
}
```

Ότι ισχύει για τους προσδιοριστές πρόσβασης των κλάσεων (που αναλύονται στο κεφάλαιο 5.13) ισχύει και για τους προσδιοριστές πρόσβασης των διασυνδέσεων. Σε περίπτωση που μια διασύνδεση ορίζεται χωρίς τον τροποποιητή **public** τότε στις μεθόδους της δεν εννοούνται οι τροποποιητές **public**, **abstract** αν δεν δηλώνονται, όπως και στις σταθερές μεταβλητές ο τροποποιητής **public**. Δηλαδή όταν μια διασύνδεση δεν ορίζεται ως **public** τότε οι μέθοδοι και οι μεταβλητές της που δεν ορίζονται ως **public**, μπορούν να χρησιμοποιηθούν μόνο από κλάσεις και διασυνδέσεις στο ίδιο πακέτο. Οι διασυνδέσεις μπορούν να ανήκουν σε κάποιο πακέτο και δηλώνονται σε αυτό όπως και οι κλάσεις (με την δήλωση package). Αναλυτική αναφορά για τα πακέτα υπάρχει στο κεφάλαιο 5.12.

---

### 5.11.2 Χρήση διασυνδέσεων

---

Για την χρησιμοποίηση μιας διασύνδεσης από μια κλάση περιλαμβάνεται η δεσμευμένη λέξη **implements** στο ορισμό της κλάσης.

Π.χ.

```
interface AppIface{

// σώμα διασύνδεσης

}
```

---

```
public class Application extends Applet implements AppIface{

// σώμα κλάσης

}
```

Στο παραπάνω παράδειγμα η κλάση `Application` επεκτείνει την συμπεριφορά που κληρονομεί από την κλάση `Applet` υλοποιώντας την συμπεριφορά της διασύνδεσης `AppIface`. Επειδή οι διασυνδέσεις περιέχουν αφηρημένες μεθόδους (δηλαδή μεθόδους χωρίς υλοποίηση) οι μέθοδοι αυτοί πρέπει να υλοποιούνται στις κλάσεις που τις περιλαμβάνουν. Σημείο προσοχής είναι ότι αν περιλαμβάνετε μια διασύνδεση στο πρόγραμμα σας πρέπει να υλοποιήσετε όλες τις μεθόδους της. Αυτή είναι και μια διαφορά με τις αφηρημένες κλάσεις στις οποίες μπορούν να επιλεγούν οι μέθοδοι που θα υλοποιηθούν.

Μπορείτε να περιλαμβάνετε σε μια κλάση περισσότερες από μια διασυνδέσεις οι οποίες χωρίζονται μεταξύ τους με κόμμα. Στο παρακάτω παράδειγμα η κλάση `Application` υλοποιεί τις διασυνδέσεις `First`, `Second` και `Third`.

Π.χ.

```
public class Application implements First, Second, Third{

// σώμα κλάσης

}
```

Σε περίπτωση που μια κλάση υλοποιεί μια διασύνδεση, οι δευτερεύουσες κλάσεις της κληρονομούν τις μεθόδους της διασύνδεσης. Δηλαδή αν μια κλάση έχει μια υπερκλάση που υλοποιεί μια διασύνδεση τότε δεν χρειάζεται να περιλαμβάνει στον ορισμό της την λέξη **implements** για να υλοποιήσει την διασύνδεση.

Στο ακόλουθο παράδειγμα ορίζεται η διασύνδεση `Arithmetic` που περιέχει δύο μεθόδους μια για πρόσθεση και μια για αφαίρεση. Η κλάση `AppInterface` υλοποιεί την διασύνδεση `Arithmetic` και φυσικά και τις μεθόδους της. Η διασύνδεση `Arithmetic` μόνο θα μεταγλωττιστεί ενώ η κλάση `AppInterface` θα μεταγλωττιστεί και θα διερμηνευτεί (εκτελεστεί).




```
public interface Arithmetic{
    public abstract void add(int x, int y);
    public abstract void sub(int x, int y);
}
```

*Παράδειγμα 5.6 – Διασύνδεση Aritmetic.java*




```
public class AppInterface implements Arithmetic{
    public AppInterface(){}

    public void add(int a, int b){
        int c=a+b;
        System.out.println("Addition="+c);
    }

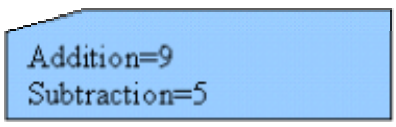
    public void sub(int a, int b){
        int c=a-b;
        System.out.println("Subtraction="+c);
    }

    public static void main(String[] args){
        AppInterface app=new AppInterface();
        app.add(4,5);
        app.sub(10,5);
    }

}
```

*Παράδειγμα 5.6 – AppInterface.java*

Η έξοδος του προγράμματος είναι:



# 5.12 Πακέτα (packages)

Ένα ***πακέτο*** (package) είναι μια συλλογή σχετικών κλάσεων και διασυνδέσεων που παρέχει διαχείριση ξεχωριστού ονόματος και προστασία πρόσβασης. Η συλλογή σχετικών μεταξύ τους κλάσεων σε ένα πακέτο χρησιμεύει στην πιο εύκολη εύρεση και χρησιμοποίηση των κλάσεων, βοηθάει στην αποφυγή διένεξης ονομάτων των κλάσεων και στην προστασία πρόσβασης.

Τα πακέτα κάνουν πιο εύκολη την εύρεση μιας συγκεκριμένης κλάσης διότι ο προγραμματιστής θα την αναζητήσει σε ένα πακέτο που οι κλάσεις του σχετίζονται με την λειτουργία της (συγκεκριμένης κλάσης). Με τα πακέτα αποφεύγονται οι διενέξεις ονομάτων. Όσο μεγαλώνει ο αριθμός των κλάσεων της `Java` τόσο αυξάνει η πιθανότητα δύο κλάσεις να έχουν το ίδιο όνομα. Όπως αναλύεται παρακάτω, μπορούν να μεταγλωττιστούν σε ένα πρόγραμμα περισσότερες από μια κλάσεις με το ίδιο όνομα αρκεί να ανήκουν σε διαφορετικό πακέτο.

Ένα πακέτο μπορεί να περιέχει και άλλα πακέτα τα οποία προσδιορίζουν ένα πιο συγκεκριμένο επίπεδο ομαδοποίησης κλάσεων. Π.χ. `java.awt.Color`. Το ανώτερο επίπεδο είναι το `java`. Το επόμενο είναι το `awt` και τέλος το όνομα της κλάσης είναι `Color`. Το ανώτερο επιπέδο (java) περιλαμβάνει και άλλα πακέτα όπως τα `net`, `util`,κ.α.

### 5.12.1 Χρήση πακέτων

Μόνο οι δημόσιες κλάσεις ενός πακέτου είναι προσπελάσιμες έξω από αυτό. Υπάρχουν δύο τρόποι σύνταξης κλάσεων που ανήκουν σε ένα πακέτο.

Ο πρώτος είναι με την αναφορά του πλήρους ονόματος του πακέτου ή των πακέτων και της κλάσης. Στην αρχή μπαίνουν τα ονόματα των πακέτων και στο τέλος το όνομα των κλάσεων τα οποία χωρίζονται με τελείες.

Π.χ.

```
java.awt.Color color=new java.awt.Color();
```

Με το δεύτερο τρόπο μπορούν να εισαχθούν στο πρόγραμμα μεμονωμένες κλάσεις ή ολόκληρα πακέτα κλάσεων. Αφού εισαχθεί μια κλάση ή ένα πακέτο κλάσεων σε ένα πρόγραμμα η αναφορά σε αυτές γίνεται μόνο με το όνομα τους. Η εισαγωγή κλάσεων με τον δεύτερο τρόπο γίνεται με την βοήθεια της δήλωσης **import**. Στην περίπτωση που εισάγεται μια μεμονωμένη κλάση η σύνταξη είναι η δήλωση **import** και στην συνέχεια το όνομα του πακέτου (ή των πακέτων) και της κλάσης.

Π.χ.

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

Στην περίπτωση εισαγωγής ολόκληρου πακέτου κλάσεων η σύνταξη είναι η δήλωση **import** το όνομα του πακέτου ( ή των πακέτων) και στο τέλος ένας αστερίσκος (*).

Π.χ.

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

Η δήλωση **import** σε ένα πρόγραμμα τοποθετείται πριν τον ορισμό της κλάσης. Όταν εισάγεται ένα πακέτο κλάσεων σε ένα πρόγραμμα αυτό δεν σημαίνει ότι το πρόγραμμα γίνεται μεγαλύτερο και βραδύτερο. Μόνο οι κλάσεις που καλούνται μέσα στο σώμα της κλάσης του προγράμματος φορτώνονται.

Σε όλα τα παραδείγματα του παρόντος εγχειριδίου χρησιμοποιείται ο δεύτερος τρόπος σύνταξης.

Έστω ότι υπάρχουν δύο πακέτα με ονόματα kostasclasses, nikosclasses τα οποία περιλαμβάνουν μια κλάση με το ίδιο όνομα (η κλάση ονομάζεται SameClass). Αν στο ίδιο πρόγραμμα κληθεί η κλάση SameClass και από τα δύο πακέτα με την παρακάτω σύνταξη τότε ο μεταγλωττιστής θα εμφανίσει μια ένδειξη λάθους λόγο της διένεξης των ονομάτων που συμβαίνει.

Π.χ.

```
import kostasclasses.SameClass;
import nikosclasses.SameClass;

public class …{
    …
    SameClass same=new SameClass(); // εσφαλμένη χρήση
    …
}
```

Η λύση σε αυτήν την περίπτωση είναι να χρησιμοποιηθεί ο πρώτος τρόπος σύνταξης δηλαδή κάθε κλάση να εμφανιστεί με το πλήρες όνομα πακέτου – κλάσης.

Π.χ.

```
kostasclasses.SameClass c=new kostasclasses.SameClass();
```

```
nikosclasses.SameClass n=new nikosclasses.SameClass();
```

Υπάρχει και μια κατηγορία κλάσεων που είναι αυτόματα διαθέσιμες σε όλα τα προγράμματα. Αυτές είναι οι κλάσεις που ανήκουν στο πακέτο `java.lang`. Δηλαδή δεν χρειάζεται η δήλωση **import** `java.lang.*`.

---

### 5.12.2 Δημιουργία πακέτων

---

Πρώτο βήμα στην δημιουργία πακέτων είναι το όνομα του πακέτου. Μπορείτε να χρησιμοποιήσετε οποιοδήποτε όνομα αλλά η σύμβαση που συνίσταται είναι η διεύθυνση της ιστοσελίδας σας αντιστραμμένη. Δηλαδή αν η ιστοσελίδα σας είναι `myhomepage.com` τότε το όνομα του πακέτου πρέπει να είναι `com.myhomepage`.

Δεύτερο βήμα στην δημιουργία πακέτων είναι η δημιουργία φακέλων (καταλόγων) στον σκληρό δίσκο που το όνομα τους να είναι ίδιο με το πακέτο. Δηλαδή αν το όνομα του πακέτου είναι `com.myhomepage` τότε πρέπει να δημιουργηθεί ένας φάκελος με όνομα `com` μέσα στον οποίο θα υπάρχει ένας φάκελος με όνομα `myhomepage`. Μέσα στον φάκελο `myhomepage` πρέπει να τοποθετηθούν οι κλάσεις που ανήκουν στο συγκεκριμένο πακέτο (…com/myhomepage/Example.class).

Τρίτο και τελευταίο βήμα είναι η προσθήκη της δήλωσης **package** μαζί με το όνομα του πακέτου στην κλάση που επιθυμείται να ανήκει σε αυτό (το πακέτο). Η δήλωση **package** πρέπει να είναι η πρώτη γραμμή κώδικα στο πηγαίο αρχείο. Δηλαδή μια κλάση που ανήκει στο πακέτο `com.myhomepage` πρέπει να έχει μια γραμμή κώδικα σαν την παρακάτω.

Π.χ.

```
package com.myhomepage;
```

Στο παρακάτω παράδειγμα η κλάση `AppPackage` ανήκει στο πακέτο `myname`. Η κλάση `AppPackage` μπορεί να μεταγλωττιστεί (με το εργαλείο javac) αλλά όχι να διερμηνευτεί αφού δεν ορίζεται στο σώμα της η μέθοδος `main(...)`. Η κλάση `AppPackage` πρέπει να βρίσκεται σε κατάλογο με όνομα `myname`.

Η κλάση `AppExample` έχει την μέθοδο `methodos()`, στο σώμα της οποίας δημιουργείται το αντικείμενο `app` της κλάσης `AppPackage`. Στην συνέχεια το αντικείμενο `app` καλεί την μέθοδο `message()` η οποία εκτυπώνει ένα μήνυμα.



```
package myname;

public class AppPackage{
    public AppPackage(){};

    public void message(){
        System.out.println("Hello, I am contained in package 'myname'");
    }

}
```

*Παράδειγμα 5.7 – AppPackage.java*




```
import myname.AppPackage;

public class AppExample{
    public AppExample(){}

     /* Δημιουργείται ένα αντικείμενο της κλάση AppPackage το οποίο καλεί την μέθοδο message */
    public void methodos(){
        AppPackage app=new AppPackage();
        app.message();
    }

    public static void main(String[] args){
        AppExample example=new AppExample();
        example.methodos();
    }

}
```

*Παράδειγμα 5.7 – AppExample.java*

Η έξοδος του προγράμματος είναι:

```
Hello, I am contained in package 'myname'
```

Κλείνοντας εάν μια κλάση η διασύνδεση δεν δηλώνετε σε κανένα πακέτο καταλήγει σε ένα προεπιλεγμένο πακέτο το οποίο δεν έχει όνομα.

---

### 5.12.3 Παραμετροποίηση του καταλόγου των κλάσεων (CLASSPATH)

---

Ένα πρόγραμμα για να χρησιμοποιήσει κάποιες κλάσεις πρέπει πρώτα να τις βρει στο σύστημα αρχείων (σκληρό δίσκο) διαφορετικά θα εμφανιστεί ένα μήνυμα ότι οι κλάσεις δεν υπάρχουν. Για την εύρεση των κλάσεων η `Java` ψάχνει στα ονόματα των πακέτων και τους καταλόγους που υπάρχουν στην μεταβλητή `CLASSPATH`. Εάν κατά την εγκατάσταση του `JDK` δεν θέσατε κάποιους καταλόγους στην μεταβλητή `CLASSPATH` (συνιστάται ο μη ορισμός καταλόγων στην classpath) τότε ο μεταφραστής και ο μεταγλωττιστής ψάχνουν για κλάσεις στον προεπιλεγμένο κατάλογο `lib` του `JDK` (όπου υπάρχουν αρκετές έτοιμες κλάσεις που χρησιμοποιούνται σε αυτό το εγχειρίδιο) και στον τρέχοντα κατάλογο. Για να μπορεί ένα πρόγραμμα να χρησιμοποιεί μια κλάση που ανήκει σε πακέτο πρέπει η μεταβλητή `CLASSPATH` να περιέχει τον κατάλογο του πρώτου επιπέδου του πακέτου. Δηλαδή έστω το πακέτο `com.myhomepage` που βρίσκεται στον κατάλογο `d:\java` (d:\java\com\myhomepage) και περιέχει την κλάση `Example`. Για να μπορεί ένα πρόγραμμα να χρησιμοποιήσει την κλάση `Example`, εκτός από την πρόταση **import** `com.myhomepage.Example` που πρέπει να περιέχει, πρέπει η μεταβλητή `CLASSPATH` να περιέχει τον κατάλογο `d:\java\`.

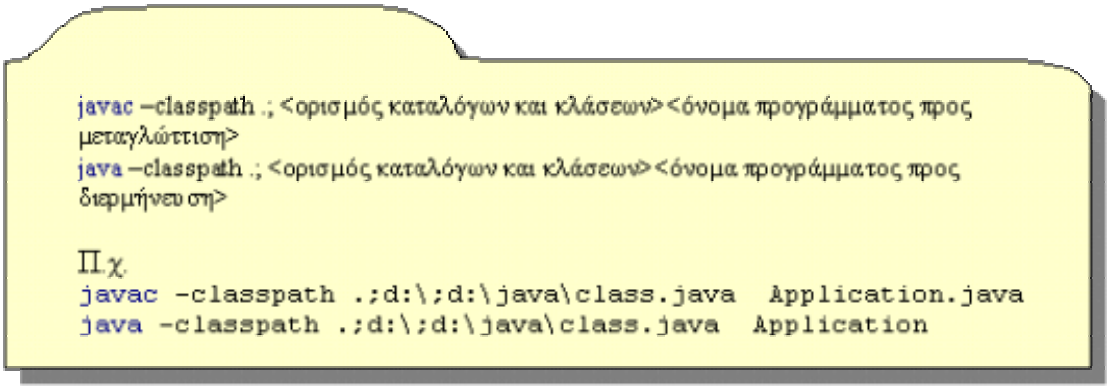
Η μεταβλητή `CLASSPATH` ορίζεται με δύο τρόπους. Ο πρώτος είναι με την σύνταξη “set classpath=ονόματα καταλόγων ή κλάσεων (συμπιεσμένων)” που χωρίζονται με ελληνικό ερωτηματικό.

Π.χ.

```
set classpath=d:\;d:\java\classes.jar
```

Ο τρόπος αυτός δεν συνιστάται διότι υπερβαίνει την τρέχουσα `CLASSPATH` και αν κληθεί στην dos-prompt αυτή παραμένει ενεργή μέχρι να τερματιστεί (η τρέχουσα dos-prompt). Επιπλέον εάν οριστεί η `CLASSPATH` να φορτώνεται κατά την εκκίνηση του λειτουργικού, το περιεχόμενο των καταλόγων της θα είναι διαθέσιμο σε κάθε dos-prompt υπερβαίνοντας φυσικά την προεπιλεγμένη τιμή της `CLASSPATH`.

Ο δεύτερος τρόπος για τον ορισμό τιμών στην `CLASSPATH` είναι με το όρισμα `-classpath` του μεταγλωττιστή **javac** και του διερμηνευτή **java**. Το όρισμα `-classpath` υπερβαίνει τον τρέχοντα κατάλογο κλάσεων μόνο όμως την στιγμή που καλείται από τον μεταγλωττιστή και τον μετάφραση. Η σύνταξη του στο λειτουργικό σύστημα `Windows` είναι:



Όλα τα παραπάνω ισχύουν για το λειτουργικό `Win98`. Στην περίπτωση των `Win2000` υπάρχει μια εξαίρεση δηλαδή κατά την μεταγλώττιση ενός προγράμματος στην μεταβλητή `CLASSPATH` δεν λαμβάνεται υπόψη ο τρέχων κατάλογος αν δεν δηλωθεί.

---

# 5.13 Προσδιοριστές πρόσβασης κλάσεων

---

Προσδιοριστές πρόσβασης εκτός από τις μεθόδους και τις μεταβλητές υπάρχουν και στις κλάσεις. Αυτοί είναι δύο, ο προεπιλεγμένος ο οποίος είναι μη εμφανίσιμος και ο **public**. Ο ορισμός μιας κλάσης μπορεί να έχει μια από τις δύο παρακάτω μορφές:

Π.χ.

```
public class AppClass{

// σώμα κλάσης

}
```

…

```
class AppClass{

// σώμα κλάσης

}
```

Ο προεπιλεγμένος προσδιοριστής πρόσβασης κάνει μια κλάση διαθέσιμη σε όλες τις κλάσεις στο ίδιο πακέτο, αλλά σε καμία έξω από αυτό το πακέτο. Ο προσδιοριστής πρόσβασης **public** επιτρέπει σε μια κλάση να εισαχθεί σε οποιαδήποτε κλάση ακόμα και αν δεν ανήκει στο ίδιο πακέτο.

---

# 5.14 Προβλήματα

---

**Άσκηση 1**. Να γραφεί ένα πρόγραμμα το οποίο θα παίρνει ένα η περισσότερα ορίσματα σαν ονόματα και θα τα εκτυπώνει μαζί με έναν χαιρετισμό. Σε περίπτωση που δεν δίνεται όρισμα θα εμφανίζεται κάποιο άλλο μήνυμα .

**Λύση :** Ορίζεται η κλάση `Hello_user` που ελέγχει μέσω των εντολών **if** αν ο χρήστης έδωσε κάποιο όρισμα στην γραμμή εντολών. Αν υπάρχουν ορίσματα στη γραμμή εντολών, εκτυπώνεται ένας χαιρετισμός μαζί με τα ονόματα των ορισμάτων αλλιώς εκτυπώνεται το μήνυμα '`Hello everybody` '. Η ακέραια μεταβλητή length δείχνει τον αριθμό των στοιχείων ενός πίνακα/




```
public class Hello_user{

    public static void main(String[] args){
          /* Αν το μήκος του πίνακα είναι μεγαλύτερο του μηδενός τότε υπάρχουν ορίσματα
          επομένως τυπώνεται το Hello+όνοματα */
        if(args.length>0)
        for(int i=0;i<args.length;i++)
        System.out.println ("Hello "+args[i]);
          /* Αν το μήκος του πίνακα είναι ίσο με το μηδέν τότε δεν υπάρχουν ορίσματα */
        if(args.length<=0)
            System.out.println("Hello everybody");
    }

}
```

*Παράδειγμα 5.8 – Hello_user.java*

- source code

Αν η είσοδος είναι '`java Hello_user`' η έξοδος είναι:




Αν η είσοδος είναι '`java Hello_user Maria Petros`' η έξοδος είναι:




**Άσκηση 2**. Να γραφεί πρόγραμμα που θα ταξινομεί 15 τυχαίους αριθμούς από τον μικρότερο στον μεγαλύτερο.

**Λύση :** Αρχικά δημιουργούνται 15 τυχαίοι αριθμοί που αποθηκεύονται στον πίνακα `a[]`. Οι αριθμοί αυτοί δημιουργούνται από την μέθοδο `nextInt()` της κλάσης `Random`. Η μέθοδος `nextInt()` παράγει τυχαίους ακεραίους αριθμούς. Μια πρωτοεμφανιζόμενη μέθοδος είναι η `System.out.print( )` η οποία είναι όμοια με την `System.out.println( )` με την διαφορά ότι παραμένει στην ίδια σειρά εκτύπωσης μετά το πέρας της εργασίας της. Ο ακέραιος `temp` χρησιμοποιείται για αποθήκευση προσωρινών τιμών. Οι συγκρίσεις γίνονται μεταξύ ενός στοιχείου του πίνακα και του επόμενου. Αν ένα στοιχείο είναι μικρότερο του προηγουμένου του τότε αλλάζει θέση με αυτό. Οι συγκρίσεις διακόπτονται όταν δεν υπάρχει κανένα στοιχείο μικρότερο του επόμενου άρα η σύγκριση `if(a[i]>a[i+1])` είναι πάντα λάθος, η μεταβλητή `bool` παραμένει **false** και επομένως ο βρόχος `while(bool)` τερματίζεται.




```
import java.util.Random;


public class AppSort{

    public static void main(String[] args){
        boolean bool=true;
        int temp;
        int[] a=new int[15];
        Random rand=new Random();
        System.out.println("The random numbers are ");
        for(int i=0;i<15;i++){
            a[i]=rand.nextInt();
            System.out.print(" "+a[i]);
        }
        System.out.println("The sorted list is:");
        while(bool) {
            bool=false;
            for(int i=0;i<14;i++){
                if(a[i]>a[i+1]){
                    temp=a[i];
                    a[i]=a[i+1];
                    a[i+1]=temp;
                    bool=true;
                }
            }
        }
        System.out.println ("");
        for(int i=0;i<14;i++)
            System.out.println ("i="+a[i]);
    }

}
```

*Παράδειγμα 5.8 – AppSort.java*

- source code




# ΤΑ ΒΑΣΙΚΑ ΤΩΝ APPLETS

## 6.1 Εισαγωγή

Ενώ οι εφαρμογές στην `Java` εκτελούνται από έναν διερμηνευτή, οι ***μικροεφαρμογές*** (applets) εκτελούνται από έναν φυλλομετρητή (browser – εξεταστή) του διαδικτύου όπως τον Netscape Navigator και τον Microsoft Internet Explorer. Επίσης οι μικροεφαρμογές μπορούν να εκτελεστούν από το εργαλείο της `Java` **appletviewer**.

Το πρώτο στάδιο για την εκτέλεση μικροεφαρμογών και εφαρμογών είναι όμοιο. Το πηγαίο αρχείο της μικροεφαρμογής μεταγλωττίζεται με τον **javac** `compiler`. Το αρχείο `bytecode` που έχει δημιουργηθεί πρέπει να συμπεριληφθεί σε μια ιστοσελίδα για να εκτελεσθεί. Επειδή οι μικροεφαρμογές τρέχουν σε κάποιον φυλλομετρητή έχουν όλα τα πλεονεκτήματα της δομής αυτού όπως γραφικά, γεγονότα κ.ο.κ.

## 6.2 Ασφάλεια

Μια από τις κυριότερες προτεραιότητες του περιβάλλον της `Java` είναι να κάνει τον εξεταστή ιστοσελίδων (browser) να αισθάνεται ασφαλής εκτελώντας μικροεφαρμογές. Για την επιτυχία αυτού του σκοπού έχουν επιβληθεί ορισμένοι περιορισμοί στα `applets` οι οποίοι είναι :

- Οι μικροεφαρμογές δεν μπορούν να γράψουν ή να διαβάσουν στο σύστημα αρχείων του χρήστη.
- Οι μικροεφαρμογές δεν μπορούν να εκτελέσουν κανένα πρόγραμμα που βρίσκεται στο περιβάλλον του χρήστη.
- Οι μικροεφαρμογές μπορούν να επικοινωνήσουν μόνο με την περιοχή του Internet που εξυπηρετεί την συγκεκριμένη ιστοσελίδα που τις περιέχει (τις μικροεφαρμογές) .

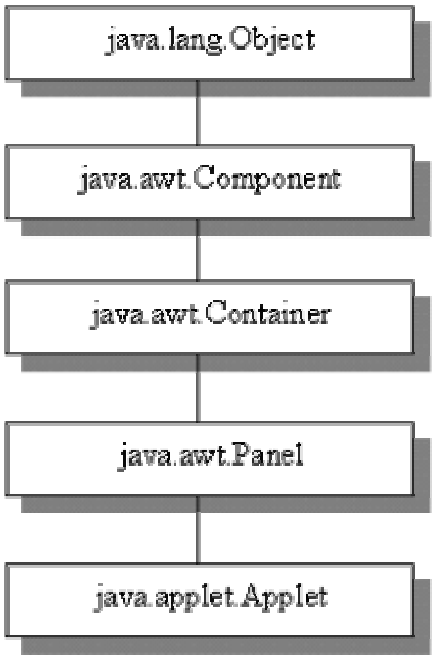
# 6.3 Μέθοδοι των μικροεφαρμογών

---

Όλες οι προηγούμενες εφαρμογές σε `Java` περιείχαν την μέθοδο `main()` η οποία εκτελούνταν πρώτη και καθόριζε την συμπεριφορά του προγράμματος. Αντιθέτως στις μικροεφαρμογές δεν υπάρχει η μέθοδος `main()` αλλά στην θέση της υπάρχουν μέθοδοι που καλούνται σε διαφορετικά σημεία κατά την εκτέλεση του `applet`. Για αυτές τις μεθόδους θα υπάρχει αναφορά στην συνέχεια του κεφαλαίου.

Οι μικροεφαρμογές είναι δευτερεύουσες κλάσεις της κλάσης `Applet` η οποία ανήκει στο πακέτο `java.applet` άρα κάθε μικροεφαρμογή όταν ορίζεται πρέπει να περιλαμβάνει την δήλωση "**extends Applet** (ή extends java.applet.Applet)".

Το επόμενο σχήμα δείχνει την ιεραρχία της κλάσης `Applet` δηλαδή από ποίες κλάσεις κληρονομεί μεθόδους και μεταβλητές .




*Σχήμα 6.1 – Ιεραρχία της κλάσης Applet*

Στην κορυφή της ιεραρχίας βρίσκεται η κλάση `Object` του πακέτου `java.lang` και στην τελευταία βαθμίδα η `Applet` του πακέτου `java.applet`.

Από την κλάση `Component` του πακέτου `java.awt` κληρονομεί η `Applet` μεθόδους σχεδίασης και χειρισμού γεγονότων. Οι μέθοδοι σχεδίασης αναφέρονται σε ό,τι έχει σχέση με την σχεδίαση της μικροεφαρμογής στην οθόνη, δηλαδή προβολή εικόνων, εμφάνιση στοιχείων διασύνδεσης όπως κουμπιών, αναδιπλούμενων λιστών κ.α. Ο χειρισμός γεγονότων αναφέρεται στην αντίληψη και επεξεργασία ενεργειών εισόδου από τον χρήστη, όπως το πάτημα των κουμπιών, ή το πάτημα πλήκτρων από το πληκτρολόγιο. Επιπλέον, αναφέρεται και σε πιο αφηρημένα γεγονότα όπως την αποθήκευση αρχείων.

Η κλάση `Container` του πακέτου `java.awt` λειτουργεί σαν υποδοχέας των στοιχείων διασύνδεσης του χρήστη όπως κουμπιά ταμπέλες κ.α. δηλαδή τα συστατικά τοποθετούνται στον υποδοχέα ο οποίος τοποθετείτε στο παράθυρο της μικροεφαρμογής .

Η σύνταξη ενός applet είναι:

```
import java.applet.*;

public class MyApplet extends Applet{
        // σώμα μικροεφαρμογής
}
```

Με την πρόταση **import** `java.applet.*` γίνονται προσπελάσιμες στην μικροεφαρμογή όλες οι δημόσιες κλάσεις του πακέτου java.applet στο οποίο ανήκει και η κλάση `Applet`. Το πρόγραμμα θα ήταν σωστό και με την δήλωση **import** `java.applet.Applet` κατά την οποία μόνο η κλάση `Applet` γίνεται προσπελάσιμη.

Οι μέθοδοι που ανήκουν στην κλάση `Applet` αν κληθούν από μόνες τους δεν εκτελούν καμιά λειτουργία. Για να παρουσιάσει μια μικροεφαρμογή κάποια συμπεριφορά, πρέπει να γίνει υπέρβαση στις μεθόδους της κλάσης `Applet`.

Οι πιο σημαντικές μέθοδοι της κλάσης `Applet` είναι η ***αρχικοποίηση*** (init()), ***εκκίνηση*** (start()), ***διακοπή*** (stop()), ***καταστροφή*** (destroy()) και ***εκτύπωση*** (paint(Graphics g)). Με υπέρβαση αυτών των μεθόδων στην δευτερεύουσα κλάση της `Applet`, παράγει η μικροεφαρμογή κάποια συμπεριφορά. Η σύνταξη των παραπάνω μεθόδων είναι :

Μέθοδος αρχικοποίησης

```
public void init(){

        // σώμα μεθόδου

}
```

Όπως φαίνεται και από το όνομα της, η μέθοδος αυτή καλείται όταν φορτώνεται η μικροεφαρμογή για να δώσει αρχικές τιμές σε μεταβλητές, να δημιουργήσει αντικείμενα, να φορτώσει εικόνες, κ.ο.κ.

Μέθοδος εκκίνησης

```
public void start(){

        // σώμα μεθόδου

}
```

Η παραπάνω μέθοδος χρησιμοποιείται για την εκκίνηση της μικροεφαρμογής. Η εκκίνηση συμβαίνει μετά την αρχικοποίηση και η διαφορά τους είναι ότι η αρχικοποίηση γίνεται μια φορά, ενώ η εκκίνηση μπορεί να γίνει περισσότερες από μια.

Μέθοδος διακοπής

```
public void stop(){

    // σώμα μεθόδου

}
```

Η μέθοδος διακοπής καλείται όταν η μικροεφαρμογή εγκαταλείπει την σελίδα που την περιέχει. Επιπλέον, ο χρήστης μπορεί να καλέσει απευθείας την μέθοδο `stop()` για να σταματήσει την εκτέλεση της μικροεφαρμογής. Η μέθοδος διακοπής μπορεί να θεωρηθεί λογικό συμπλήρωμα της μεθόδου εκκίνησης.

Μέθοδος καταστροφής

```
public void destroy(){

    // σώμα μεθόδου

}
```

Η μέθοδος καταστροφής καλείται λίγο πριν τερματιστεί η μικροεφαρμογή, με σκοπό να απελευθερώσει τους πόρους του συστήματος που είχαν διατεθεί σε αυτή. Η μέθοδος καταστροφής συνήθως δεν υπερβαίνεται εκτός από ειδικές περιπτώσεις .

Μέθοδος εκτύπωσης

```
public void paint(Graphics g){

    // σώμα μεθόδου

}
```

Η μέθοδος εκτύπωσης χρησιμοποιείται για την σχεδίαση των περιεχομένων του κεντρικού πλαισίου της μικροεφαρμογής καθώς και του γραφικού περιβάλλοντος διεπαφής χρήστη (Graphical User's Interface), όποτε αυτό χρειάζεται. Δηλαδή, καλείται κατά την αρχικοποίηση της μικροεφαρμογής, αλλά και κάθε φορά που πραγματοποιείται μετακίνηση (συμπεριλαμβανομένου της απόκρυψης και της επαναφοράς) του κεντρικού πλαισίου. Η `paint()` είναι η μόνη από της πέντε μεθόδους που έχει όρισμα, το οποίο είναι ένα ομότυπο της κλάσης `Graphics`. Το αντικείμενο αυτό είναι υπεύθυνο για ότι εμφανίζεται στην οθόνη. Η κλάση `Graphics` υπάρχει στο πακέτο java.awt.Graphics επομένως

στην αρχή κάθε πηγαίου κώδικα μικροεφαρμογής θα πρέπει να υπάρχει η έκφραση **import** `java.awt.Graphics`.

Όπως αναφέρθηκε οι μικροεφαρμογές τρέχουν και μέσα από έναν φυλλομετρητή. Η γλώσσα που καταλαβαίνει ο φυλλομετρητής είναι η γλώσσα μορφοποίησης υπερκειμένου γνωστή ως `HTML`. Στο επόμενο κεφάλαιο δίνονται ορισμένα βασικά στοιχεία της γλώσσας `HTML`.

---

# 6.4 Βασικές σημάνσεις της HTML

---

Οι ιστοσελίδες σχεδιάζονται με την ***HTML*** (Hypertext Markup Language) γλώσσα. Η βάση της `html` είναι τα ***tags*** (σημάνσεις). Τα πιο σημαντικά `tags` της `html` είναι τα παρακάτω.

| Σημάνσεις(tags) | Ερμηνεία |
|---|---|
| <HTML> περιλαμβάνονται όλες οι σημάνσεις του html εγγράφου</HTML> | Μέσα στις <html>,</html> σημάνσεις περικλείονται όλες οι υπόλοιπες σημάνσεις. |
| <HEAD> | Η σήμανση head εισάγεται μετά την html σήμανση και εισάγει την ενότητα της επικεφαλίδας |
| <TITLE>τίτλος</TITLE> | Οι σημάνσεις <title> και </title> περικλείουν τον τίτλο του html εγγράφου. |
| </HEAD> | Ακολουθεί την σήμανση </title> και κλείνει την ενότητα της επικεφαλίδας. |
| <BODY>κύριο σώμα του html εγγράφου</BODY> | Ακολουθεί την σήμανση </HEAD> και περιέχει ότι εμφανίζεται στον εξεταστή. |
| <APPLET CODE=”Καθορίζει το όνομα της κλάσης της μικροεφαρμογής” WIDTH=Πλάτος του παράθυρου της μικροεφαρμογής σε pixels HEIGHT=΄Υψος του παράθυρου της μικροεφαρμογής σε pixels </APPLET> | Οι applet σημάνσεις εισάγονται μέσα στις σημάνσεις body και ορίζουν μια μικροεφαρμογή. |
| <PARAM NAME=Ορίζεται το όνομα της παραμέτρου VALUE=”Ορίζεται η τιμή της παραμέτρου” > | Πρόκειται για μία μονή σήμανση η οποία χρησιμοποιείται για πέρασμα παραμέτρων στις μικροεφαρμογές |

*Πίνακας 6.1-Βασικές σημάνσεις της html*

Η γλώσσα υπερκειμένου (html) δεν κάνει διάκριση μεταξύ κεφαλαίων και πεζών χαρακτήρων. Ένα έγγραφο `html` μπορεί να συνταχθεί με οποιοδήποτε `editor` και πρέπει να σωθεί με την κατάληξη `.html`. Η σειρά με την οποία τοποθετούνται οι παραπάνω σημάνσεις πρέπει να τηρείται. Μόνο μέσα στα `tags body` μπορούν να υπάρχουν σημάνσεις με οποιαδήποτε σειρά. Όλες οι παραπάνω σημάνσεις εισάγονται στο παρακάτω παράδειγμα.

Π.χ.

```
<HTML> // αρχή του αρχείου HTML
<HEAD> // αρχή της επικεφαλίδας
<TITLE>MY FIRST APPLET</TITLE> // τίτλος MY FIRST APPLET
</HEAD> // τέλος επικεφαλίδας
<BODY> // αρχή του σώματος
<APPLET CODE="MyProgram.class" WIDTH=300 HEIGHT=300>
// φορτώνεται η μικροεφαρμογή σε ένα παράθυρο 300 επί 300 pixels
<PARAM NAME=... VALUE='...'> // η σήμανση param είναι προαιρετική
</APPLET>
// υπόλοιπα tags
</BODY> // τέλος του σώματος
</HTML> // τέλος του αρχείου HTML
```

Το παραπάνω πρόγραμμα μπορεί να σωθεί με οποιοδήποτε όνομα αρκεί να έχει την κατάληξη `.html`. Έστω ότι σώζεται με το όνομα `First_html.html`. Για να εκτελεσθεί το τελευταίο αρχείο, θα πρέπει να ανοίξει με την βοήθεια ενός φυλλομετρητή, ή του εργαλείου **appletviewer**.

---

## 6.5 Η πρώτη μικροεφαρμογή

---

Σε αυτό το κεφάλαιο δημιουργείται η πρώτη μικροεφαρμογή σε Java η οποία θα εμφανίζει στον φυλλομετρητή το απλό μήνυμα 'Hello user'. Παρακάτω συντάσσονται τα δύο αρχεία., το πηγαίο αρχείο της μικροεφαρμογής και το έγγραφο html μέσα από το οποίο προβάλλεται η μικροεφαρμογή. Για να τρέξει η μικροεφαρμογή πρέπει πρώτα να μεταγλωττιστεί το πηγαίο αρχείο. Στην συνέχεια φορτώνεται το αρχείο HelloApplet.html από έναν φυλλομετρητή, ή τρέχει με την βοήθεια του εργαλείου **appletviewer**. Προϋπόθεση για να τρέξει η μικροεφαρμογή είναι ότι τα αρχεία HelloApplet.class και HelloApplet.html θα βρίσκονται στον ίδιο κατάλογο. Στο αρχείο HelloApplet.html, αναφέρεται το μέγεθος του παραθύρου της μικροεφαρμογής οι διαστάσεις του οποίου είναι 400x200 pixels.




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

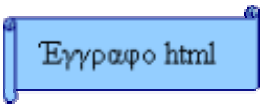
public class HelloApplet extends Applet{

    public void paint(Graphics g){
        Font f=new Font("TimesRoman",Font.BOLD,40);
        g.setFont (f);
        g.drawString("Hello user",50,50);
    }

}
```

*Παράδειγμα 6.1-HelloApplet.java*

- source code




```
<HTML>
<HEAD>
<TITLE>MY FIRST APPLET</TITLE>
</HEAD>
<BODY>
<APPLET CODE="HelloApplet.class" WIDTH=400 HEIGHT=200>
</APPLET>
</BODY>
</HTML>
```

*helloapplet.html*

Για να τυπωθεί το παραπάνω αποτέλεσμα χρησιμοποιείται η μέθοδος `drawstring(...)` της κλάσης `Graphics`. Η `drawstring(...)` δέχεται τρία ορίσματα που χωρίζονται με κόμμα. Το πρώτο όρισμα είναι τύπου **String** και αποτελεί το μήνυμα που θα εμφανιστεί στο παράθυρο της μικροεφαρμογής. Οι άλλες δύο παράμετροι είναι ακέραιου τύπου και τοποθετούν το κείμενο στην επιθυμητή θέση. Δηλαδή θεωρείται ότι στο παράθυρο υπάρχουν δύο άξονες, ο οριζόντιος (x άξονας) και ο κατακόρυφος (y άξονας), με φορά από δεξιά προς τα αριστερά και από πάνω προς τα κάτω αντίστοιχα . Το σημείο εκκίνησης είναι η επάνω αριστερή γωνία που έχει συντεταγμένες (0, 0). Η μονάδα μέτρησης των συντεταγμένων είναι τα pixels. Στο συγκεκριμένο παράδειγμα η πάνω αριστερή γωνία του κειμένου θα τοποθετηθεί στο σημείο με συντεταγμένες (50,50) pixels. Επιπλέον, μπορεί να οριστεί η κατάλληλη γραμματοσειρά με το αντικείμενο της κλάσης `Font`. Η κλάση `Font` δέχεται τρία ορίσματα τα οποία θέτουν τον τύπο, το στυλ και το μέγεθος της χρησιμοποιούμενης γραμματοσειράς. Τέλος, η μέθοδος `setFont(...)` της κλάσης `Graphics` θέτει σε λειτουργία τις ρυθμίσεις της κλάσης `Font`.

# 6.6 Πέρασμα παραμέτρων σε μικροεφαρμογές

Για το πέρασμα παραμέτρων σε μικροεφαρμογές χρησιμοποιείται η σήμανση `Param` της γλώσσας υπερκειμένου που έχει δύο ιδιότητες, την `name` και την `value`. Η παράμετρος name προσδιορίζει το όνομα της ιδιότητας, ενώ η `value` την τιμή της. Μια μικροεφαρμογή μπορεί να έχει περισσότερες από μια σημάνσεις `Param`. Οι παράμετροι περνούν στην μικροεφαρμογή με την μέθοδο `getParmeter(String str)` η οποία έχει ως όρισμα ένα **String** που αντιστοιχεί στην παράμετρο `name` της σήμανσης και επιστρέφει ένα **String** που αντιστοιχεί στην παράμετρο `value`.

Στο παράδειγμα 6.2 η τιμή `value` περνάει στην μικροεφαρμογή.




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

public class GetParameterClass extends Applet {
    String timh;

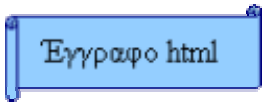
    public void init() {
        timh= getParameter("name_parameter");
        timh="Hello "+timh;
    }

    public void paint(Graphics g) {
        Font f=new Font("TimesRoman",Font.BOLD,40);
        g.setFont (f);
        g.drawString(timh, 50, 50);
    }

}
```

*Παράδειγμα 6.2-GetParameterClass.java*

- source code




```
<HTML>
<HEAD>
<TITLE>GetParameter APPLET</TITLE>
</HEAD>
<BODY>
<APPLET CODE="GetParameterClass.class" WIDTH=400 HEIGHT=100>
```

```
<PARAM name=name_parameter value='students'>
</APPLET>
</BODY>
</HTML>
```

*getparameterclass.html*

---

# 6.7 Κλάση Graphics

---

Οι πιο σημαντικές μέθοδοι της κλάσης `Graphics` παρουσιάζονται συνοπτικά ακολούθως .

- `drawLine(int x1, int y1, int x2, int y2)`: Η μέθοδος αυτή σχεδιάζει μια γραμμή από το σημείο με συντεταγμένες `x1`, `y1` μέχρι το σημείο με συντεταγμένες `x2`, `y2`
- `drawRect(int x, int y, int width, int height)`: Η συγκεκριμένη μέθοδος σχεδιάζει ένα ορθογώνιο που αρχίζει από το σημείο με συντεταγμένες `x`, `y` και έχει πλάτος (από αριστερά προς τα δεξιά) `width` και ύψος (από πάνω προς τα κάτω) `height`.
- `drawRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)`: Η μέθοδος αυτή σχεδιάζει ορθογώνια με στρογγυλεμένες γωνίες. Τα τέσσερα πρώτα ορίσματα είναι ίδια με την μέθοδο `drawRect(...)`. Τα αλλά δύο προσδιορίζουν το πλάτος και το ύψος που θα έχουν οι τέσσερις στρογγυλεμένες γωνίες.
- `fillRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)`: Είναι ίδια με την `drawRoundRect(...)` με την διαφορά ότι γεμίζει τα ορθογώνια.
- `draw3DRect(int x, int y, int width, int height, boolean raised)`: Είναι ίδια με την `drawRect(...)` αλλά με τονισμένο το περίγραμμα του ορθογωνίου. Το πέμπτο όρισμα προσδιορίζει αν το ορθογώνιο θα εμφανίζεται ανυψωμένο από την επιφάνεια (προσκήνιο) ή θα βυθίζεται σε αυτήν.
- `drawPolygon(int[] xPoints,int[] yPoints,int nPoints)`: Η συγκεκριμένη μέθοδος σχεδιάζει ένα πολύγωνο που αποτελείται από `nPoints` σημεία που οι συντεταγμένες τους είναι στους πίνακες `xPoints[]` και `yPoints[]`. Η `drawPolygon(...)` δημιουργεί ένα κλειστό πολύγωνο δηλαδή δεν χρειάζεται η πρώτη συντεταγμένη να είναι ίδια με την πρώτη.
- `drawPolygon(Polygon p)`: Η μέθοδος αυτή σχεδιάζει κατευθείαν ένα αντικείμενο της κλάσης Polygon. Η κλάση Polygon σχεδιάζει πολύγωνα και η μέθοδος constructor της είναι `Polygon(int[] xpoints, int[] ypoints, int npoints)` της οποίας τα ορίσματα είναι ίδια με της προηγούμενης μεθόδου.
- `drawOval(int x, int y, int width, int height)`: Η συγκεκριμένη μέθοδος χρησιμοποιείται για το σχεδιασμό ελλείψεων και κύκλων με κέντρο τις συντεταγμένες `x`, `y` και πλάτος, ύψος `width` και `height` αντίστοιχα . Τα ορίσματα `width` και `height` έχουν την ίδια χρήση με τα ορίσματα `arcWidth` και

`arcHeight` της συνάρτησης drawRoundRect(...). Στην περίπτωση που οι `width` και `height` είναι ίσες σχηματίζεται κύκλος.

- `fillOval(int x, int y, int width, int height)`: Είναι όμοια με την προηγούμενη με την διαφορά ότι γεμίζει το σχήμα.
- `drawArc(int x, int y, int width, int height, int startAngle, int arcAngle)`: Με την μέθοδο αυτή σχεδιάζονται τόξα. Το όρισμα `startAngle` είναι η γωνία από που αρχίζει το τόξο και η `arcAngle` είναι ο αριθμός των μοιρών του τόξου που προστίθενται. Οι μοίρες στο τόξο προστίθενται κατά την ανθωρολογιακή φορά .
- `fillArc(int x, int y, int width, int height, int startAngle, int arcAngle)`: Είναι ίδια με την παραπάνω και επιπλέον γεμίζει το σχήμα.
- `copyArea(int x, int y, int width, int height, int dx, int dy)`: Η μέθοδος αυτή αντιγράφει μια ορθογώνια περιοχή του παραθύρου της μικροεφαρμογής σε μια άλλη περιοχή του παραθύρου. Τα πρώτα δύο ορίσματα είναι οι συντεταγμένες τις αρχής του ορθογωνίου και τα επόμενα δύο το πλάτος και το ύψος του. Τα ορίσματα `dx` και `dy` είναι η μετακίνηση σε pixels του ορθογωνίου από την αρχική του θέση στον άξονα `x`, `y` αντίστοιχα.
- `clearRect(int x, int y, int width, int height)`: Η μέθοδος αυτή καθαρίζει μια ορθογώνια περιοχή δηλαδή την γεμίζει με το χρώμα του παρασκηνίου (background).

---

## 6.8 Γραμματοσειρές

---

Η υπεύθυνη κλάση για το όνομα το στυλ και το μέγεθος μιας γραμματοσειράς είναι η `Font` που ανήκει στο πακέτο `java.awt`. Τα παραπάνω χαρακτηριστικά προσδιορίζονται στην μέθοδο αρχικών συνθηκών (constructor) της κλάσης δηλαδή `Font (όνομα γραμματοσειράς, στυλ γραμματοσειράς, μέγεθος γραμματοσειράς)`. Το όνομα της γραμματοσειράς αντιστοιχεί σε ένα από τα διαθέσιμα είδη γραμματοσειράς που είναι εγκατεστημένα στον υπολογιστή του χρήστη, όπως `Arial`, `Courier`, `Times New Roman` κ.α. Το στυλ της γραμματοσειράς μπορεί να είναι από τα `Font.PLAIN`, **`Font.BOLD`**, *`Font.ITALIC`* ή συνδυασμός αυτών. Τέλος το μέγεθος της γραμματοσειράς προσδιορίζεται από ένα ακέραιο αριθμό. Για να ενεργοποιηθεί η επιθυμητή γραμματοσειρά, δεν αρκεί να προσδιοριστεί, αλλά πρέπει να περάσει ως όρισμα στη μέθοδο `setFont(...)` της κλάσης. Δηλαδή μια γραμματοσειρά γίνεται ενεργή με τις εντολές:

```
Font font=new Font("Arial",Font.BOLD+Font.ITALIC,14);

g.setFont(font); // Το g είναι αντικείμενο της κλάσης Graphics
```

---

# 6.9 Χρώμα

---

Μια από τις πιο σημαντικές κλάσης που είναι υπεύθυνη για τον χρωματισμό μιας εφαρμογής `Java` είναι η κλάση `Color` η οποία ανήκει στο πακέτο `java.awt`. Η `Color` χρησιμοποιεί το χρωματικό σύστημα `RGB`. Τα αρχικά αυτού του ονόματος προέρχονται από τα τρία χρώματα R(red), G(green), B(blue). Η Color αναμιγνύει αυτά τα χρώματα για να παράγει το επιθυμητό. Τα χρώματα αυτά έχουν κλίμακα από το 0 που σημαίνει πλήρη απουσία τους μέχρι το 255 που αντιστοιχεί σε πλήρες γέμισμα. Εναλλακτικά, υπάρχει η δυνατότητα να καθορίζονται τα ποσοστά των χρωμάτων με τιμές **float** όπου με `0F` συμβολίζεται η μηδενική ποσότητας χρώματος και με `1F` η μέγιστη. Δεν αρκεί να δημιουργηθεί ένα αντικείμενο `Color`, αλλά πρέπει να καταστεί ενεργό. Αυτό γίνεται με την μέθοδο `setColor(...)` της κλάσης `Graphics` η οποία έχει σαν όρισμα ένα αντικείμενο της κλάσης `Color` όπως φαίνεται στο ακόλουθο παράδειγμα.

Π.χ.

```
Color color=new Color(255,0,0); /* Οι τιμές του δημιουργού της Color αντιστοιχούν σε 255 κόκκινο, 0 πράσινο, 0 μπλε άρα το τρέχον χρώμα είναι το κόκκινο */

g.setColor(color); // Η g είναι αντικείμενο της κλάσης Graphics
```

Η συμπληρωματική μέθοδος της `setColor(...)` είναι η `getColor()` η οποία δεν έχει ορίσματα και επιστρέφει το τρέχον χρώμα.

Οι μέθοδοι που θέτουν ή παίρνουν το χρώμα του προσκηνίου ή του παρασκηνίου κληρονομούνται από μια υπερκλάση της κλάσης `Applet`. Επομένως όλες οι μικροεφαρμογές που δημιουργούνται τις κληρονομούν. Πιο συγκεκριμένα, οι μέθοδοι αυτές ορίζονται ακολούθως :

- `getBackground()`: Δεν έχει ορίσματα και επιστρέφει το χρώμα του παρασκηνίου.
- `getForeground()`: Δεν έχει ορίσματα και επιστρέφει το χρώμα του προσκηνίου.
- `setBackground(αντικείμενο Color)`: Η μέθοδος αυτή θέτει το χρώμα του παρασκηνίου σύμφωνα με το όρισμα της που είναι ένα αντικείμενο της κλάσης `Color`.
- `setForeground(αντικείμενο Color )`: Ίδια με την προηγούμενη με την διαφορά ότι αναφέρεται στο προσκήνιο.

Το παρακάτω παράδειγμα διασαφηνίζει τις προαναφερθείσες μεθόδους .




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

public class ShapeColor extends Applet{
    Font newfont;
```

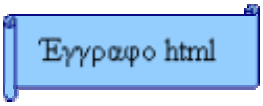
```
    public void init(){
     /* Αρχικοποιήση της μικροεφαρμογής, επιλέγεται το χρώμα φόντου, το χρώμα προσκηνίου και η γραμματοσειρά.
     Αυτή η διαδικασία μπορεί να γίνει και στο σώμα της paint( ). */
         setBackground(new Color(0.3f,0.5f,1f));
         newfont=new Font("Arial",Font.ITALIC,18);
         setForeground(new Color(0,0,255));
    }

    public void paint(Graphics g){
     /* Οι ενέργειες της Graphics είναι: Γίνεται ενεργό το χρώμα φόντου, τυπώνεται ένα μήνυμα,
     σχεδιάζεται μια γραμμή, μετά ένα ορθογώνιο, ακολουθεί ένα ορθογώνιο με στρογγυλεμένες
     γωνίες και ένα τόξο με συμπληρωμένο το εσωτερικό του. Στη συνέχεια, ορίζεται νέο χρώμα
     προσκηνίου και νέα γραμματοσειρά. Τέλος σχεδιάζεται ένας κύκλος και τυπώνεται ένα μήνυμα. */
         g.setFont(newfont);
         g.drawString("Καλωσηρθατε στον μαγικο κοσμο της Java",20,20);
         g.drawLine(20,23,370,23);
         g.drawRect(20,40,100,100);
         g.fillRoundRect(200,40,100,150,45,74);
         g.fillArc(350 ,50,150,150, 30, 224);
         g.setColor(new Color(0,255,0));
         newfont=new Font("Times new Roman",Font.BOLD,20);
         g.setFont(newfont);
         g.drawOval(450,100,80,80);
         g.drawString("Αυτη ειναι η τελευταια σειρα του προγραμματος",25,180);
    }
}
```

*Παράδειγμα 6.3-ShapeColor.java*

- source code

Στο παράδειγμα 6.3 συνδυάζονται οι πιο πολλές από τις προαναφερθείσες μεθόδους. Ακολούθως, συντάσσεται το έγγραφο `html` που θα εκτελεστεί από τον **appletviewer** ή από έναν εξεταστή. Αν εκτελεστεί η μικροεφαρμογή μόνο μέσω του εργαλείου **appletviewer** και όχι με την βοήθεια ενός εξεταστή, η μόνη απαραίτητη σήμανση είναι η `<applet>` αφού οι άλλες αγνοούνται από το εργαλείο **appletviewer** της `Java`.




```
<APPLET CODE=”ShapeColor.class” WIDTH=550 HEIGHT=200>
</APPLET>
```

*shapecolor.html*

Οι μέθοδοι `setBackground()` και `setForeground()` χρησιμοποιούνται χωρίς να προτάσσεται ένα αντικείμενο ή μια κλάση πριν από αυτές. Αυτό συμβαίνει επειδή η κλάση στην οποία ανήκουν αυτές οι μέθοδοι είναι υπερκλάση της κλάσης `Applet`. Επομένως, είναι προσπελάσιμες από την κλάση `ShapeColor` του παραδείγματος 6.3. Η αλλαγή του χρώματος και της γραμματοσειράς ισχύει για όσα σχήματα σχεδιάζονται μετά την αλλαγή αυτών των χαρακτηριστικών. Αντίθετα η αλλαγή χρώματος παρασκηνίου, προσκηνίου επηρεάζει οτιδήποτε σχεδιάζεται ή έχει σχεδιαστεί. Δηλαδή, αν στο παράδειγμα 6.3 η τελευταία εντολή στην μέθοδο `paint(...)` αντιστοιχεί σε αλλαγή χρώματος

προσκηνίου, τότε ό,τι έχει σχεδιαστεί θα αποκτήσει το συγκεκριμένο χρώμα. Μια άλλη λεπτομέρεια του συγκεκριμένου παραδείγματος είναι ότι η σύνταξη του, `g.setColor(new Color(0,255,0))` μπορεί να γραφεί και ως ακολούθως :

```
Color color=new Color(0,255,0);

g.setColor(color);
```

Ο πρώτος τρόπος είναι πιο σύντομος και προτιμάται τις περισσότερες φορές.

# ΚΕΦΑΛΑΙΟ 7

# ΝΗΜΑΤΑ

## 7.1 Εισαγωγή στα νήματα

Μια σημαντική δυνατότητα ενός συστήματος είναι να μπορεί να εκτελεί περισσότερες από μια εργασίες ταυτόχρονα. Αυτή η διαδικασία ονομάζεται πολυεπεξεργασία (multitasking). Το `Unix` και τα `Windows` χρησιμοποιούν πολυεπεξεργασία ενώ το `MS-DOS` όχι. Δηλαδή στο `MS-DOS` πρέπει πρώτα να τελειώσει η τρέχουσα εργασία (task) του για να αρχίσει μια άλλη.

Στην `Java` υπάρχει η δυνατότητα πολυεπεξεργασίας και αυτή πραγματοποιείται μέσω των νημάτων. ***Νήμα*** (thread) είναι μια εργασία ενός προγράμματος που τρέχει ενώ παράλληλα το υπόλοιπο μέρος του προγράμματος εκτελείται κανονικά. Σε ένα πρόγραμμα μπορούν να εισαχθούν όσα νήματα επιθυμεί ο προγραμματιστής και φυσικά μπορούν να εκτελούνται ταυτόχρονα.

Η κλάση `Thread` είναι υπεύθυνη για την πολυεπεξεργασία στην `Java` και υλοποιεί την διασύνδεση `Runnable`. Ανήκει στο πακέτο `java.lang` για αυτό δεν χρειάζεται να γίνει διαθέσιμη σε ένα πρόγραμμα μέσω μιας πρότασης `import`. Η πιο σημαντική μέθοδος της (της κλάσης Thread) είναι η `run()` που περιέχει τις εντολές των νημάτων, δηλαδή το μέρος του προγράμματος που εκτελείται ταυτόχρονα. Επειδή η κλάση `Thread` υλοποιεί γενικά τα νήματα, εξορισμού δεν κάνει τίποτα, για αυτό η μέθοδος της `run()` είναι άδεια. Έτσι στην θέση της διασύνδεσης `Runnable` χρησιμοποιείται η μέθοδος `run()`. Σε μια μικροεφαρμογή για να αρχίσει να εκτελείται ένα νήμα πρέπει να υπερβούμε την μέθοδο της `start()`. Επιπλέον πρέπει να υπερβούμε και την μέθοδο `stop()` όταν επιθυμούμε την διακοπή ενός νήματος θέτοντας του την τιμή `null`. Ανακεφαλαιώνοντας, για να γίνει ένα νηματικό πρόγραμμα σε μια μικροεφαρμογή απαιτούνται η διασύνδεση `Runnable`, τουλάχιστον ένα αντικείμενο της `Thread` και οι μέθοδοι `run()`, `start()` και `stop()`.

Οι σημαντικότεροι δημιουργοί της κλάσης `Thread` είναι:

- `Thread(Runnable r)`: Δημιουργεί ένα νήμα στο οποίο η μέθοδος `run` προέρχεται από το αντικείμενο `r`.
- `Thread(Runnable r, String str)`: Ίδια με την παραπάνω και επιπλέον το νήμα ονομάζεται `str`.

Χρήσιμες μέθοδοι της κλάσης `Thread` είναι:

- `currentThread()`: Επιστρέφει το τρέχον εκτελούμενο νήμα. Τι σημαίνει αυτό; Τα νήματα δεν εκτελούνται ταυτόχρονα; Αυτά τα ερωτήματα θα απαντηθούν στην συνέχεια του κεφαλαίου.

- `sleep(long num)`: Η μέθοδος αυτή προκαλεί την προσωρινή διακοπή ενός νήματος για `num` milliseconds.
- `sleep(long num, int n)`: Προκαλεί την προσωρινή διακοπή ενός νήματος για `num` milliseconds και `n` nanoseconds. Ο ακέραιος `n` μπορεί να πάρει τιμές από 0 έως 999999. Γενικά οι μέθοδοι `sleep(...)` μπορούν να προκαλέσουν την εξαίρεση (σφάλμα) `InterruptedException`. Επειδή υπάρχει πιθανή εξαίρεση, η μέθοδος `sleep()` πρέπει να περικλείεται στο μπλοκ `try{}catch{}`. Περισσότερα στοιχεία για τις εξαιρέσεις στο κεφάλαιο 9.
- `start()`: Έχει σαν αποτέλεσμα ένα νήμα να αρχίσει να εκτελείται διότι η εικονική μηχανή καλεί την μέθοδο `run()` του νήματος.

Όπως αναφέρθηκε παραπάνω, το καλύτερο σημείο για να δημιουργηθεί ένα νήμα σε μια μικροεφαρμογή είναι στην μέθοδο `start(...)`. Ο πιθανός κώδικας ενός νήματος στην μέθοδο `start(...)` είναι:

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

Αρχικά, ορίζεται ένα αντικείμενο της κλάσης `Thread`. Στην συνέχεια δημιουργείται το αντικείμενο `thread` και μετά καλείται η μέθοδος `start()` (της κλάσης Thread) που εκτελεί το νήμα, δηλαδή καλεί την μέθοδο `run(...)`. Η λέξη κλειδί **this** δείχνει ότι η το πρόγραμμά μας, στην συγκεκριμένη περίπτωση η μικροεφαρμογή μας, είναι υπεύθυνη να παρέχει την συμπεριφορά του νήματος. Για να είναι σωστή η λέξη **this** πρέπει το πρόγραμμα να υλοποιεί την διασύνδεση `Runnable`, δηλαδή να έχει σύνταξη:

```
public class AppThread extends Applet implements Runnable {

…

}
```

Προηγουμένως σε αυτό το κεφάλαιο έγινε ο ισχυρισμός πως τα νήματα μπορούν να τρέχουν ταυτόχρονα. Ενώ εννοιολογικά ο ισχυρισμός αυτός είναι σωστός στην πράξη συνήθως δεν είναι. Οι περισσότεροι χρήστες έχουν κάποιο PC με έναν επεξεργαστή, έτσι κάθε φορά τρέχει μόνο ένα νήμα στην μονάδα του χρόνου με τέτοιο τρόπο όμως που να δίνει την ψευδαίσθηση ότι εκτελείται με άλλα νήματα.

Το κλειδί για την πολυεπεξεργασία είναι ότι τα νήματα μπορούν να σταματούν ή αναστέλλουν την εργασία τους για κάποιο χρονικό διάστημα. Στην διάρκεια που ένα νήμα είναι σταματημένο, κάποιο άλλο νήμα μπορεί χρησιμοποιήσει τον επεξεργαστή. Η διαδικασία της διακοπής ενός νήματος για κάποιο χρονικό διάστημα γίνεται με την μέθοδο sleep(…).

Το ακόλουθο παράδειγμα δείχνει πόση ώρα είναι ανοιχτό το συγκεκριμένο πρόγραμμα. Αυτό επιτυγχάνεται με την βοήθεια των νημάτων. Η γραφική διασύνδεση του προγράμματος γίνεται με την βοήθεια του AWT το οποίο αναλύεται στο επόμενο κεφάλαιο. Συνοπτικά υπάρχουν τρία πεδία κειμένου και τρεις ετικέτες. Στα πεδία κειμένου αποτυπώνονται οι ώρες, τα λεπτά και τα δευτερόλεπτα από το αριστερότερο ως το δεξιότερο πεδίο αντίστοιχα.

Ο μηχανισμός του προγράμματος είναι ο εξής: Ορίζονται τρεις ακέραιες μεταβλητές για τις ώρες (μεταβλητή h), τα λεπτά (μεταβλητή m), τα δευτερόλεπτα (μεταβλητή s) και ένα νήμα. Το προσδοκώμενο αποτέλεσμα είναι η μεταβλητή s να αυξάνεται κατά μία μονάδα σε κάθε δευτερόλεπτο. Αυτό επιτυγχάνεται με τον ορισμό του χρονικού διαστήματος που διακόπτεται το νήμα σε ένα δευτερόλεπτο. Ο κώδικας που προκαλεί την διακοπή του ενός δευτερολέπτου είναι:

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

Δηλαδή το νήμα σταματάει για 1000 milliseconds ή αλλιώς για ένα δευτερόλεπτο. Τα μπλοκ `try`, `catch` χρησιμεύουν στην περίπτωση που η μέθοδος `sleep(...)` προκαλέσει σφάλμα.

Βεβαίως ορίζονται όρια στις μεταβλητές δηλαδή τα δευτερόλεπτα και τα λεπτά φτάνουν μέχρι την τιμή 59 και μετά ξαναρχίζουν από το μηδέν. Επίσης η μεταβλητή των ωρών όταν περάσει την τιμή 23 ξαναρχίζει από το μηδέν. Εννοείται ότι κάθε φορά που περνάνε εξήντα δευτερόλεπτα προστίθεται μια μονάδα στην μεταβλητή των λεπτών. Το ίδιο συμβαίνει και με τις ώρες. Αυτές οι πράξεις γίνονται με τον παρακάτω κώδικα.

```
if(s==60){m+=1;s=0;}
if(m==60){h+=1;m=0;}
if(h==24) h=0;
```

Ο ολοκληρωμένος κώδικας της μικροεφαρμογής είναι:




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

public class AppThread extends Applet implements Runnable{
    Thread thread,current;
    TextField text1,text2,text3;
    Label label1,label2,label3;
    int h=0,m=0,s=0;
```

```
    public void init(){
     // Γραφική διασύνδεση (graphical interface) του προγράμματος
        text1=new TextField(2);
        text2=new TextField(2);
        text3=new TextField(2);
        label1=new Label("Ωρες");
        label2=new Label("Λεπτα");
        label3=new Label("Δευτερολεπτα");
        text1.setEditable(false);
        text2.setEditable(false);
        text3.setEditable(false);
        add(label1);
        add(text1);
        add(label2);
        add(text2);
        add(label3);
        add(text3);
    }

     // Δημιουργία και έναρξη του νήματος
    public void start(){
        if(thread==null){
            thread=new Thread(this);
            thread.start();
        }
    }

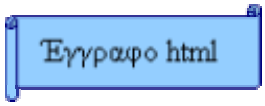
     // Διακοπή του νήματος
    public void stop(){
        if(thread!=null){
            thread=null;
        }
    }

     // Εργασία νήματος
    public void run(){
        current=new Thread();
        current=Thread.currentThread();
        while(current==thread){
            text1.setText(""+h);
            text2.setText(""+m);
            text3.setText(""+s);
            s+=1;
            if(s==60){m+=1;s=0;}
            if(m==60){h+=1;m=0;}
            if(h==24) h=0;
            try{
                thread.sleep(1000);
            }
            catch(InterruptedException e)
            {
            }
        }
   }

}
```

*Πρόγραμμα 7.1 – AppThread.java*

- source code




```
<APPLET CODE="AppThread.class" WIDTH=350 HEIGHT=50>
</APPLET>
```

*appthread.html*

Στο πρόγραμμα 7.1 η πρόταση `current = Thread.currentThread()` επιστρέφει το νήμα που βρίσκεται σε εξέλιξη. Η πρόταση `while` προκαλεί ένα ατέρμονο βρόχο. Οι μέθοδοι `setText(...)` θέτουν το κείμενο στο κάθε πεδίο κειμένου. Στο συγκεκριμένο παράδειγμα περιλαμβάνουν ομοιωματικά διότι το άθροισμα ενός `String` με ένα αριθμητικό δίνει ένα String και ο τύπος ορίσματος αυτής της μεθόδου είναι String.

Όταν κλείνει η μικροεφαρμογή καλείται η μέθοδος `stop()` η οποία θέτει στο αντικείμενο `thread` την τιμή `null`. Αυτό έχει σαν αποτέλεσμα να σταματήσει η εκτέλεση της μεθόδου `run()`.




---




---

# 8.1 Γενικά για το AWT (Abstract Window Toolkit)

---

Το `AWT` είναι ένα σύνολο κλάσεων που επιτρέπει την δημιουργία γραφικής διασύνδεσης χρήστη – `Graphical User Interface (GUI).` Το `GUI` χρησιμοποιείται για τον τρόπο εμφάνισης και χειρισμού του προγράμματος από τον χρήστη. Οι ιεραρχία των κυριοτέρων κλάσεων του AWT υπάρχει στο διάγραμμα 1.

---

# 8.2 Γραφική διασύνδεση χρήστη

---

Υπάρχουν δύο βήματα για την δημιουργία `GUI` εφαρμογής ή μικροεφαρμογής :

- Σύνθεση μιας γραφικής διασύνδεσης χρήστη προσθέτοντας συστατικά (components) στους υποδοχείς (containers).
- Δημιουργία χειριστών συμβάντων που ανταποκρίνονται στην αλληλεπίδραση χρήστη με το `GUI`.

---

# 8.3 Σύνθεση γραφικής διασύνδεσης με το AWT

---

Η σύνθεση γραφικής διασύνδεσης ή διεπαφής χρήστη χρειάζεται να περιέχει, τρία στοιχεία για να ολοκληρωθεί με επιτυχία:

- Συστατικά (components): Τα ***συστατικά*** είναι ό,τι μπορεί να χρησιμοποιηθεί σε μια διασύνδεση χρήστη όπως κουμπιά, ετικέτες, λίστες επιλογής, περιοχές κειμένου κ.α.
- Υποδοχείς (containers): Ο ***υποδοχέας*** είναι ένα συστατικό που μπορεί να περιέχει και άλλα συστατικά. Οι υποδοχείς είναι δευτερεύουσες κλάσεις της κλάσης `Container`, συνεπώς και της `Component`. Οι κύριοι υποδοχείς μπορεί να είναι το παράθυρο μιας μικροεφαρμογής (applet), ένα αντικείμενο της κλάσης `Frame`, ένα αντικείμενο της κλάσης `Window` ή ένα αντικείμενο της κλάσης `Dialog`. Για τους υποδοχείς θα υπάρξει αναφορά στο κεφάλαιο 8.3.3.
- Διαχειριστές διάταξης (layouts) : Είναι ο τρόπος που διατάσσονται τα συστατικά σε έναν υποδοχέα. Δηλαδή ο διαχειριστής διάταξης χρησιμοποιείται για την δημιουργία δομημένης και φιλικής προς τον χρήστη γραφικής διασύνδεσης.

---

### 8.3.1 Συστατικά

---

Για τους διαχειριστές διάταξης θα υπάρξει αναφορά στο επόμενο κεφάλαιο. Ο προεπιλεγμένος διαχειριστής, όταν ο υποδοχέας είναι μια μικροεφαρμογή, είναι η διάταξη ροής (flow layout), ο χειρισμός της οποίας γίνεται στην κλάση `FlowLayout`. Ο συγκεκριμένος διαχειριστής τοποθετεί κάθε συστατικό σε μια γραμμή του υποδοχέα από τα αριστερά προς τα δεξιά, μέχρι να γεμίσει η γραμμή και μετά συνεχίζει στην επόμενη. Τα συστατικά με αυτόν τον διαχειριστή είναι κεντραρισμένα σε μια γραμμή του υποδοχέα.

Όπως αναφέρθηκε προηγουμένως τα συστατικά τοποθετούνται σε έναν υποδοχέα. Επειδή ο υποδοχέας είναι από μόνος του ένα συστατικό μπορεί να προστεθεί σε άλλους υποδοχείς.

Για την πρόσθεση ενός συστατικού σε έναν υποδοχέα πρώτα δημιουργείται το συστατικό και έπειτα προστίθεται στον υποδοχέα με την μέθοδο `add()` η οποία υπάρχει για όλους τους υποδοχείς καθώς και για μερικά συστατικά. Όταν προστίθεται ένα συστατικό σε έναν υποδοχέα δεν καθορίζονται οι συντεταγμένες του συστατικού που θα τοποθετηθεί στον υποδοχέα. Η θέση του συστατικού στον υποδοχέα καθορίζεται από τον διαχειριστή διάταξης.

Μια μικροεφαρμογή είναι ήδη υποδοχέας (γιατί κληρονομεί την κλάση Applet που είναι υποδοχέας) επομένως το μόνο που χρειάζεται είναι η πρόσθεση των συστατικών με την μέθοδο `add()`. Τα περισσότερα προγράμματα αυτού του κεφαλαίου θα χρησιμοποιούν τον υποδοχέα μικροεφαρμογής. Όπως αναφέρθηκε προηγουμένως, ο προεπιλεγμένος διαχειριστής διάταξης στις μικροεφαρμογές είναι η διάταξη ροής. Δηλαδή στα προγράμματα που ακολουθούν δεν θα αναφέρεται (η διάταξη ροής) αλλά θα εννοείται.

Παρατήρηση: Όταν προστίθεται ένα συστατικό στην μικροεφαρμογή δεν προκαλεί την εμφάνιση του. Θα εμφανιστεί μόνο όταν κληθεί η μέθοδος `paint(...)` της μικροεφαρμογής. Αυτό είναι κάτι που χειρίζεται η `Java` στο παρασκήνιο. Δηλαδή για την εμφάνιση των συστατικών δεν χρειάζεται να γραφεί κώδικας στην μέθοδο `paint(...)`.

Όλα τα συστατικά του `AWT` κληρονομούν τις μεθόδους της κλάσης `Component`. Οι πιο συχνά χρησιμοποιούμενες μέθοδοι της κλάσης `Component` είναι οι παρακάτω:

- `getHeight()`: Επιστρέφει σε pixels το ύψος του συστατικού που προσδιορίζει.

- `getWidth()`: Επιστρέφει σε pixels το πλάτος του συστατικού που προσδιορίζει.

Π.χ.

```
int x=systatiko.getHeight();

int y=systatiko.getWidth();
```

- `getX()`: Επιστρέφει σε pixels την `x` συντεταγμένη του συστατικού σε σχέση με τον υποδοχέα.
- `getY()`: Επιστρέφει σε pixels την `y` συντεταγμένη του συστατικού σε σχέση με τον υποδοχέα.
- `setBounds(int x, int y, int width, int height)`: Μια πολλή χρήσιμη μέθοδος, που προσδιορίζει το μέγεθος και τις συντεταγμένες του συστατικού. Οι μεταβλητές `x`, `y` είναι οι συντεταγμένες του συστατικού πάνω στον υποδοχέα και `width`, `height` είναι το πλάτος και το ύψος του συστατικού.
- `setEnabled(boolean bl)`: Καθορίζει την κατάσταση του συστατικού. Αν το όρισμα `bl` είναι **true** το συστατικό θα αλληλεπιδρά με τον χρήστη και θα εμφανίζεται κανονικά. Αν το `bl` είναι **false** το συστατικό θα αγνοεί την αλληλεπίδραση και θα εμφανίζεται απενεργοποιημένο. Δεν επιστρέφει τιμή.
- `setVisible(boolean bl)`: Καθορίζει αν το συστατικό θα είναι ορατό ή αόρατο στον χρήστη. Αν το όρισμα της μεθόδου είναι **true** το συστατικό θα είναι ορατό, αλλιώς θα είναι αόρατο.

Τα κύρια συστατικά του `AWT` είναι:

---

### 8.3.1.1 Κουμπιά (κλάση Button)

---

Για την δημιουργία ενός συστατικού κουμπιού χρησιμοποιείται η κλάση `Button`. Ακολούθως δίνονται δύο δημιουργοί της:

- `Button()`: Δημιουργεί ένα κουμπί χωρίς ετικέτα.
- `Button(String str)`: Δημιουργεί ένα κουμπί με την ετικέτα `str`.

Δύο χρήσιμες μέθοδοι της κλάσης `Button` είναι οι:

- `setLabel(String str)`: Θέτει σε ένα κουμπί την ετικέτα `str`. Δεν επιστρέφει κάποια τιμή.
- `getLabel()`: Επιστρέφει ένα **`String`** το οποίο είναι η ετικέτα του κουμπιού που προσδιορίζει.

Στο παράδειγμα 8.1 δημιουργούνται 4 κουμπιά με την προεπιλεγμένη διάταξη `FlowLayout`.




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

public class AppButton extends Applet {
     Button enter = new Button();
     Button exit = new Button("Exit");
     Button again = new Button("Again");
     Button tryagain = new Button("Try again");

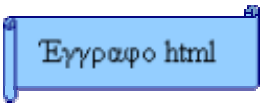
      // Αρχικοποίηση μικροεφαρμογής
     public void init() {
          enter.setLabel("Enter");
          add(enter);
          add(exit);
          add(again);
          add(tryagain);
     }

      // Εκτύπωση της ετικέτας του κουμπιού tryagain
     public void paint(Graphics g){
          g.drawString(tryagain.getLabel(),30,100);
     }

}
```

*Παράδειγμα 8.1-AppButton.java*

- source code




```
<APPLET CODE="AppButton.class" WIDTH=150 HEIGHT=150>
</APPLET>
```

*appbutton.html*

Στο παράδειγμα 8.1 ορίζονται τέσσερα κουμπιά από τα οποία το ένα δεν έχει ετικέτα. Στην μέθοδο `init()` προστίθενται τα κουμπιά στον υποδοχέα με την μέθοδο `add()` και ορίζεται η ετικέτα του κουμπιού που δεν είχε οριστεί με την μέθοδο `setLabel(...)`. Τέλος, στην μέθοδο `paint(...)` τυπώνεται η ετικέτα του κουμπιού `tryagain`.

---

### 8.3.1.2 Ετικέτες (κλάση Label)

Για την δημιουργία ενός συστατικού ετικέτας χρησιμοποιείται η κλάση `Label` η οποία έχει τους ακόλουθους δημιουργούς:

- `Label()`: Δημιουργεί μία ετικέτα χωρίς κείμενο.
- `Label(String str)`: Δημιουργεί μια ετικέτα με κείμενο `str` που είναι στοιχισμένο αριστερά.
- `Label(String str, int x)`: Δημιουργεί μια ετικέτα με κείμενο `str` με στοίχιση που δηλώνεται από τον ακέραιο `x`. Ο `x` μπορεί να πάρει τις τιμές `Label.RIGHT`, `Label.LEFT` και `Label.CENTER` για στοίχιση στα δεξιά, στα αριστερά και στο κέντρο αντίστοιχα.

Δύο χρήσιμες μέθοδοι της κλάσης `Label` είναι οι ακόλουθες:

- `setText(String str)`: Θέτει ένα κείμενο σε μια ετικέτα. Δεν επιστρέφει καμία τιμή.
- `getText()`: Επιστρέφει το κείμενο της ετικέτας που προσδιορίζει.

Το παράδειγμα 8.2 περιέχει αντικείμενα της κλάσης `Label` και της προαναφερθείσες μεθόδους.

Πηγαίο αρχείο

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

public class AppLabel extends Applet {
    Label label1 = new Label();
    Label label2 = new Label("Πρόγραμμα στο GUI" );
    Label label3 = new Label("Διαχειριστής", Label.RIGHT);
    GridLayout h=new GridLayout(7,1);

    public void init() {
        setLayout(h); // Θέτει τον διαχειριστή διάταξης
        label1.setText("Παράδειγμα στα Labels");
        add(label1);
        add(label2);
        add(label3);
    }

    public void paint(Graphics g){
        g.drawString(label3.getText(),50,150);
    }

}
```

*Παράδειγμα 8.2-AppLabel.java*

- source code

Έγγραφο html

```
<APPLET CODE="AppLabel.class" WIDTH=200 HEIGHT=200>
</APPLET>
```

*applabel.html*

Στο παράδειγμα 8.2 χρησιμοποιείται ο διαχειριστής διάταξης `GridLayout` (διάταξη πλέγματος) διότι οι ετικέτες δεν μπορούν να εισαχθούν στον υποδοχέα με τον προεπιλεγμένο διαχειριστή. Ο διαχειριστής `GridLayout` δημιουργεί 7 γραμμές και μία στήλη όπου στις πρώτες τρεις εμφανίζονται οι ετικέτες. Η μέθοδος `setLayout(...)` κάνει έναν διαχειριστή διάταξης ενεργό (τρέχων). Περισσότερες πληροφορίες για τον διαχειριστή διάταξης στο κεφάλαιο 8.3.2.3.

---

### 8.3.1.3 Πεδία κειμένου (κλάση TextField)

---

Τα ***πεδία κειμένου*** είναι συστατικά που δέχονται κείμενο μιας γραμμής το οποίο είναι επεξεργάσιμο. Η υπεύθυνη κλάση για την δημιουργία πεδίου κειμένου είναι η `TextField` η οποία έχει τους ακόλουθους δημιουργούς.

- `TextField()`: Δημιουργεί ένα κενό πεδίο κειμένου χωρίς ορισμένο πλάτος.
- `TextField(int x)`: Δημιουργεί ένα κενό πεδίο κειμένου με καθορισμένο πλάτος `x`.
- `TextField(String str)`: Δημιουργεί ένα πεδίο με κείμενο `str` και μη καθορισμένο πλάτος
- `TextField(String str, int x)`: Δημιουργεί ένα πεδίο κειμένου `str` και καθορισμένο πλάτος `x`.

Οι κύριες μέθοδοι της κλάσης `TextField` είναι:

- `setText(String str)`: Θέτει το κείμενο `str` στο πεδίο κειμένου. Δεν επιστρέφει κάποια τιμή.
- `setEchoChar(Char ch)`: Θέτει τον χαρακτήρα `ch` στο πεδίο κειμένου, όταν πατηθεί οποιοδήποτε πλήκτρο. Αυτή η μέθοδος χρησιμοποιείται για λόγους ασφαλείας, όταν για παράδειγμα ο χρήστης εισάγει μια κωδική λέξη (password) σε ένα πεδίο κειμένου, έτσι ώστε να μην γίνεται αντιληπτό το περιεχόμενο της. Δεν επιστρέφει κάποια τιμή.

Χρήσιμες μέθοδοι που κληρονομούνται από την κλάση `TextComponent` του πακέτου `java.awt` είναι οι παρακάτω:

- `getText()`: Επιστρέφει το περιεχόμενο της περιοχής κειμένου.
- `setEditable(boolean bl)`: Θέτει, αν επιτρέπεται ή όχι, η επεξεργασία του περιεχομένου του πεδίου κειμένου. Αν η μεταβλητή `bl` είναι **`true`** επιτρέπεται η επεξεργασία του πεδίου κειμένου, διαφορετικά δεν επιτρέπεται. Δεν επιστρέφει κάποια τιμή.

Το παράδειγμα 8.3 διασαφηνίζει τις προηγούμενες μεθόδους.




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

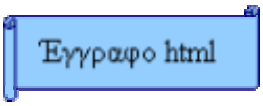
public class AppTextField extends Applet {
    Label name=new Label("Name of company");
    TextField textfield0=new TextField("Institute of Java");
    Label console = new Label("Console");
    TextField textfield1= new TextField(20);
    Label username = new Label("Username");
    TextField textfield2 = new TextField(20);
    Label password = new Label("Password");
    TextField textfield3 = new TextField(20);

    public void init() {
        add(name);
        textfield0.setEditable(false);
        add(textfield0);
        add(console);
        textfield1.setText("Windows 2000");
        add(textfield1);
        add(username);
        add(textfield2);
        add(password);
        textfield3.setEchoChar('*');
        add(textfield3);
    }

}
```

*Παράδειγμα 8.3-AppTextField*

- source code




```
<APPLET CODE="AppTextField.class" WIDTH=250 HEIGHT=200>
</APPLET>
```

*apptextfield.html*

Στο παραπάνω παράδειγμα ορίζονται τέσσερα αντικείμενα της κλάσης `Label` και τέσσερα της κλάσης `TextField`. Το πρώτο πεδίο κειμένου ορίζεται ως μη επεξεργάσιμο με την πρόταση `textfield0.setEditable(false)` στην μέθοδο `init()`. Στο τελευταίο πεδίο κείμενου οτιδήποτε πληκτρολογείται εμφανίζεται σαν τον χαρακτήρα '*' στο πεδίο κειμένου. Αυτό επιτυγχάνεται με την πρόταση `textfield3.setEchoChar('*')`.

Παρατήρηση: Ένας γενικός κανόνας είναι ότι οι μέθοδοι που έχουν το πρόθεμα `set` δεν επιστρέφουν τιμή, ενώ αυτές που έχουν το πρόθεμα `get` επιστρέφουν.

---

### 8.3.1.4 Περιοχές κειμένου (κλάση TextArea)

---

Οι ***περιοχές κειμένου*** είναι πεδία κειμένου πολλών γραμμών δηλαδή ένα συστατικό το οποίο μπορεί να επεξεργαστεί περισσότερες από μία γραμμές κειμένου. Οι περιοχές κειμένου δημιουργούνται από την κλάση `TextArea` η οποία έχει τους παρακάτω δημιουργούς:

- `TextArea()`: Δημιουργεί μια άδεια περιοχή κειμένου χωρίς καθορισμένο αριθμό γραμμών και στηλών.
- `TextArea(int x,int y)`: Δημιουργεί μια κενή περιοχή κειμένου με καθορισμένο αριθμό γραμμών και στηλών όπου σε κάθε στήλη αντιστοιχεί το πάχος ενός χαρακτήρα.
- `TextArea(String str)`: Δημιουργεί μια περιοχή κειμένου με καθορισμένο κείμενο `str`, χωρίς καθορισμένα όρια.
- `TextArea(String str, int x, int y)` : Δημιουργεί μια περιοχή κειμένου με κείμενο `str` και `x` σειρές, `y` στήλες.
- `TextArea(String str, int x, int y , int z)`: Δημιουργεί μια περιοχή κείμενου με κείμενο str και x σειρές, y στήλες. Η μεταβλητή z καθορίζει αν θα φαίνονται μπάρες ολίσθησης (scrollbar) και πως θα φαίνονται. Μπορεί να πάρει τις ακέραιες τιμές, scrollbars_both, scrollbars_Vertical_only, scrollbars_Horizontal_only και scrollbars_none που σημαίνουν διπλή μπάρα κύλισης, οριζόντια μπάρα κύλισης, κάθετη μπάρα κύλισης και ανυπαρξία μπάρα, κύλισης αντίστοιχα.

Χρήσιμες μέθοδοι της κλάσης `TextArea` είναι:

- `insert(String str, int x)`: Η συγκεκριμένη μέθοδος εισάγει το κείμενο `str` στην θέση `x` της περιοχής κειμένου. Δεν επιστρέφει τιμή.
- `replaceRange(String str, int x, int y)`: Η μέθοδος αυτή αντικαθιστά το περιεχόμενο της περιοχής κειμένου ανάμεσα στις θέσεις `x` και `y` με το κείμενο `str`.

Οι προαναφερθείσες μέθοδοι `getText()` και `setEditable(...)` της κλάσης `TextComponent` κληρονομούνται και από την `TextArea`.

Το παράδειγμα 8.4 αναφέρεται στις παραπάνω κλάσεις.




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

public class AppTextArea extends Applet {
    String str = "A platform is the hardware or software \n"+
    "environment in which a program runs\n"+
```

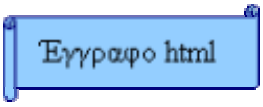
```
    "The Java platform differs from most\n"+
    "other platforms in that it's a software-only \n"+
    "platform that runs on top of other, \n"+
    "hardware-based platforms.";
    TextArea textarea= new TextArea(str,3,30,TextArea.SCROLLBARS_BOTH);

    public void init() {
        textarea.replaceRange("TextArea",91,100);
        add(textarea);
    }

}
```

*Παράδειγμα 8.4-AppTextArea.java*

- source code




```
<APPLET CODE="AppTextArea.class" WIDTH=230 HEIGHT=100>
</APPLET>
```

*apptextatrea.html*

Στο παραπάνω παράδειγμα δημιουργείται ένα **String** το οποίο έχει τρεις γραμμές. Ο χαρακτήρας '\n' χρησιμοποιείται για αλλαγή γραμμής και δεν είναι ορατός. Η περιοχή κειμένου ορίζεται ώστε να μπορεί, αν χρειαστεί, να έχει μπάρες κύλισης. Τέλος η λέξη TextField στο **String** αντικαθίσταται μέσω της μεθόδου `replaceRange(...)` με την λέξη `TextArea`.

---

### 8.3.1.5 Πλαίσια ελέγχου (κλάση Checkbox)

---

Τα ***πλαίσια ελέγχου*** είναι πλαίσια που μπορεί να είναι επιλεγμένα ή όχι και μπορούν να έχουν και ετικέτα. Τα πλαίσια ελέγχου μπορούν να είναι και μη αποκλειστικά, δηλαδή να είναι επιλεγμένα ταυτόχρονα. Τα πλαίσια ελέγχου δημιουργούνται με την κλάση Checkbox. Για να γίνουν τα πλαίσια ελέγχου αποκλειστικά πρέπει να οργανωθούν σε μια ομάδα πλαισίων ελέγχου. Αυτή η ομάδα δημιουργείται από την κλάση `CheckboxGroup`. Οι δημιουργοί της κλάση `Checkbox` είναι οι παρακάτω:

- `Checkbox()`: Δημιουργεί ένα πλαίσιο ελέγχου μη επιλεγμένο και χωρίς ετικέτα.
- `Checkbox(String str)`: Δημιουργεί ένα πλαίσιο ελέγχου μη επιλεγμένο με ετικέτα `str`.
- `Checkbox(String str, boolean bl)`: Δημιουργεί ένα πλαίσιο ελέγχου με ετικέτα `str`. Αν η μεταβλητή `bl` είναι **true** το πλαίσιο είναι επιλεγμένο διαφορετικά δεν είναι.

- `Checkbox(String str, boolean bl, CheckBoxGroup cbg)`: Είναι ίδια με την παραπάνω με την διαφορά ότι αποτελεί μέλος της ομάδας πλαισίων ελέγχου `cbg`. Αυτό σημαίνει ότι μόνο ένα από τα πλαίσια ελέγχου της `cbg` μπορεί να είναι επιλεγμένο.

Ο δημιουργός της κλάσης `CheckBoxGroup` είναι ο παρακάτω:

- `CheckboxGroup()`: Δημιουργεί ένα αντικείμενο της κλάσης `CheckBoxGroup`.

Χρήσιμες μέθοδοι της κλάσης Checkbox είναι:

- `setCheckboxGroup(CheckboxGroup cbg)`: Θέτει την ομάδα πλαισίων ελέγχου `cbg` που ανήκει το πλαίσιο ελέγχου. Η μέθοδος δεν επιστρέφει τιμή.
- `getState()`: Επιστρέφει την κατάσταση ενός πλαισίου δηλαδή εάν είναι επιλεγμένο (true) ή όχι (false).
- `setState(boolean bl)`: Θέτει την κατάσταση ενός πλαισίου με την τιμή της μεταβλητής `bl` δηλαδή αν είναι **true** το πλαίσιο είναι επιλεγμένο διαφορετικά δεν είναι.

Το παράδειγμα 8.5 αποσαφηνίζει τις παραπάνω μεθόδους.

Πηγαίο αρχείο

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

public class AppCheckbox extends Applet {
    CheckboxGroup cbg=new CheckboxGroup();
    Checkbox c1 = new Checkbox("Πρωτη επιλογη");
    Checkbox c2 = new Checkbox("Δευτερη επιλογη", true, cbg);
    Checkbox c3 = new Checkbox("Τριτη επιλογη");

    public void init() {
        c1.setCheckboxGroup(cbg);
        add(c1);
        add(c2);
        c3.setState(true);
        c3.setEnabled(false);
        add(c3);
    }

}
```

*Παράδειγμα 8.5-AppCheckbox*

- source code

Έγγραφο html

```
<APPLET CODE="AppCheckbox.class" WIDTH=250 HEIGHT=150>
</APPLET>
```

Στο παράδειγμα 8.5 ορίζονται 3 πλαίσια ελέγχου. Τα δύο από αυτά ανήκουν σε ένα πλαίσιο ομάδων ελέγχου `cbg` για αυτό έχουν και διαφορετικό σχήμα (είναι στρόγγυλα). Επειδή τα πλαίσια ανήκουν σε ομάδα πλαισίων είναι αποκλειστικά δηλαδή κάθε φορά που επιλέγεται κάποιο από τα δύο το άλλο αποεπιλέγεται. Το τρίτο πλαίσιο είναι επιλεγμένο, αλλά και απενεργοποιημένο, αφού εμφανίζεται γκρι και ο χρήστης δεν μπορεί να αλληλεπίδραση με αυτό. Αυτό γίνεται με την μέθοδο `setEnabled(...)` που κληρονομείται από την κλάση `Component`.

---

### 8.3.1.6 Λίστες επιλογής (κλάση Choice)

---

Οι ***λίστες επιλογής*** είναι αναδιπλούμενες λίστες που επιτρέπουν την επιλογή ενός από τα στοιχεία της λίστας. Η υπεύθυνη κλάση για τις λίστες επιλογής είναι η `Choice` η οποία έχει τον παρακάτω δημιουργό:

- `Choice()`: Δημιουργεί μια λίστα επιλογής χωρίς στοιχεία.

Χρήσιμες μέθοδοι της κλάσης `Choice` είναι οι ακόλουθες:

- `add(String str)`: Προσθέτει ένα στοιχείο με ετικέτα `str` στην λίστα επιλογής. Δεν επιστρέφει τιμή.
- `getItem(int x)`: Επιστρέφει το κείμενο του `x` στοιχείου της λίστας επιλογής. Η αρίθμηση των στοιχείων στην λίστα επιλογής αρχίζει από το μηδέν.
- `getItemCount()`: Επιστρέφει τον αριθμό των στοιχείων της λίστας επιλογής.
- `getSelectedIndex()`: Επιστρέφει την θέση του επιλεγμένου στοιχείου.
- `getSelectedItem()`: Επιστρέφει το κείμενο του επιλεγμένου στοιχείου της λίστας.
- `select(int x)`: Η μέθοδος αυτή επιλέγει το x στοιχείο της λίστας επιλογής.
- `select(String str)`: Η μέθοδος αυτή επιλέγει το στοιχείο εκείνο που έχει κείμενο `str`.




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

public class AppChoice extends Applet
{
    Choice c = new Choice();

    public void init() {
     /* Οι 4 πρώτες μέθοδοι add προσθέτουν στοιχεία στην λίστα επιλογής ενώ η τελευταία προσθέτει
     την λίστα επιλογής στον υποδοχέα */
        c.add("First day");
```

```
        c.add("Second day");
        c.add("Third day");
        c.add("Four day");
        c.select(3);
        add(c);
    }

    public void paint(Graphics g){
        g.drawString("Το δευτερο στοιχειο εχει κειμενο "+c.getItem(1),20,100);
        g.drawString("Τα στοιχεια στην λιστα ειναι "+c.getItemCount(),20,120);
        g.drawString("Το τρεχον στοιχειο εχει κειμενο "
                                        +c.getSelectedItem(),20,140);
    }

}
```

*Παράδειγμα 8.6-AppChoice*

- source code

Έγγραφο html

```
<APPLET CODE="AppChoice.class" WIDTH=300 HEIGHT=150>
</APPLET>
```

*appchoice.html*

Στο παράδειγμα 8.6 δημιουργείται μια λίστα επιλογής με τέσσερα στοιχεία. Το επιλεγμένο στοιχείο είναι το τέταρτο. Η μέθοδος `paint()` εμφανίζει το κείμενο του δεύτερου στοιχείου της λίστας, το πλήθος των στοιχείων της λίστας, καθώς και το κείμενο του τρέχοντος στοιχείου.

---

### 8.3.1.7 Λίστες κύλισης (κλάση List)

---

Οι ***λίστες κύλισης*** είναι λίστες που μπορούν να επιλεγούν περισσότερα από ένα στοιχεία τους. Δεν είναι αναδιπλούμενες σαν τις λίστες επιλογής και αν περιέχουν περισσότερα στοιχεία από όσα εμφανίζονται, έχουν μια γραμμή κύλισης για την εμφάνιση όλων των στοιχείων. Η κλάση που δημιουργεί τις λίστες επιλογής είναι η `List` η οποία έχει τους ακόλουθους δημιουργούς:

- `List()`: Δημιουργεί μια κενή λίστα κύλισης.
- `List(int x)`: Δημιουργεί μια λίστα κύλισης με x σειρές.
- `List(int x, boolean bl)`: Δημιουργεί μια λίστα κύλισης με `x` σειρές. Η μεταβλητή `bl` καθορίζει αν μπορούν να επιλεγούν περισσότερα από ένα στοιχεία. Αν είναι **true** επιτρέπεται η πολλαπλή επιλογή, διαφορετικά δεν είναι εφικτή.

Οι μέθοδοι `add()`, `getItem(...)`, `getItemCount()`, `getSelectedIndex()`, `getSelectedItem()` και `select(...)` της κλάσης `Choice` υπάρχουν και στην κλάση `List` και επιτελούν τις ίδιες λειτουργίες.

Επιπρόσθετες μέθοδοι της κλάσης List είναι:

- `getSelectedIndexes()`: Επιστρέφει ένα πίνακα που περιέχει την θέση των επιλεγμένων στοιχείων.
- `getSelectedItems()`: Επιστρέφει ένα πίνακα που περιέχει το κείμενο του κάθε επιλεγμένου στοιχείου.

Πηγαίο αρχείο

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

public class AppList extends Applet {
    List ls = new List(7, true);

    public void init() {
        ls.add("Πρωτη επιλογη");
        ls.add("Δευτερη επιλογη");
        ls.add("Τριτη επιλογη");
        ls.add("Τεταρτη επιλογη");
        ls.add("Πεμπτη επιλογη");
        ls.add("Εκτη επιλογη");
        ls.add("Εβδομη επιλογη");
        ls.select(1);
        ls.select(2);
        ls.select(5);
        ls.select(6);
        add(ls);
    }

    public void paint(Graphics g){
        String[] str=ls.getSelectedItems();
        for(int i=0;i<str.length;i++)
        g.drawString("Επιλεγμενο στοιχειο με κειμενο "+str[i],20,180+i*20);
    }

}
```

*Παράδειγμα 8.7-AppList.java*

- source code

Έγγραφο html

```
<APPLET CODE="AppList.class" WIDTH=400 HEIGHT=300>
</APPLET>
```

Στο παράδειγμα 8.7 δημιουργείται μια λίστα κύλισης με εφτά στοιχεία από τα οποία τα τέσσερα είναι επιλεγμένα. Το κείμενο των επιλεγμένων στοιχείων αποθηκεύεται σε ένα πίνακα **String** και μετά τυπώνεται με την μέθοδο `drawstring(...)`.

---

### 8.3.2 Διαχειριστές διάταξης

---

Οι ***διαχειριστές διάταξης*** καθορίζουν τον τρόπο που τα συστατικά διατάσσονται σε έναν υποδοχέα. Ο προεπιλεγμένος διαχειριστής είναι ο `FlowLayout`. Ένας διαχειριστής διάταξης γίνεται ενεργός με την μέθοδο `setLayout(αντικείμενο διαχειριστή)`. Ο διαχειριστής διάταξης πρέπει να προστίθεται στον υποδοχέα πριν από την πρόσθεση στοιχείων σε αυτόν (υποδοχέα). Επιπλέον σε μερικούς διαχειριστές παίζει ρόλο η σειρά με την οποία προστίθενται τα συστατικά στον υποδοχέα. Στην συνέχεια αναλύονται πέντε διαχειριστές διάταξης.

---

#### 8.3.2.1 Διάταξή ροής (κλάση FlowLayout)

---

Η κλάση `FlowLayout` είναι υπεύθυνη για την δημιουργία της ***διάταξης ροής***. Η διάταξη ροής διατάσσει τα συστατικά από αριστερά προς τα δεξιά σε έναν υποδοχέα. Όταν η γραμμή του υποδοχέα γεμίσει ο διαχειριστής συνεχίζει στην επόμενη γραμμή. Οι δημιουργοί της κλάσης FlowLayout είναι:

- `FlowLayout()`: Επειδή δεν έχει κανένα όρισμα θέτει στοίχιση κέντρου.
- `FlowLayout(int x)`: Ο ακέραιος `x` αναφέρεται στην στοίχιση των συστατικών. Οι τιμές που μπορεί να πάρει είναι `FlowLayout.LEFT`, `FlowLayout.RIGHT`, `FlowLayout.CENTER`, που αντιστοιχούν σε αριστερή, δεξιά και κεντρική στοίχιση.
- `FlowLayout(int x, int y1, int y2)`: Είναι ίδια με την παραπάνω και επιπλέον `y1`, `y2` είναι το οριζόντιο και το κατακόρυφο κενό μεταξύ των συστατικών αντίστοιχά.

Στον συγκεκριμένο διαχειριστή παίζει ρόλο η σειρά με την οποία προστίθενται τα συστατικά στον υποδοχέα. Μια παραλλαγή της μικροεφαρμογής 8.1 είναι το παρακάτω παράδειγμα στο οποίο τα διαφορετικά στοιχεία από το πρωτότυπο είναι: Πρώτον, αλλάζει η στοίχιση κέντρου και μετατρέπεται σε αριστερή στοίχιση. Δεύτερον, ορίζεται το οριζόντιο διάκενο σε 10 pixels και το κατακόρυφο σε 20 pixels.



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

public class AppButton1 extends Applet {
    …
    FlowLayout fl=new FlowLayout(FlowLayout.LEFT,10,20);

    public void init() {
        setLayout(fl);
        …
    }

    public void paint(Graphics g){
        …
    }

}
```

*Παράδειγμα 8.8-AppButton2.java*

- source code

---

### 8.3.2.2 Διάταξη περιγράμματος (κλάση BorderLayout)

---

Η ***διάταξη περιγράμματος*** χωρίζει έναν υποδοχέα σε πέντε τμήματα ανατολικό, δυτικό, βόρειο, νότιο, κεντρικό και σε αυτά τα τμήματα τοποθετεί τα συστατικά. Η διάταξη περιγράμματος υλοποιείται με την κλάση `BorderLayout` η οποία έχει τους παρακάτω δημιουργούς:

- `BorderLayout()`: Δημιουργεί μια διάταξη περιγράμματος.
- `BorderLayout(int y1, int y2)`: Δημιουργεί μια διάταξη περιγράμματος με `y1`, `y2` οριζόντιο και κατακόρυφο κενό ανάμεσα στα συστατικά.

Η μέθοδος `add(...)` σε αυτόν τον διαχειριστή αλλάζει κάπως και έχει σύνταξη `add(String str, component cm)`, όπου το όρισμα `str` μπορεί να πάρει τις τιμές `North`, `South`, `East`, `West`, `Center` για το βόρειο, το νότιο, το ανατολικό, το δυτικό και το κεντρικό τμήμα αντίστοιχα. Το όρισμα `cm` αντιπροσωπεύει το συστατικό που προστίθεται στον υποδοχέα.

Το παρακάτω παράδειγμα δημιουργεί μια διάταξη περιγράμματος με 10, 15 pixels οριζόντιο και κατακόρυφο διάκενο ανάμεσα στα συστατικά.




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

public class AppBorderLayout extends Applet {
    Button button1 = new Button("Button1");
```

```
    Button button2 = new Button("Button2");
    Button button3 = new Button("Button3");
    Button button4 = new Button("Button4");
    BorderLayout bl=new BorderLayout(5,5);

    public void init() {
     /* Το αντικείμενο button1 τοποθετείται στο βόρειο τμήμα, το button1 στο ανατολικό τμήμα,
     το button3 στο κέντρο, το button4 στο δυτικό τμήμα και το button5 στο νότιο τμήμα. */
        setLayout(bl);
        add("North",button1);
        add("East",button2);
        add("Center",button3);
        add("West",button4);
        add("South",new Button("Button5"));
    }

}
```

*Παράδειγμα 8.9 AppBorderLayout.java*

- source code

Έγγραφο html

```
<APPLET CODE="AppBorderLayout.class" WIDTH=300 HEIGHT=200>
</APPLET>
```

*appborderlayout.html*

---

### 8.3.2.3 Διάταξη πλέγματος (κλάση GridLayout)

---

Η ***διάταξη πλέγματος*** δημιουργεί ένα πλέγμα γραμμών και στηλών στο οποίο τα συστατικά εισάγονται από τα αριστερά προς τα δεξιά. Όταν γεμίσει μια σειρά του πλέγματος, τότε το επόμενο συστατικό προστίθεται στην επόμενη γραμμή. Η διαφορά με τους άλλους διαχειριστές διάταξης είναι ότι τα συστατικά που υπάρχουν στην διάταξη πλέγματος μεταβάλλουν το μέγεθος τους όταν αλλάζει μέγεθος ο υποδοχέας δηλαδή όταν μεγαλώνει το μέγεθος του παραθύρου (υποδοχέας) μεγαλώνει και το μέγεθος των συστατικών. Η κλάση που είναι υπεύθυνη για την διάταξη πλέγματος είναι η `GridLayout` η οποία έχει τους παρακάτω δημιουργούς:

- `GridLayout()`: Δημιουργεί μια στήλη για κάθε συστατικό σε μια απλή σειρά δηλαδή όλα τα συστατικά μπαίνουν στην ίδια σειρά και έχουν όλα το ίδιο πλάτος όσο μια στήλης. Στον συγκεκριμένο δημιουργό το πάχος μιας στήλης ισούται με το πλάτος του υποδοχέα δια τον αριθμό των συστατικών.
- `GridLayout(int x1, int x2)`: Δημιουργεί ένα πλέγμα γραμμών με `x1`, `x2` γραμμές και στήλες αντίστοιχα.

- `GridLayout(int x1, int x2, int y1, int y2)`: Δημιουργεί ένα πλέγμα γραμμών με `x1`, `x2` γραμμές και στήλες αντίστοιχα. Οι μεταβλητές `y1`, `y2` προσδιορίζουν το οριζόντιο και το κατακόρυφο διάκενο μεταξύ των συστατικών.

Όπως και στην διάταξη ροής (FlowLayout) έτσι και στην διάταξη πλέγματος παίζει ρόλο η σειρά με την οποία θα προστεθούν τα συστατικά στον υποδοχέα. Με την σειρά που θα προστεθούν στον υποδοχέα με αυτή την σειρά θα εμφανιστούν. Ένα παράδειγμα σχετικό με την διάταξη πλέγματος είναι το παρακάτω:

Πηγαίο αρχείο

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

public class AppGridLayout extends Applet {
    GridLayout gl = new GridLayout(4,4,5,5);
    Button button1 = new Button("Button1");
    Button button2 = new Button("Button2");
    TextField tf=new TextField("TextField");
    Label label=new Label("Label");
    Button button3 = new Button("Button3");
    Button button4 = new Button("Button4");
    Button button5 = new Button("Button5");
    Button button6 = new Button("Button6");
    Button button7 = new Button("Button7");
    Button button8 = new Button("Button8");
    Button button9 = new Button("Button9");
    TextArea ta=new TextArea("TextArea");

    public void init() {
        setLayout(gl);
        add(button1);
        add(button2);
        add(button3);
        add(tf);
        add(label);
        add(button4);
        add(button5);
        add(button6);
        add(button7);
        add(button8);
        add(button9);
        add(ta);
    }

}
```

*Παράδειγμα 8.10- AppGridLayout.java*

- source code

Έγγραφο html

```
<APPLET CODE="AppGridLayout.class" HEIGHT=200 WIDTH=300>
</APPLET>
```

*appgridlayout.html*

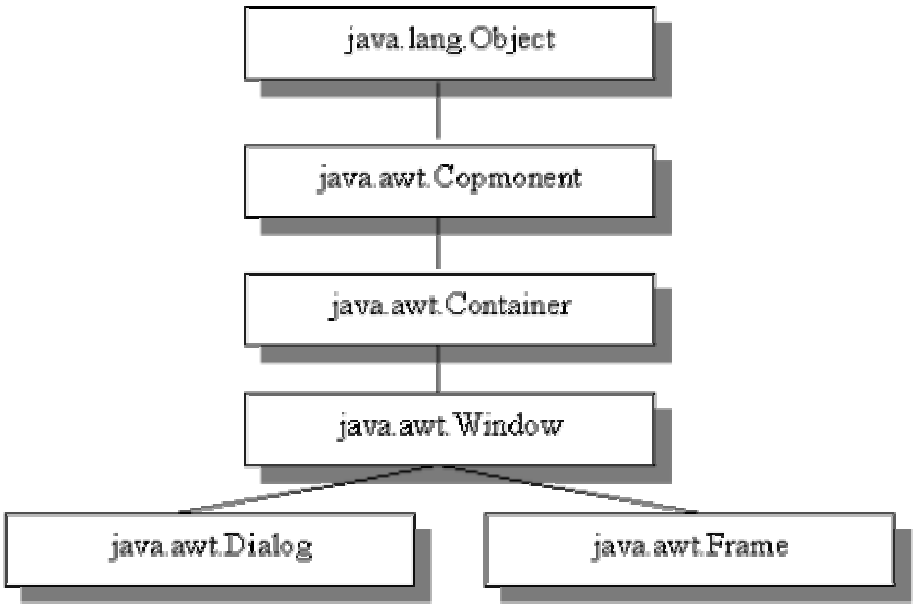
Στο παράδειγμα 8.10 η διάταξη πλέγματος (διαχειριστής διάταξης) δημιουργεί ένα πλέγμα τεσσάρων γραμμών και τεσσάρων στηλών. Επιπλέον ορίζει σε πέντε pixels το κατακόρυφο και οριζόντιο διάστημα μεταξύ των συστατικών. Στον υποδοχέα προστίθενται εννέα κουμπιά, ένα πεδίο κειμένου, μία ετικέτα και μια περιοχή κειμένου. Όλα τα συστατικά με αυτόν τον διαχειριστή διάταξης έχουν το ίδιο μέγεθος.

---

### 8.3.3 Υποδοχείς

---

Το υποκεφάλαιο αυτό αναφέρεται σε υποδοχείς που δημιουργούν παράθυρα και παράθυρα διαλόγου. Οι ιεραρχία των κλάσεων που δημιουργούν παράθυρα είναι:




*Σχήμα 8.1 – Ιεραρχία κλάσεων Dialog, Frame*

Η υπεύθυνη κλάση για την δημιουργία παράθυρων είναι η κλάση `Window` όπως φαίνεται στο σχήμα 8.1. Αντί όμως της κλάσης `Window` χρησιμοποιούνται οι δευτερεύουσες κλάσεις της, `Frame` και `Dialog`. Τα αντικείμενα των κλάσεων `Dialog` και `Frame` κληρονομούν της μεθόδους και τις μεταβλητές των κλάσεων `Window`, `Container`, `Component` και `Object`.

---

### 8.3.3.1 Πλαίσια (κλάση Frame)

---

Τα ***πλαίσια*** είναι κανονικά παράθυρα δηλαδή έχουν γραμμή τίτλου και όρια. Ακόμα έχουν πλαίσια κλεισίματος, λαβές αλλαγής μεγέθους και μπορούν να έχουν και μενού. Ο προεπιλεγμένος διαχειριστής διάταξης για τα πλαίσια είναι η διάταξη περιγράμματος (BorderLayout). Τα πλαίσια δημιουργούνται από την κλάση `Frame` η οποία ανήκει στο πακέτο `java.awt` και έχει του παρακάτω δημιουργούς:

- `Frame()`: Δημιουργεί ένα αντικείμενο της κλάσης Frame.
- `Frame(String str)`: Δημιουργεί ένα αντικείμενο της κλάσης `Frame` με τίτλο `str`.

Όταν δημιουργείται ένα παράθυρο αυτό είναι αόρατο. Για να γίνει ορατό χρησιμοποιείται η μέθοδος `setVisible(boolean bl)` που κληρονομείται από την κλάση `Component`. Αν το όρισμα `bl` είναι **true** το πλαίσιο είναι ορατό διαφορετικά είναι αόρατο. Το μέγεθος του πλαισίου σε pixels καθορίζεται από την μέθοδο `setSize(int x, int y)` της κλάσης `Component` όπου `x`, `y` το οριζόντιο και κατακόρυφο πλάτος του παραθύρου.

Στο παράδειγμα 8.11 η κλάση `AppFrame` δημιουργεί ένα `Frame` με τίτλο “New Frame”. Στο Frame προστίθενται 3 κουμπιά και ορίζεται το μέγεθος του σε 200, 200 pixels. Τέλος, το Frame γίνεται ορατό ώστε να είναι αντιληπτό κατά την εκτέλεση του προγράμματος.




```
import java.awt.*;

public class AppFrame{

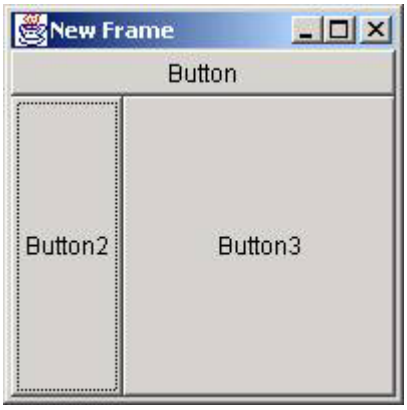
     // Δημιουργός της κλάσης AppFrame – αρχικοποίηση του προγράμματος
    public AppFrame(){
         Frame f=new Frame("New Frame");
         f.setSize(200,200);
         Button button=new Button("Button");
         Button button2=new Button("Button2");
         Button button3=new Button("Button3");
         f.add("West",button2);
         f.add("North",button);
         f.add("Center",button3);
         f.setVisible(true);
    }

    public static void main(String[] str){
         AppFrame appframe=new AppFrame();
    }

}
```

*Παράδειγμα 8.11-AppFrame.java*

Ακολουθεί ένα στιγμιότυπο της εφαρμογής `AppFrame.java`.




Το παραπάνω πρόγραμμα είναι εφαρμογή, σε αντίθεση με τα μέχρι στιγμης προγράμματα αυτού του κεφαλαίου που ήταν μικροεφαρμογές. Επιπλέον, περιέχει την μέθοδο `main(...)` δηλαδή πρώτα μεταγλωττίζεται από το εργαλείο **`javac`** και μετά διερμηνεύεται από το **`java`**.

Αν δοκιμάσετε να κλείσετε η να μεγιστοποιήσετε το `Frame`, θα δείτε ότι δεν θα υπάρχει κάποιο αποτέλεσμα. Αυτό είναι φυσιολογικό αφού δεν έχει οριστεί καμία αλληλεπίδραση με τον χρήστη. Για την αλληλεπίδραση υπάρχει αναφορά στο κεφάλαιο 8.4. Ένας τρόπος για να κλείσετε αυτό το παράθυρο είναι με το συνδυασμό των κουμπιών Ctrl+C.

Μια παραλλαγή του προγράμματος 8.11 είναι το 8.12 όπου η κλάση που δημιουργείται είναι δευτερεύουσα της κλάσης `Frame`. Ενώ στο 8.11 η μέθοδος `add(...)` προσδιορίζει το αντικείμενο της `Frame` (f.add(…)), επειδή αυτό ορίζεται μέσα στο σώμα του παραδείγματος, στο 8.12 υπάρχει σκέτη η μέθοδος `add(...)` γιατί είναι ήδη υποδοχέας η `AppFrame2` αφού κληρονομεί από τον υποδοχέα `Frame`. Το αποτέλεσμα των δυο προγραμμάτων είναι το ίδιο.

Πηγαίο αρχείο

```
import java.awt.*;

public class AppFrame2 extends Frame{

     // Δημιουργός της κλάσης AppFrame – αρχικοποίηση του προγράμματος
    public AppFrame2(){
         super("New Frame");
         setSize(200,200);
         Button button=new Button("Button");
         Button button2=new Button("Button2");
         Button button3=new Button("Button3");
         add("West",button2);
         add("North",button);
         add("Center",button3);
         setVisible(true);
    }

    public static void main(String[] str){
         AppFrame2 appframe=new AppFrame2();
    }

}
```

Στο παραπάνω παράδειγμα χρησιμοποιείται η πρόταση `super("New Frame")`. Αυτή η πρόταση είναι αναγκαία ώστε να τεθεί τίτλος στο πρόγραμμα διότι καλείται για να εκτελεστεί ο δημιουργός της `Frame` που παίρνει το όρισμα που υπάρχει στις παρενθέσεις.

---

### 8.3.3.2 Παράθυρα διαλόγου

---

Υπάρχουν δύο είδη παράθυρων διαλόγου. Το πρώτο είναι το παράθυρο διαλόγου της κλάσης `Dialog` και το δεύτερο της κλάσης `FileDialog`. Η κλάση `Dialog` δημιουργεί ένα πρωτογενές παράθυρο διαλόγου. Η κλάση `FileDialog` δημιουργεί ένα παράθυρο αναζήτησης αρχείων όπου μπορούν να αποθηκευτούν η να φορτωθούν αρχεία. Η `Dialog` είναι υπερκλάση της `FileDialog`. Τα παράθυρα διαλόγου μπορεί ορισθούν ως σε υποχρεωτικά. Αυτό σημαίνει ότι απαγορεύεται η επιλογή οποιονδήποτε άλλου παραθύρου του προγράμματος μέχρι αυτό να κλείσει. Επιπλέον, όταν τα παράθυρα δημιουργούνται είναι αόρατα. Για να γίνουν ορατά χρησιμοποιείται η μέθοδος `setVisible(boolean bl)` με το όρισμα `bl` να έχει την τιμή **true**.

---

#### 8.3.3.2.1 Dialog

---

Η κλάση `Dialog` δημιουργεί ένα παράθυρο διαλόγου που είναι οριοθετιμένο και μπορεί να έχει γραμμή τίτλου. Ένα παράθυρο διαλόγου `Dialog` από μόνο του δεν μπορεί να δημιουργήσει εφαρμογή, για αυτό θα πρέπει να είναι προσαρτημένο σε ένα πλαίσιο ή σε ένα άλλο `Dialog`. Αυτό σημαίνει, για παράδειγμα, πως όταν το πλαίσιο, από όπου εξαρτάται το παράθυρο διαλόγου ελαχιστοποιείται, ελαχιστοποιείται και το τελευταίο. Οι δημιουργοί του παράθυρου διαλόγου `Dialog` είναι:

- `Dialog(Dialog dialog)`: Δημιουργεί ένα παράθυρο διαλόγου προσαρτημένο στο παράθυρο `dialog`.
- `Dialog(Dialog dialog, String str)`: Δημιουργεί ένα παράθυρο διαλόγου προσαρτημένο στο παράθυρο dialog με τίτλο `str`.
- `Dialog(Dialog dialog, String str, boolean bl)`: Δημιουργεί ένα παράθυρο διαλόγου προσαρτημένο στο παράθυρο dialog με τίτλο `str`. Η μεταβλητή `bl` θέτει πότε το παράθυρο διαλόγου θα είναι υποχρεωτικό (true) και πότε όχι.
- `Dialog(Frame frame)`: Δημιουργείται ένα παράθυρο διαλόγου προσαρτημένο στο πλαίσιο `frame`.

- `Dialog(Frame frame, boolean bl)`: Δημιουργεί ένα παράθυρο διαλόγου προσαρτημένο στο πλαίσιο `frame`. Η μεταβλητή bl θέτει πότε το παράθυρο διαλόγου θα είναι υποχρεωτικό (true) και πότε όχι (false).
- `Dialog(Frame frame, String str)`: Δημιουργείται ένα παράθυρο διαλόγου προσαρτημένο στο πλαίσιο frame με δεδομένο τίτλο `str`.
- `Dialog(Frame frame, String str, boolean bl)`: Δημιουργείται ένα παράθυρο προσαρτημένο στο πλαίσιο frame με τίτλο `str`. Η μεταβλητή `bl` κάνει το παράθυρο υποχρεωτικό (true) ή όχι (false).

Αφού τα παράθυρα διαλόγου είναι προσαρτημένα σε κάποιο πλαίσιο ή σε άλλο παράθυρο διαλόγου πρέπει να υπάρχει μια αναφορά σε αυτά. Η αναφορά γίνεται με την μέθοδο `super(...)` η οποία καλεί τον δημιουργό της υπερκλάσης του παράθυρου διαλόγου.

Π.χ.

```
class AppDialog extends Dialog{

    AppDialog(Frame frame, String str, boolean bl){
        super(frame, str, bl);
        …
    }
    …
}
```

Για να έχει νόημα το παράθυρο διαλόγου απαιτεί την αλληλεπίδραση με τον χρήστη (π.χ πάτημα ενός κουμπιού σε ένα πλαίσιο και εμφάνιση του παραθύρου). Σχετικό παράδειγμα με παράθυρο διαλόγου υπάρχει στους χειριστές συμβάντων στο κεφάλαιο 8.4.6.7.

---

### 8.3.3.2.2 FileDialog

---

Η κλάση `FileDialog` εμφανίζει ένα παράθυρο διαλόγου από όπου ο χρήστης μπορεί να επιλέξει ένα αρχείο, δηλαδή μπορεί να το ανοίξει (open) ή να το αποθηκεύσει (save). Τα αντικείμενα της κλάσης `FileDialog` είναι υποχρεωτικά. Επιπλέον, πρέπει να είναι προσαρτημένα σε κάποιο πλαίσιο (Frame) δηλαδή δεν μπορούν να δημιουργήσουν εφαρμογή από μόνα τους. Οι δημιουργοί της κλάσης `FileDialog` είναι:

- `FileDialog(Frame frame)`: Δημιουργεί ένα παράθυρο διαλόγου προσαρτημένο στο πλαίσιο frame.
- `FileDialog(Frame frame, String str)`: Δημιουργεί ένα παράθυρο διαλόγου με τίτλο `str` προσαρτημένο στο πλαίσιο `frame`.

- `FileDialog(Frame frame, String str, int x)`: δημιουργεί ένα παράθυρο διαλόγου με τίτλο `str` προσαρτημένο στο πλαίσιο `frame`. Η μεταβλητή `x` μπορεί να πάρει τις τιμές `FileDialog.LOAD`, `FileDialog.SAVE` δηλαδή οι τιμές αυτές προσδιορίζουν την λειτουργία του παραθύρου διαλόγου για το άνοιγμα, αποθήκευση ενός αρχείου αντίστοιχα.

Δύο χρήσιμες μέθοδοι της κλάσης `FileDialog` είναι οι παρακάτω:

- `getDirectory()`: Επιστρέφει το όνομα της διαδρομής του φακέλου που υπάρχει το αρχείο που επιλέγεται.
- `getFile()`: Επιστρέφει το όνομα του αρχείου που επιλέγεται.

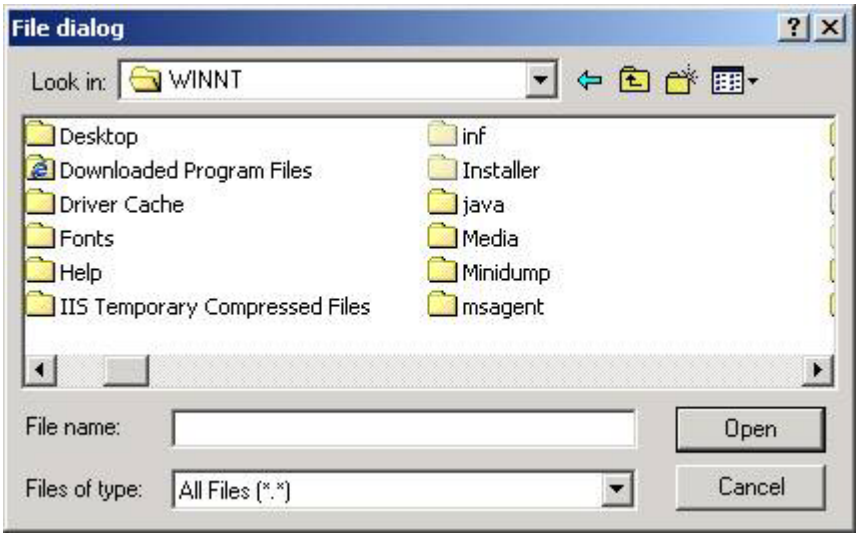
Όπως στην κλάση `Dialog` έτσι και στην `FileDialog` πρέπει να υπάρχει αναφορά στο πλαίσιο στο οποίο προσαρτάται.

Π.χ.

```
class AppFileDialog extends FileDialog{

    AppFileDialog(Frame frame, String str){
        super(frame, str);
        …
    }
    …
}
```

Ένα αντικείμενο της `FileDialog` έχει την παρακάτω εμφάνιση.




---

## 8.4 Χειρισμός συμβάντων (Event handler)

---

Στην συνέχεια (κεφάλαιο 8 –AWT), αναλύεται ο ***χειρισμός συμβάντων***. Ένα συμβάν (γεγονός) λαμβάνει χώρα όταν π.χ. ο χρήστης πληκτρολογεί κάτι, κάνει click ή εστιάζει ένα κουμπί μιας εφαρμογής. Κάθε αντικείμενο μπορεί να αλληλεπιδράσει με ένα γεγονός.

Όταν συμβαίνει ένα οποιοδήποτε γεγονός, δημιουργείται μια συγκεκριμένη κλάση ή περισσότερες κλάσεις που είναι δευτερεύουσες της κλάσης `AWTEvent`. Οι κλάσεις αυτές είναι μέρος του πακέτου `java.awt.event`. Αναφορά σε αυτές υπάρχει στο υποκεφάλαιο 8.4.2.

Για να αποκριθεί (απαντήσει) μια κλάση σε ένα γεγονός πρέπει να υλοποιήσει την διασύνδεση που χειρίζεται συμβάντα. Οι διασυνδέσεις που χειρίζονται συμβάντα λέγονται ***ακροατές συμβάντων*** (event listener). Αφού δημιουργηθεί ο ακροατής συμβάντων χρειάζεται μια μέθοδος που θα συσχετίσει ένα συστατικό (αντικείμενο) με τον συγκεκριμένο ακροατή. Οι μέθοδοι αυτοί αναφέρονται στο υποκεφάλαιο 8.4.4.

Στα κεφάλαια που ακολουθούν επεξηγούνται οι ακροατές συμβάντων, οι δευτερεύουσες κλάσεις της AWTEvent, οι μέθοδοι που συσχετίζουν τους ακροατές συμβάντων με τα συστατικά και οι μέθοδοι χειρισμού συμβάντων. Επίσης, υπάρχουν παραδείγματα για κάθε ακροατή συμβάντων.

---

### 8.4.1 Ακροατές συμβάντων (event listener)

---

Οι ***ακροατές συμβάντων*** υλοποιούν την διασύνδεση που χειρίζεται συμβάντα. Στον παρακάτω πίνακα συνοψίζονται οι ακροατές συμβάντων όταν πηγή συμβάντων θεωρηθεί ένα συστατικό.

| Συστατικό | Ακροατής συμβάντων | Χρήση του ακροατή συμβάντων |
|---|---|---|
| Οποιοδήποτε συστατικό | ComponentListener | Γεγονότα συστατικών που παράγονται. |
| | FocusListener | Χρησιμοποιείται για συμβάντα εστίασης πληκτρολογίου δηλαδή εστίασης ή μη. |
| | KeyListener | Χρησιμοποιείται για συμβάντα πληκτρολογίου. |
| | MouseListener | Χρησιμοποιείται για συμβάντα ποντικιού. |
| | MouseMotionListener | Χρησιμοποιείται για συμβάντα κίνησης ποντικιού |
| Υποδοχέας | ContainerListener | |
| Παράθυρο | WindowListener | Χρησιμοποιείται για συμβάντα παραθύρου. |
| Button | ActionListener | Χρησιμοποιείται για συμβάντα ενέργειας όπως το click σε ένα κουμπί. |
| List | | |
| TextField | | |

| | | |
|---|---|---|
| Choice | ItemListener | Χρησιμοποιείται για συμβάντα στοιχείων. |
| Checkbox | | |
| List | | |
| Scrollbar | AdjustmentListener | Χρησιμοποιείται για συμβάντα ρύθμισης. |
| TextArea | TextListener | Χρησιμοποιείται για συμβάντα κειμένου. |
| TextField | | |

*Πίνακας 8.1-Η λειτουργίες των ακροατών συμβάντων*

Στα αριστερά των ακροατών συμβάντων υπάρχουν τα συστατικά που μπορούν να χρησιμοποιηθούν με τον συγκεκριμένο ακροατή συμβάντων.

Μια κλάση μπορεί να έχει περισσότερους από έναν ακροατές συμβάντων.

Π.χ.

```
public class AppEventListener extends Applet implements
                                                ActionListener, WindowListener
{
      // σώμα κλάσης
}
```

Κάθε κλάση ακροατή συμβάντων έχει μεθόδους χειρισμού συμβάντων. Δηλαδή οι μέθοδοι αυτοί όταν γίνει ένα συμβάν (σχετικό με τον ακροατή συμβάντων τους) απαντούν σε αυτό. Οι μέθοδοί αυτοί καλούνται αυτόματα κάθε φορά που συμβαίνει ένα κατάλληλο γεγονός. Με αυτόν τον τρόπο επιτυγχάνεται η αλληλεπίδραση. Για αυτές τις μεθόδους υπάρχει αναφορά στο κεφάλαιο 8.4.5. Επιπλέον αυτές οι μέθοδοι έχουν ως όρισμα ένα αντικείμενο κλάσης που κληρονομεί την κλάση AWTEvent.

---

### 8.4.2 AWTEvent

---

Όπως ειπώθηκε παραπάνω οι κλάσεις που είναι υπεύθυνες για κάθε συμβάν είναι δευτερεύουσες της κλάσης `AWTEvent`. Δηλαδή όταν λαμβάνει χώρα ένα γεγονός ενεργοποιείται η αντίστοιχη κλάση. Οι κλάσεις αυτές περιέχουν στοιχεία για το γεγονότα π.χ. ποιο συστατικό δημιούργησε το συμβάν. Η ιεραρχία των δευτερεύουσων κλάσεων της `AWTEvent` υπάρχει στο διάγραμμα 2.

Οι κλάσεις αυτές μπορεί να είναι υπεύθυνες για περισσότερα από ένα γεγονότα. Π.χ. πατώντας ένα πλήκτρο δημιουργούνται τρία γεγονότα ένα για το πάτημα, ένα για την αποδέσμευση και ένα για το γράψιμο. Παρακάτω αναλύονται αυτές οι κλάσεις.

| Δευτερ. AWTEvent | Αιτία επίκλησης της κλάσης |
|---|---|
| ActionEvent | Δημιουργείται από ένα γεγονός συστατικού. |
| AdjustmentEvent | Δημιουργείται από ένα γεγονός ρύθμισης. |
| ComponentEvent | Δημιουργείται από ένα γεγονός κίνησης, δηλαδή αλλαγή μεγέθους και αλλαγή ορατότητας ενός συστατικού. |
| ContainerEvent | Δημιουργείται από το γεγονός της πρόσθεσης και της αφαίρεσης ενός συστατικού. |
| FocusEvent | Δημιουργείται από γεγονός εστίασης, δηλαδή της έλλειψης ή μη της εστίασης ενός συστατικού. |
| ItemEvent | Δημιουργείται από γεγονός στοιχείων, δηλαδή την επιλογή η μη ενός στοιχείου (Choice, Checkbox, List). |
| KeyEvent | Δημιουργείται από γεγονός πληκτρολογίου, δηλαδή το πάτημα, την αποδέσμευση και το γράψιμο ενός πλήκτρου. |
| MouseEvent | Δημιουργείται από συμβάν ποντικιού και κίνησης ποντικιού, δηλαδή την αποδέσμευση, το σύρσιμο, την είσοδο, την έξοδο και την κίνηση του ποντικιού σε ένα συστατικό. |
| TextEvent | Δημιουργείται από συμβάν κειμένου, δηλαδή με την αλλαγή κειμένου σε ένα συστατικό (TextField, TextArea) |
| WindowEvent | Δημιουργείται από συμβάν παραθύρου, δηλαδή το άνοιγμα, το κλείσιμο, την ενεργοποίηση, την απενεργοποίηση, την μεγιστοποίηση και την ελαχιστοποίηση ενός παραθύρου. |

*Πίνακας 8.2 -Δευτερεύουσες κλάσεις της AWTEvent*

Οι κλάσεις του πίνακα 8.2 είναι ορίσματα των μεθόδων χειρισμού συμβάντων των ακροατών συμβάντων. Μια χρήσιμη μέθοδος των παραπάνω κλάσεων είναι:

- getSource(): Επιστρέφει το αντικείμενο από το οποίο προήρθε το γεγονός. Κληρονομείται από την υπερκλάση `java.util.EventObject` της AWTEvent.

---

### 8.4.3 Μέθοδοι συσχέτισης

Όπως αναφέρθηκε στο κεφάλαιο 8.4, μετά τον ορισμό του ακροατή συμβάντων χρειάζεται και η μέθοδος η οποία θα συσχετίσει τον ακροατή συμβάντων με ένα συστατικό. Η μέθοδος αυτή είναι μια από:

- `addActionListener(ActionListener al)`: Προσθέτει τον ακροατή συμβάντων ActionListener σε ένα συστατικό για να πιάνει τα συμβάντα ενέργειας από αυτό. Το συστατικό σε αυτήν την περίπτωση μπορεί να είναι από τα `Button`, `TextField` και `List`.
- `addAdjustmentListener(AdjustmentListener al)`: Προσθέτει τον ακροατή συμβάντων AdjustmentListener σε ένα συστατικό Scrollbar ώστε να δεχθεί ένα αντικείμενο της `AdjustmentEvent` από αυτό το συστατικό.
- `addContainerListener(ContainerListener cl)`: Προσθέτει τον ακροατή συμβάντων ContainerListener σε έναν υποδοχέα ώστε να δεχθεί γεγονότα υποδοχέα από αυτόν (τον υποδοχέα).
- `addFocusListener(FocusListener fl)`: Προσθέτει τον ακροατή εστίασης FocusListener σε οποιοδήποτε συστατικό, ώστε να δεχθεί ένα γεγονός εστίασης από αυτό (το συστατικό).
- `addItemListener(ItemListener il)`: Προσθέτει τον ακροατή συμβάντων ItemListener σε ένα από τα αντικείμενα των κλάσεων `Choice`, `Checkbox`, `List`, ώστε να δεχθεί γεγονότα στοιχείων από αυτά.
- `addKeyListener(KeyListener kl)`: Προσθέτει τον ακροατή συμβάντων KeyListener σε ένα οποιοδήποτε συστατικό, ώστε να δεχθεί ένα γεγονός πληκτρολογίου από αυτό.
- `addMouseListener(MouseListener ml)`: Προσθέτει τον ακροατή συμβάντων MouseListener σε ένα οποιαδήποτε συστατικό, ώστε να δεχθεί ένα γεγονός ποντικιού από αυτό.
- `addMouseMotionListener(MouseMotionListener mml)`: Προσθέτει τον ακροατή συμβάντων MouseMotionListener σε ένα οποιαδήποτε συστατικό, ώστε να δεχθεί ένα γεγονός κίνησης ποντικιού από αυτό.
- `addTextListener(TextListener tl)`: Προσθέτει τον ακροατή συμβάντων TextListener σε ένα αντικείμενο των κλάσεων `TextField`, `TextArea`, ώστε να δεχθεί ένα γεγονός κειμένου από αυτά.
- `addWinodwListener(WindowListener wl)`: Προσθέτει τον ακροατή συμβάντων WindowListener σε ένα παράθυρο, ώστε να δεχθεί ένα γεγονός παραθύρου από αυτό (το παράθυρο).

### 8.4.4 Μέθοδοι χειρισμού συμβάντων

Οι ***μέθοδοι χειρισμού συμβάντων*** είναι μέθοδοι των ακροατών συμβάντων και περιέχουν τον κώδικα της απάντησης σε κάποιο συμβάν. Στον πίνακα 8.3 υπάρχουν οι μέθοδοι χειρισμού συμβάντων του κάθε ακροατή συμβάντων.

| Ακροατές συμβάντων | Μέθοδοι |
|---|---|

| | |
|---|---|
| ActionListener | actionPerformed(ActionEvent e) |
| AdjustmentListener | adjustmentValueChanged(AdjustmentEvent e) |
| ComponentListener | componentHidden(ComponentEvent e) |
| | componentMoved(ComponentEvent e) |
| | componentResized(ComponentEvent e) |
| | componentShow(ComponntEvent e) |
| ContainerListener | componentAdded(ContainerEvent e) |
| | componentRemoved(ContainerEvent e) |
| FocusListener | focusGained(FocusEvent e) |
| | focusLost(FocusEvent e) |
| ItemListener | itemStateChanged(ItemEvent e) |
| KeyListener | keyPressed(KeyEvent e) |
| | keyRealeased(keyEvent e) |
| | keyTyped(KeyEvent e) |
| MouseListener | mouseClicked(MouseEvent e) |
| | mouseEntered(MouseEvent e) |
| | mouseExited(MouseEvent e) |
| | mousePressed(MouseEvent e) |
| | mouseReleased(MouseEvent e) |
| MouseMotionListener | mouseDragged(MouseEvent e) |
| | mouseMoved(mouseEvent e) |
| TextListener | textValueChanged(TextEvent e) |
| WindowListener | windowActivated(WindowEvent e) |
| | windowClosed(WindowEvent e) |
| | windowClosing(WindowEvent e) |
| | windowDeactivated(WindowEvent e) |
| | windowDeiconified(WindowEvent e) |
| | windowIconified(WindowEvent e) |
| | windowOpened(WindowEvent e) |

*Πίνακας 8.3- Μέθοδοι ακροατών συμβάντων*

Παρατηρείται ότι οι ακροατές συμβάντων `ComponentListener`, `ContainerListener`, `FocusListener`, `KeyListener`, `MouseListener`, `MouseMotionListener` και `WindowListener` έχουν περισσότερες από μία μεθόδους χειρισμού συμβάντων. Καμιά από τις παραπάνω μεθόδους δεν επιστρέφει τιμή. Επειδή οι ακροατές συμβάντων είναι διασυνδέσεις και όχι κλάσεις, πρέπει κάθε κλάση που σχετίζεται με μια διασύνδεση να υλοποιεί όλες τις μεθόδους της.

Ανακεφαλαιώνοντας, όταν συμβεί ένα γεγονός οι πληροφορίες σχετικά με αυτό αποθηκεύονται σε ένα αντικείμενο που ανήκει σε υποκλάση της κλάση `AWTEvent`. Για να μπορέσει ο χρήστης να

επεξεργαστεί το γεγονός πρέπει να ορίσει έναν ακροατή συμβάντων για να ακούει τα συμβάντα. Επίσης, πρέπει να υλοποιηθούν οι μέθοδοι χειρισμού συμβάντων, οι οποίες είναι μέλη των κλάσεων ακροατών συμβάντων και δίνουν απάντηση σε ένα συμβάν. Τέλος, πρέπει να συμπεριληφθούν στον κώδικα οι μέθοδοι addXXXListener(…) που σχετίζουν τα συστατικά με τους ακροατές συμβάντων.

Μέχρι στιγμής διαβάσατε την θεωρία για τον χειρισμό συμβάντων το οποίο είναι και το πιο δύσκολο κομμάτι τους. Στην συνέχεια υπάρχουν παραδείγματα που διαφωτίζουν τις παραπάνω έννοιες.

---

### 8.4.5 Ανάλυση συμβάντων

---

#### 8.4.5.1 Συμβάντα ενέργειας

---

Τα συμβάντα ενέργειας υποδηλώνουν ότι ένα γεγονός συνέβη σε ένα από τα συστατικά `Button`, `TextField`, `List`. Οι πληροφορίες του συμβάντος, όπως π.χ. ποιο συστατικό προκάλεσε το συμβάν, αποθηκεύονται σε ένα αντικείμενο της κλάσης ActionEvent. Για τον χειρισμό των συμβάντων ενέργειας είναι απαραίτητος ο ακροατής συμβάντων ActionListener. Επιπλέον, πρέπει να προστεθεί η μέθοδος `addActionListener(ActionListener act)` σε κάθε συστατικό που παράγει ένα συστατικό ενέργειας, διαφορετικά το συμβάν του συστατικού αγνοείται. Δηλαδή, η μέθοδος αυτή συσχετίζει ένα αντικείμενο του ακροατή συμβάντων `ActionListener` με ένα συστατικό (Button, TextField, List).

Η απάντηση στο συμβάν ενός συστατικού γίνεται μέσω της μοναδικής μεθόδου της `ActionListener` που είναι η `actionPerformed(ActionEvent e)`, η οποία όπως φαίνεται από το όρισμα που έχει, συλλαμβάνει ένα αντικείμενο της κλάσης ActionEvent που δημιουργείται από ένα συμβάν ενέργειας.

Ανακεφαλαιώνοντας, για να μπορεί ένα πρόγραμμα να επεξεργαστεί οποιοδήποτε συμβάν ενέργειας, προϋποθέτει την μέθοδο `addActionListener(...)` σε κάθε συστατικό που μπορεί να παράγει ένα τέτοιο συμβάν. Επιπλέον, στην μέθοδο `actionPerformed(...)` προσδιορίζεται το συστατικό που προκάλεσε το συμβάν ενέργειας και επιλέγεται η απάντηση στο συμβάν.

Το πρόγραμμα 8.13 έχει δύο κουμπιά και ένα πεδίο κειμένου. Πατώντας το πρώτο κουμπί, το πεδίο κείμενο αποκτά το κείμενο “First button”, ενώ πατώντας το δεύτερο κουμπί, το κείμενο γίνεται “Second button”.

Αναλυτικότερα υλοποιείται η διασύνδεση ActionListener, η οποία ακούει τα γεγονότα που μπορεί να συμβούν σε ένα από τα κουμπιά, όπως π.χ. επιλογή τους από το ποντίκι. Η μέθοδος της `ActionListener`, actionPerformed(…) εκτελείται αυτόματα όταν συμβαίνει ένα γεγονός ενέργειας. Στην μέθοδο αυτή, με την συνθήκη ελέγχου **if**, εντοπίζεται ποιο κουμπί προκαλεί ένα συμβάν και η απάντηση σε αυτό είναι η εκτύπωση του κουμπιού αυτού στο πεδίο κειμένου. Όπως έχει ειπωθεί, η μέθοδος getSource() προσδιορίζει την πηγή του συμβάντος και στο συγκεκριμένο παράδειγμα ισούται με το αντικείμενο `button1` ή με το `button2`. Τέλος, υπάρχουν και οι μέθοδοι `addActionListener(...)` που συσχετίζουν τα δύο κουμπιά με τον ίδιο ακροατή συμβάντων (θα μπορούσαν και με διαφορετικό). Το όρισμα this των μεθόδων αυτών (addActionListener(this))

δηλώνει το τρέχον αντικείμενο της κλάσης `AppActionEvent` που στην περίπτωση αυτή υλοποιεί την διασύνδεση `ActionListener`.




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

public class AppActionEvent extends Applet implements ActionListener {
    Button button1;
    Button button2;
    TextField textfield;

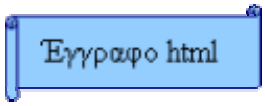
    public void init() {
        button1=new Button("button1");
        button2=new Button("button2");
        textfield=new TextField(20);
        add(button1);
        add(button2);
        add(textfield);
        button1.addActionListener(this);
        button2.addActionListener(this);
    }

    public void actionPerformed (ActionEvent e) {
        if(e.getSource()==button1)
            textfield.setText("First button");
        else if (e.getSource()==button2)
            textfield.setText("Second button");
    }

}
```

*Παράδειγμα 8.13 – AppActionEvent.java*

- source code




```
<APPLET CODE="AppActionEvent.class" WIDTH=300 HEIGHT=100>
</APPLET>
```

Μια παραλλαγή του παραπάνω προγράμματος με τα ίδια όμως αποτελέσματα είναι:




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

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

public class AppActionEvent2 extends Applet {
    Button button1;
    Button button2;
    TextField textfield;

    public void init() {
        button1=new Button("button1");
        button2=new Button("button2");
        textfield=new TextField(20);
        add(button1);
        add(button2);
        add(textfield);
        button1.addActionListener(new ActionListener(){
            public void actionPerformed (ActionEvent e) {
                    textfield.setText("first button");
        }});

        button2.addActionListener(new ActionListener(){
            public void actionPerformed (ActionEvent e) {
                    textfield.setText("Second button");
        }});
    }

}
```

*Παράδειγμα 8.14 –AppActionEvent2.java*

- source code

Στο παραπάνω παράδειγμα η κλάση `AppActionEvent2` δεν υλοποιεί την διασύνδεση `ActionListener`. Η κάθε μέθοδος `addActionListener(...)` έχει σαν όρισμα ένα αντικείμενο της κλάσης `ActionListener` το οποίο περιέχει και την μέθοδο του `actionPerformed(...)`. Δεν χρειάζεται η συνθήκη **if** στην μέθοδο `actionPerformed(...)` για την αναγνώριση του συστατικού που παρήγαγε το συμβάν ενέργειας, αφού κάθε συστατικό έχει τον δικό του ακροατή συμβάντων.

---

### 8.4.5.2 Συμβάντα ρύθμισης

---

Τα συμβάντα ρύθμισης υποδηλώνουν ότι ένα γεγονός ρύθμισης συνέβη σε ένα συστατικό `Scrollbar` (αντικείμενο της κλάσης `Scrollbar`). Η ***λίστα κύλισης*** (κλάση `Scrollbar`) είναι ένας μηχανισμός που επιτρέπει στον χρήστη να διαλέξει μέσα από ένα πεδίο τιμών. Μερικά από τα γεγονότα ρύθμισης είναι η μετακίνηση του συστατικού (αντικείμενο της Scrollbar) με τα βέλη του πληκτρολογίου, ή με το click του ποντικιού. Τα γεγονότα ρύθμισης δημιουργούν την κλάση AdjustmentEvent, η οποία περιέχει στοιχεία για το συμβάν. Για τον χειρισμό των συμβάντων ρύθμισης χρειάζεται ο ακροατής συμβάντων AdjustmentListener. Επιπλέον, πρέπει να προστεθεί η μέθοδος addAdjustmentListener(AdjustmentListener al), σε κάθε συστατικό Scrollbar διαφορετικά τα

συμβάντα ρύθμισης του Scrollbar αγνοούνται. Δηλαδή, η μέθοδος αυτή προσθέτει ένα αντικείμενο του ακροατή συμβάντων `AdjustmentListener` σε μια λίστα κύλισης .

Οι δημιουργοί για την μέθοδο `Scrollbar` είναι:

- `Scrollbar(int or)`: Δημιουργεί ένα αντικείμενο της κλάσης `Scrollbar` με διεύθυνση που δίνεται από την μεταβλητή `or`. Η `or` μπορεί να πάρει τις τιμές `Scrollbar.HORIZONTAL`, `Scrollbar.VERTICAL` για οριζόντια και κάθετη στοίχιση αντίστοιχα.
- `Scrollbar(int or, int value, int v, int min, int max)`: Δημιουργεί ένα αντικείμενο `Scrollbar` με διεύθυνση `or`, αρχική τιμή `value`, μικρότερη τιμή `min` και μέγιστη `max`. Η μεταβλητή `v` είναι προεπιλεγμένα μηδέν και προσδιορίζει το μέγεθος του πλαισίου που μετακινείται για να αλλάξει τιμή η λίστα κύλισης.

Στο ακόλουθο πρόγραμμα ορίζονται δύο λίστες κύλισης και ένα πεδίο κειμένου. Η προεπιλεγμένη τιμή για τις λίστες είναι η τιμή 50. Όταν μια από τις λίστες κύλισης αλλάζει τιμή, η άλλη λίστα θέτεται στην ίδια τιμή και το πεδίο κειμένου την τυπώνει. Αυτές οι διαδικασίες γίνονται στο σώμα της μεθόδου adjustmentValueChanged(…), η οποία είναι μέθοδος της διασύνδεσης AdjustmentListener. Την διασύνδεση αυτή υλοποιεί η κλάση `AppAdjustmentEvent`. Στην αρχή αναγνωρίζεται ποια λίστα κύλισης προκάλεσε το συμβάν ρύθμισης και μετά τίθεται το πεδίο κειμένου και στην άλλη λίστα κύλισης με την καινούργια τιμή. Επειδή η μέθοδος `setText(...)` δέχεται ένα **String** σαν όρισμα, χρησιμοποιείται η μέθοδος `valueOf(...)`, η οποία μετατρέπει την ακέραια τιμή της λίστας κύλισης σε τύπο **String**. Οι μέθοδοι `setValue(...)`, `getValue()` θέτουν και επιστρέφει την τρέχουσα τιμή της γραμμής κύλισης. Οι μέθοδοι `addAddjustmentListener(...)` συσχετίζουν τις λίστες κύλισης με την κλάση `AppAdjustmentEvent`, η οποία υλοποιεί την διασύνδεση `AdjustmentListener`.




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

public class AppAdjustmentEvent extends Applet implements AdjustmentListener{
    GridLayout gl;
    Scrollbar sb;
    Scrollbar sb2;
    TextField textfield;

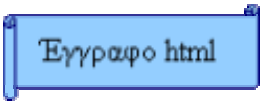
    public void init(){
        gl=new GridLayout(3,1,10,10);
        setLayout(gl);
        textfield=new TextField("50");
        textfield.setEditable(false);
        sb=new Scrollbar(Scrollbar.HORIZONTAL,50,0,1,100);
        sb2=new Scrollbar(Scrollbar.HORIZONTAL,50,0,1,100);
        add(sb);
        add(textfield);
        add(sb2);
        sb.addAdjustmentListener(this);
        sb2.addAdjustmentListener(this);
    }
```

```
    // χειρισμός συμβάντων ρύθμισης
    public void adjustmentValueChanged(AdjustmentEvent e){
        if(e.getSource()==sb)
        {
            textfield.setText(String.valueOf(sb.getValue()));
            sb2.setValue(sb.getValue());
        }
        else if(e.getSource()==sb2)
        {
            textfield.setText(String.valueOf(sb2.getValue()));
            sb.setValue(sb2.getValue());
        }
    }

}
```

*Παράδειγμα 8.15 – AppAdjustmentEvent.java*

- source code




```
<APPLET CODE="AppAdjustmentEvent.class" WIDTH=300 HEIGHT=100>
</APPLET>
```

*appadjustmentevent.html*

---

### 8.4.5.3 Συμβάντα στοιχείων

---

Τα συμβάντα στοιχείων υποδηλώνουν ότι ένα στοιχείο εκ των συστατικών `Choice`, `Checkbox` και `List` επιλέγεται η αποεπιλέγεται. Τα συμβάντα στοιχείων δημιουργούν την κλάση `ItemEvent` που περιέχει στοιχεία για το συμβάν. Για τον χειρισμό των συμβάντων στοιχείων χρειάζεται ο ακροατής συμβάντων ItemListener. Επιπλέον πρέπει να προστεθεί η μέθοδος `addItemListener(ItemListener il)`, σε κάθε συστατικό (`Choice`, `Checkbox`, `List`), διαφορετικά τα συμβάντα στοιχείων του συστατικού αγνοούνται. Δηλαδή, η μέθοδος αυτή συσχετίζει ένα αντικείμενο του ακροατή συμβάντων `ItemListener` με τα προαναφερθέντα συστατικά..

Η απάντηση σε ένα συμβάν στοιχείου δίνεται με την μέθοδο `itemStateChanged(ItemEvent e)` της διασύνδεσης `ItemListener`.

Στο ακόλουθο παράδειγμα ορίζεται μία λίστα επιλογής (κλάση Choice) με πέντε στοιχεία και ένα πεδίο κειμένου. Όταν ένα στοιχείο της λίστας επιλέγεται, λαμβάνει χώρα ένα συμβάν στοιχείου, πληροφορίες για το οποίο υπάρχουν σε ένα αντικείμενο της κλάσης `ItemEvent`, το οποίο υπάρχει σαν όρισμα στην μέθοδο itemStateChanged(…). Στο συγκεκριμένο παράδειγμα, το επιλεγμένο στοιχείο τυπώνεται στο πεδίο κειμένου. Αυτή η διαδικασία γίνεται στην μέθοδο `itemStateChanged(…)`.




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

public class AppItemEvent extends Applet implements ItemListener{
    Choice c;
    TextField textfield;

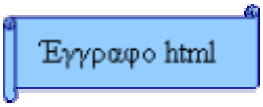
     // ορίζεται μια λίστα επιλογής με πέντε στοιχεία και ένα πεδίο κειμένου
    public void init(){
        c=new Choice();
        textfield=new TextField(20);
        c.addItemListener(this);
        c.add("First choice");
        c.add("Second choice");
        c.add("Third choice");
        c.add("Forth choice");
        c.add("Fifth choice");
        add(c);
        add(textfield);
    }

     // χειρισμός συμβάντων στοιχείου
    public void itemStateChanged(ItemEvent e){
        textfield.setText(c.getSelectedItem());
    }

}
```

*Παράδειγμα 8.16 – AppItemEvent.java*

- source code




```
<APPLET CODE="AppItemEvent.class" WIDTH=300 HEIGHT=100>
</APPLET>
```

*appitemevent.html*

---

### 8.4.5.4 Συμβάντα κειμένου

---

Τα συμβάντα κείμενου υποδηλώνουν την αλλαγή κατάστασης κειμένου των συστατικών TextField και TextArea. Τα συμβάντα κειμένου δημιουργούν την κλάση TextEvent η οποία περιέχει στοιχεία

για το συμβάν. Για τον χειρισμό των συμβάντων κειμένου χρησιμοποιείται ο ακροατής συμβάντων TextListener. Το επόμενο βήμα είναι η πρόσθεση της μεθόδου `addTextListener(TextLitsener tl)`, σε κάθε συστατικό (`TextField`, `TextArea`) διαφορετικά τα συμβάντα κειμένου τους, αγνοούνται.

Η απάντηση σε ένα συμβάν κειμένου δίνεται μέσω της μεθόδου `textValueChanged(TextEvent e)` της διασύνδεσης `TextListener`.

Στο ακόλουθο παράδειγμα ορίζεται μια περιοχή κειμένου και ένα πεδίο κειμένου των οποίων τα συμβάντα κειμένου αλληλεπιδρούν με τον χρήστη. Επιπλέον ορίζονται δύο πεδία κειμένου που εμφανίζουν την κατάσταση για τα δύο παραπάνω συστατικά. Όταν ένα γεγονός κειμένου συμβεί στο πεδίο κειμένου, τότε εμφανίζεται το μήνυμα “The textarea has been changed “ στο πεδίο κειμένου που βρίσκεται κάτω από αυτό. Το αντίστοιχο συμβαίνει και με το πρώτο πεδίο κειμένου, με την διαφορα ότι το μήνυμα που εμφανίζεται είναι “The textfield has been changed”. Το κουμπί `clear` καθαρίζει το περιεχόμενο των δύο πεδίων κειμένου, που δίνουν την αναφορά της κατάστασης. Η κλάση `AppTextEvent` υλοποιεί δύο διασυνδέσεις. Το κουμπί `clear` σχετίζεται με τον ακροατή συμβάντων ActionListener μέσω της μεθόδου `addActionListener(...)`, ενώ η περιοχή κειμένου και το πρώτο πεδίο κειμένου σχετίζεται με τον ακροατή συμβάντων `TextListener` μέσω της μεθόδου `addTextListener(...)`.




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

public class AppTextEvent extends Applet implements ActionListener,TextListener{
    TextArea textarea;
    TextField tf,tf1,tf2;
    Button button;

    public void init(){
        setLayout(new GridLayout(3,2,15,15));
        textarea=new TextArea("TextArea");
        tf=new TextField(20);
        tf1=new TextField(20);
        tf2=new TextField(20);
        tf1.setEditable(false);
        tf2.setEditable(false);
        button=new Button("Clear");
        tf.setText("TextField");
        textarea.addTextListener(this);
        tf.addTextListener(this);
        button.addActionListener(this);
        add(textarea);
        add(tf);
        add(tf1);
        add(tf2);
        add(button);
    }

    // χειρισμός συμβάντων κειμένου
    public void textValueChanged(TextEvent e){
        if(e.getSource()==textarea)
```

```
            tf1.setText("The textarea has been changed");
        else if (e.getSource()==tf)
            tf2.setText("The textfield has been changed");
    }

    // χειρισμός συμβάντων ενέργειας
    public void actionPerformed(ActionEvent e){
        tf1.setText("");
        tf2.setText("");
    }

}
```

*Παράδειγμα 8.17 – AppTextEvent.java*

- source code

Έγγραφο html

```
<APPLET CODE="AppTextEvent.class" WIDTH=400 HEIGHT=150>
</APPLET>
```

*apptextevent.html*

Τα γεγονότα που εξετάστηκαν μέχρι στιγμής έχουν ακροατές συμβάντων με μία μέθοδο. Επειδή οι ακροατές συμβάντων είναι διασυνδέσεις, οι κλάσεις που τις χρησιμοποιούν πρέπει να υλοποιούν όλες τις μεθόδους τους. Στην συνέχεια θα αναλυθούν γεγονότα που έχουν ακροατές συμβάντων που εμπεριέχουν περισσότερες από μια μεθόδους. Για να μην είναι αναγκασμένος ο προγραμματιστής να υλοποιεί όλες τις μεθόδους του ακροατή συμβάντων, αντί αυτού χρησιμοποιούνται κατάλληλες προσαρμοσμένες κλάσεις γεγονότων (event adapter class) στις οποίες μπορεί να οριστεί ακόμα και μια μέθοδο του ακροατή συμβάντων. Δηλαδή χρησιμοποιείται το όνομα της προσαρμοσμένης κλάσης αντί της κλάσης ακροατή συμβάντων αν είναι επιθυμητή η υλοποίηση λιγότερων από των συνολικών μεθόδων του ακροατή συμβάντων. Οι προσαρμοσμένες κλάσεις είναι αφηρημένες. Το όνομα των προσαρμοσμένων κλάσεων έχει ως δεύτερο συνθετικό την λέξη `Adapter` και ως πρώτο το όνομα του ακροατή συμβάντων χωρίς την κατάληξη `Listener`. Δηλαδή ο ακροατής συμβάντων `WindowListener` θα έχει ως προσαρμοσμένη κλάση την `WindowAdapter`.

---

### 8.4.5.5 Συμβάντα πλήκτρων

---

Τα συμβάντα πλήκτρων λαμβάνουν χώρα όταν ένα πλήκτρο πατιέται, αποδεσμεύεται, και τυπώνεται. Μπορούν να παραχθούν από οποιοδήποτε συστατικό. Ένα συστατικό δημιουργεί την κλάση `KeyEvent` όταν ένα γεγονός πλήκτρου συμβεί σε αυτό. Για τον χειρισμό των συμβάντων πλήκτρων χρησιμοποιείται η διασύνδεση του ακροατή συμβάντων KeyListener ή η προσαρμοσμένη κλάση

KeyAdapter. Τέλος πρέπει να προστεθεί η μέθοδος `addKeyListener(KeyEvent k)` σε κάθε συστατικό διαφορετικά τα συμβάντα πλήκτρου του συστατικού αγνοούνται.

Οι μέθοδοι του ακροατή συμβάντων `KeyListener` είναι:

- `KeyPressed(KeyEvent e)`: Η μέθοδος αυτή δίνει απάντηση όταν το συμβάν πλήκτρου είναι το πάτημα ενός πλήκτρου.
- `KeyReleased(KeyEvent e)`: Η μέθοδος αυτή απαντάει στην αποδέσμευση ενός πλήκτρου.
- `KeyTyped(KeyEvent e)`: Η μέθοδος αυτή απαντάει στην τύπωση ενός πλήκτρου.

---

### 8.4.5.6 Συμβάντα ποντικιού – κίνησης ποντικιού

---

Τα γεγονότα ποντικιού συμβαίνουν όταν το ποντίκι κάνει κλικ σε ένα συστατικό, εισέρχεται στην περιοχή ενός συστατικού, εξέρχεται από αυτή, επιλέγει ένα συστατικό και απελευθερώνει ένα συστατικό. Ένα συστατικό δημιουργεί την κλάση MouseEvent όταν ένα γεγονός ποντικιού συμβεί σε αυτό. Για τον χειρισμό των συμβάντων ποντικιού χρησιμοποιείται η διασύνδεση του ακροατή συμβάντων MouseListener ή η προσαρμοσμένη κλάση MouseAdapter. Χρησιμοποιείται η μέθοδος `addMouseListener(MouseListener m)` για να συσχετίσει τον ακροατή συμβάντων με το συστατικό που μπορεί να παράγει ένα συμβάν ποντικιού.

Η απάντηση στα γεγονότα ποντικιού δίνονται από τις παρακάτω μεθόδους της διασύνδεσης `MouseListener`.

- `mouseClicked(MouseEvent e)`: Δίνει απάντηση στο κλικ του ποντικιού σε ένα συστατικό
- `mouseEntered(MouseEvent e)`: Δίνει απάντηση για την είσοδο του ποντικιού στην περιοχή του συστατικού.
- `mouseExited(MouseEvent e)`: Δίνει απάντηση για την έξοδο του ποντικιού από την περιοχή του συστατικού.
- `mousePressed(MouseEvent e)`: Δίνει απάντηση για το πάτημα του ποντικιού σε ένα συστατικό.
- `mouseReleased(MouseEvent e)`: Δίνει απάντηση για την ελευθέρωση του ποντικιού σε ένα συστατικό.

Χρήσιμες μέθοδοι της κλάσης `MouseEvent` είναι η `getClickCount()` και η `getPonit()`. Η πρώτη επιστρέφει των αριθμό των κλικ πάνω σε ένα συστατικό και η δεύτερη επιστρέφει τις συντεταγμένες του σημείου που έκανε κλικ το ποντίκι.

Εκτός από τα γεγονότα ποντικιού υπάρχουν και τα γεγονότα κίνησης ποντικιού που συμβαίνουν όταν το ποντίκι κινείται ή σέρνεται σε ένα συστατικό. Ένα συστατικό δημιουργεί την κλάση `MouseEvent` όταν ένα γεγονός κίνησης ποντικιού συμβεί σε αυτό. Για τον χειρισμό των συμβάντων κίνησης ποντικιού χρησιμοποιείται η διασύνδεση του ακροατή συμβάντων MouseMotionListener ή η προσαρμοσμένη κλάση MouseMotionAdapter. Χρησιμοποιείται η μέθοδος `addMouseMotionListener(MouseMotionListener m)` για να συσχετίσει τον ακροατή συμβάντων `MouseMotionListener` με το συστατικό που μπορεί να παράγει ένα συμβάν κίνησης ποντικιού.

Οι απαντήσεις στα συμβάντα κίνησης ποντικιού δίνονται από τις παρακάτω μεθόδους της κλάσης `MouseMotionListener`.

- `mouseDragged(MouseEvent e)`: Όταν το ποντίκι σέρνεται, η μέθοδος αυτή απαντάει σε αυτό το γεγονός.
- `mouseMoved(MouseEvent e)`: Δίνει απάντηση στο συμβάν κίνησης του ποντικιού.

Στο ακόλουθο παράδειγμα ορίζονται δύο κουμπιά το `button2` και το `button1` σε ένα πλαίσιο. Στην επόμενη σειρά υπάρχουν δύο πεδία κειμένου που αντιστοιχούν στα `button2`, `button1`, δηλαδή δείχνουν σε ποια κατάσταση είναι τα κουμπιά ως προς τα συμβάντα ποντικιού. Στο `button2` έχουν οριστεί οι μέθοδοι της `MouseListener` ώστε να απαντάει όταν το ποντίκι εισέρχεται, φεύγει από την επιφάνεια του και όταν γίνεται απλό και διπλό κλικ πάνω σε αυτό. Αντίστοιχα το `button1` δίνει απάντηση όταν το ποντίκι εισέρχεται, φεύγει, πατιέται στην επιφάνειά του και απελευθερώνεται σε αυτή. Λόγω του ότι δεν χρησιμοποιούνται όλες οι μέθοδοι της διασύνδεσης `MouseListener`, στην θέση της εισέρχεται η προσαρμοσμένη κλάση `MouseAdapter`. Για την ταξινόμηση των συστατικών χρησιμοποιείται ως διαχειριστής διάταξης η διάταξη πλέγματος (κλάση GridLayout) που δημιουργεί ένα πλέγμα δύο γραμμών και δύο στηλών.

Πηγαίο αρχείο

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

public class AppMouse extends Applet {
    GridLayout gl;

    public void init(){
        gl=new GridLayout(2,2,20,20);
        setLayout(gl);
        Button button1=new Button("Button1");
        Button button2=new Button("Button2");
        final TextField tf=new TextField(20);
        final TextField tf1=new TextField(20);
        add(button2);
        add(button1);
        add(tf);
        add(tf1);
          /* χειρισμός συμβάντων ποντικιού όταν αυτό εισέρχεται (mouseEntered), εξέρχεται (mouseExited)
           από την επιφάνεια του κουμπιού και όταν κάνει μονό και διπλό κλικ στην επιφάνεια του (κουμπιού) */
        button2.addMouseListener(new MouseAdapter(){
            public void mouseEntered(MouseEvent e){
                tf.setText("Entered button2");}
            public void mouseExited(MouseEvent e){
                tf.setText("Exited button2");}
            public void mouseClicked(MouseEvent e){
                if(e.getClickCount()==1)
                    tf.setText("Click button2");
                if(e.getClickCount()==2)
                    tf.setText("Clicked button2 twice");}
```

```
        });
          /* χειρισμός συμβάντων ποντικιού όταν αυτό εισέρχεται (mouseEntered), εξέρχεται (MouseExited),
          πατείτε (mousePressed) και ελευθερώνεται (mouseReleased) από την επιφάνεια του button1 */
        button1.addMouseListener(new MouseAdapter(){
            public void mouseEntered(MouseEvent e){
                tf1.setText("Entered button1");}
            public void mouseExited(MouseEvent e){
                tf1.setText("Exited button1");}
            public void mousePressed(MouseEvent e){
                tf1.setText("Pressed button1 ");}
            public void mouseReleased(MouseEvent e){
                tf1.setText("Released button1");}
        });
    }

}
```

*Παράδειγμα 8.18 – AppMouse.java*

- source code

```
<APPLET CODE="AppMouse.class" WIDTH=300 HEIGHT=130>
</APPLET>
```

*appmouse.html*

---

### 8.4.5.7 Συμβάντα παραθύρων

---

Τα γεγονότα παραθύρων λαμβάνουν χώρα όταν ένα παράθυρο ανοίγει, κλείνει, ενεργοποιείται, απενεργοποιείται, ελαχιστοποιείται και μεγιστοποιείται. Ένα παράθυρο (δηλαδή δευτερεύουσες κλάσεις της Window) δημιουργεί την κλάση WindowEvent όταν συμβούν γεγονότα παραθύρου σε αυτό. Για τον χειρισμό των γεγονότων παραθύρων χρησιμοποιείται ο ακροατής συμβάντων WindowListener ή η προσαρμοσμένη κλάση WindowAdapter. Η μέθοδος `addWindowListener(WindowListener w)` προστίθεται σε ένα παράθυρο για να το συσχετίσει με τον ακροατή συμβάντων `WindowListener`.

Οι παρακάτω μέθοδοι του ακροατή `WindowListener` απαντούν στα συμβάντα παραθύρων.

- `windowActivated(WindowEvent e)`: Απαντάει όταν ενεργοποιείται το παράθυρο που συνδέεται με αυτή.
- `windowClosed(WindowEvent e)`: Καλείται αφού έχει κλείσει το παράθυρο που συνδέεται με αυτή.
- `windowClosing(WindowEvent e)`: Καλείται όταν κλείνει το παράθυρο.
- `windowDeactivated(WindowEvent e)`: Καλείται όταν απενεργοποιείται το παράθυρο που συνδέεται με αυτή
- `windowDeiconified(WindowEvent e)`: Καλείται στην περίπτωση ελαχιστοποίησης του παραθύρου.

- `windowIconified(WindowEvent e)`: Καλείται στην περίπτωση μεγιστοποίησης του παραθύρου.
- `windowOpened(WindowEvent e)`: Καλείται όταν ανοίγει το παράθυρο.

Το ακόλουθο παράδειγμα είναι εφαρμογή και ορίζει την κλάση `AppFrame3`, η οποία είναι δευτερεύουσα κλάση της Frame. Η `AppFrame3` είναι ένα πλαίσιο, το οποίο περιέχει τρία κουμπιά και ένα πεδίο κειμένου. Το ένα κουμπί έχει ετικέτα `Exit` και χρησιμοποιείται για την έξοδο του προγράμματος. Το κουμπί αυτό (`button2`) συσχετίζεται με τον ακροατή συμβάντων `ActionListener` για να τερματίσει την λειτουργία του πλαισίου. Ο κώδικας που κάνει αυτήν την ενέργεια είναι:

```
button2.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent e){
        setVisible(false);
        dispose();
        System.exit(0);
}});
```

Τρεις είναι οι προτάσεις για τον τερματισμό του πλαισίου. Η πρώτη (`setVisible(false)`) κάνει απλώς αόρατο το πλαίσιο. Η δεύτερη πρόταση ελευθερώνει τους πόρους του συστήματος. Η μέθοδος `dispose()` είναι μέλος της κλάσης `Window`. Η τελευταία πρόταση τερματίζει την λειτουργία της εικονικής μηχανής. Η μέθοδος `exit(...)` ανήκει στην κλάση `System` και το όρισμα μηδέν που έχει σημαίνει κανονικός τερματισμός.

Προεπιλεγμένα το πλαίσιο κλεισίματος ενός πλαισίου (το σημαδάκι x στην πάνω δεξιά πλευρά του πλαισίου) δεν κάνει τίποτα. Η λειτουργία που θέλουμε να επιτελεί είναι ίδια με αυτή του κουμπιού button2, η οποία γίνεται με τον κώδικά:

```
addWindowListener(new WindowAdapter(){
    public void windowClosing(WindowEvent e){
        setVisible(false);
        dispose();
        System.exit(0);
}});
```

Παρατηρούμε ότι η μέθοδος `addWindowListener(...)` δεν συσχετίζει κάποιο αντικείμενο με την προσαρμοσμένη κλάση `WindowAdapter`. Στην πραγματικότητα, σχετίζεται ένα αντικείμενο του πλαισίου `AppFrame3`, δηλαδή η παραπάνω μέθοδος μπορεί να γραφεί πιο κατανοητά `this.addWindowListener(...)`.

Στο υπόλοιπο πρόγραμμα, το κουμπί με ετικέτα "`Open dialog`" κάνει ορατό το παράθυρο διαλόγου (κλάση `Dialog`) που υπάρχει. Αυτό περιέχει ένα κουμπί για την έξοδο του και ένα πεδίο κειμένου. Το κουμπί εξόδου του παράθυρου διαλόγου απλώς το κάνει αόρατο χρησιμοποιώντας την μέθοδο `setVisible(false)`. Το παράθυρο διαλόγου είναι μη υποχρεωτικό (λόγω του ορίσματος false στον δημιουργό του) και ό,τι γράφεται στο πεδίο κειμένου του, αντιγράφεται στο πεδίο κειμένου του πλαισίου όταν γίνεται αόρατο. Ένας άλλος τρόπος για να γίνει το παράθυρο διαλόγου αόρατο είναι μέσω του κουμπιού με ετικέτα "Close Dialog". Η δήλωση του δημιουργού του παραθύρου διαλόγου έχει σαν πρώτο όρισμα την λέξη **this**, δηλαδή έχει σαν όρισμα το τρέχον `Frame`.




```
import java.awt.*;

import java.awt.event.*;

public class AppFrame3 extends Frame {
    Dialog dl;
    GridLayout gl,gl1;
    TextField tf;
    TextField tf1;

    public AppFrame3(){
        super("Frame");
        setSize(200,100);
        dl=new Dialog(this,"Dialog",false);
        dl.setSize(150,100);
        gl=new GridLayout(2,1,20,20);
        gl1=new GridLayout(2,2,20,20);
        dl.setLayout(gl);
        setLayout(gl1);
        Button button=new Button("Open Dialog");
        Button button2=new Button("Exit");
        Button button3=new Button("Close Dialog");
        Button button4=new Button("Exit");
        tf=new TextField(20);
        tf1=new TextField(20);
        dl.add(button4);
        dl.add(tf);
         /* χειρισμός συμβάντων ενέργειας του κουμπιού button4 (ετικέτα exit-ανήκει στο παράθυρο διαλόγου)
         που κάνει αόρατο το παράθυρο διαλόγου και τυπώνει το περιεχόμενο του πεδίου του κειμένου,
         στο πεδίο κειμένου του πλαισίου */
        button4.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e){
                tf1.setText(tf.getText());
                dl.setVisible(false);
       }});
         /* χειρισμός συμβάντων ενέργειας του κουμπιού button (ετικέτα open dialog) που κάνει
          ορατό το παράθυρο διαλόγου */
        button.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e)
                {dl.setVisible(true);}
        });
        add(button2);
        add(button);
        add(button3);
        add(tf1);
        button3.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e){
                dl.setVisible(false);
        }});
           /* χειρισμός συμβάντων ενέργειας του κουμπιού button2 (ετικέτα exit) που προκαλεί
           τον τερματισμό του πλαισίου */
        button2.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e){
                setVisible(false);
```

```
                dispose();
                System.exit(0);
        }});
          /* χειρισμός συμβάντων παραθύρου του πλαισίου (στο κλείσιμο του πλαισίου)
          που προκαλεί τον τερματισμό του */
        addWindowListener(new WindowAdapter(){
            public void windowClosing(WindowEvent e){
                setVisible(false);
                dispose();
                System.exit(0);
        }});
          /* χειρισμός συμβάντων παραθύρου του αντικειμένου της κλάσης Dialog που το κάνει αόρατο */
        dl.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                dl.setVisible(false);
            }
        });
    }

    public static void main(String[] str){
        AppFrame3 appframe=new AppFrame3();
        appframe.setVisible(true);
    }

}
```

*Παράδειγμα 8.19 – AppFrame3.java*

- source code

Ακολουθεί ένα στιγμιότυπο της εφαρμογής `AppFrame.java`.









## 9.1 Γενικά

Οι εφαρμογές που δημιουργούνται από ένα προγραμματιστή μπορούν ανά πασά στιγμή να καταρρεύσουν από κάποιο λάθος κατά την λειτουργία τους. Αυτές οι καταστάσεις συμβαίνουν από αμέλεια του προγραμματιστή ή από λάθος χειρισμό του χρήστη, και η `Java` τις ονομάζει ***εξαιρέσεις***.

Η ***εξαίρεση*** είναι ένα γεγονός που συμβαίνει κατά την εκτέλεση ενός προγράμματος και διακόπτει την ροή των εντολών του.

## 9.2 Εισαγωγικά

Στις πιο πολλές γλώσσες προγραμματισμού η διαδικασία ανίχνευσης και χειρισμού σφαλμάτων ενός προγράμματος κάνει τον κώδικα του δυσανάγνωστο και μεγάλο. Έστω ότι θέλουμε να διαβάσουμε ένα αρχείο και να βάλουμε τα περιεχόμενα του στην μνήμη. Ο ψευδοκώδικας αυτού του προγράμματος στις συνηθισμένες γλώσσες είναι:

```
readFile {
    open the file; // Άνοιγμα του αρχείου
    determine its size; // Καθορισμός του μεγέθους του
    allocate that much memory; // Καταμερισμός της μνήμης
    read the file into memory; // Ανάγνωση του αρχείου στην μνήμη
    close the file; // Κλείσιμο αρχείου
}
```

*Πρόγραμμα 9.1 – Πέρασμα δεδομένων στην μνήμη*

Ο παραπάνω κώδικας φαίνεται σωστός αλλά αγνοεί τα σφάλματα που μπορούν να συμβούν. Δηλαδή τι θα γίνει αν δεν ανοίγει το αρχείο, αν δεν μπορεί να καθοριστεί το μέγεθος του, αν δεν μπορεί να γίνει καταμερισμός μνήμης, αν δεν μπορεί να γίνει ανάγνωση του και αν δεν μπορεί να κλείσει το αρχείο. Για να χειρίζεται το παραπάνω πρόγραμμα αυτά τα σφάλματα πρέπει να προστεθεί παραπάνω κώδικας σε αυτό. Επομένως το πρόγραμμα 9.1 γίνεται:

```
errorCodeType readFile {
    initialize errorCode = 0;
    open the file;
    if (theFileIsOpen) {
        determine the length of the file;
            if (gotTheFileLength) {
                allocate that much memory;
                    if (gotEnoughMemory) {
                        read the file into memory;
                            if (readFailed) {
                                errorCode = -1;
                            }
                    } else {
                        errorCode = -2;
                    }
            } else {
                errorCode = -3;
            }
            close the file;
            if (theFileDidntClose && errorCode == 0) {
                errorCode = -4;
            } else {
                errorCode = errorCode and -4;
            }
    } else {
        errorCode = -5;
    }
    return errorCode;
}
```

*Πρόγραμμα 9.2 – Πέρασμα δεδομένων στην μνήμη και χειρισμός σφαλμάτων*

Γίνεται αντιληπτό ότι ο κώδικας του πρωτότυπου προγράμματος αυξήθηκε αρκετά. Επιπλέον το πρόγραμμα έγινε δυσανάγνωστο διότι οι εντολές σφαλμάτων αναμείχθηκαν με τις πρωτότυπες εντολές. Η λύση της Java στο πρόγραμμα 9.1, πάλι σε μορφή ψευδοκώδικα, είναι:

```
readFile {
    try {
        open the file;
        determine its size;
        allocate that much memory;
        read the file into memory;
        close the file;
    } catch (fileOpenFailed) {
        doSomething;
```

```
    } catch (sizeDeterminationFailed) {
        doSomething;
    } catch (memoryAllocationFailed) {
        doSomething;
    } catch (readFailed) {
        doSomething;
    } catch (fileCloseFailed) {
        doSomething;
    }
}
```

*Πρόγραμμα 9.3 – Χειρισμός σφαλμάτων μέσω Java*

Το πρόγραμμα 9.3 είναι ευανάγνωστο διότι ο κώδικας του είναι μαζεμένος στην αρχή και στο τέλος υπάρχει ο εντοπισμός και χειρισμός πιθανών σφαλμάτων. Ένα ακόμα πλεονέκτημα είναι ότι το μέγεθος του προγράμματος 9.3 είναι μικρότερο σε σχέση με το μέγεθος του 9.2.

---

## 9.3 Κλάσεις εξαιρέσεων

---

Για τις εξαιρέσεις η `Java` έχει δημιουργήσει ένα σύνολο κλάσεων που κληρονομούν από την κλάση `Throwable`. Δηλαδή η κλάση `Throwable` και όλες οι δευτερεύουσες κλάσεις αυτής είναι εξαιρέσεις που μπορούν να προκληθούν. Η κλάση `Throwable` έχει δύο υποκλάσεις, την `Error` και την `Exception`. Τα σφάλματα της κλάσης `Error` και των υποκλάσεων της είναι καταστροφικά για το τρέχον πρόγραμμα, γιατί ο προγραμματιστής δεν μπορεί να τα χειριστεί. Αντίθετα η κλάση `Exception` και οι υποκλάσεις της όταν προκληθούν μπορούν να διαχειριστούν από τον χρήστη. Η δευτερεύουσες κλάσεις της `Exception` χωρίζονται σε δύο κατηγορίες. Η πρώτη κατηγορία περιλαμβάνει τις εξαιρέσεις χρόνου εκτέλεσης (δευτερεύουσες κλάσεις της `RuntimeException`) και η δεύτερη όλες τις υπόλοιπες κλάσεις, μερικές από τις οποίες φαίνονται στο διάγραμμα 5. Οι εξαιρέσεις χρόνου εκτέλεσης συμβαίνουν επειδή ο κώδικας δεν είναι πολύ ακριβής. Π.χ. η κλάση `ArrayIndexOutOfBoundsException` (δευτερεύουσα της RuntimeException) προκαλείται όταν ένας πίνακας βγαίνει έξω από τα όρια του. Αυτό το λάθος μπορεί εύκολα να διορθωθεί. Ορισμένες άλλες δευτερεύουσες κλάσεις της `Exception` είναι: η κλάση `IOException`, η οποία προκαλείται από την διακοπή ή την αποτυχία μιας λειτουργίας εισόδου εξόδου. Μια δευτερεύουσά της κλάση, η `EOFException`, συμβαίνει όταν ένα αρχείο προς ανάγνωση τελειώνει πριν το προσδοκώμενο τέλος του.

Οι κλάσεις εξαιρέσεων έχουν κυρίως δύο δημιουργούς, έναν χωρίς όρισμα και έναν με όρισμα ένα **String**. Το **String** είναι το καθορισμένο μήνυμα που προβάλεται όταν συμβεί η εξαίρεση. Οι μέθοδοι της κλάσης `Throwable` κληρονομούνται στις δευτερεύουσες κλάσεις της. Σημαντικές μέθοδοί της είναι:

- `getMessage()`: Αυτή η μέθοδος επιστρέφει ένα **String** με το σφάλμα που προκάλεσε την εξαίρεση.

- `toString()`: Αυτή η μέθοδος επιστρέφει το όνομα της εξαίρεσης που προκλήθηκε καθώς και το σφάλμα που έγινε.

Ιδιαίτερη σημασία πρέπει να δοθεί σε ορισμένες λέξεις. Προκαλείται μια εξαίρεση σημαίνει ότι έγινε ένα σφάλμα κατά την εκτέλεση του προγράμματος. Η σύλληψη μιας εξαίρεσης σημαίνει αναγνώριση της εξαίρεσης και χειρισμός της ώστε να μην καταρρεύσει το πρόγραμμα.

---

# 9.4 Σύλληψη εξαιρέσεων

---

Η σύλληψη εξαιρέσεων, όπως αναλύθηκε περιληπτικά πιο πάνω, περιλαμβάνει δύο βήματα, το πιάσιμο και τον χειρισμό της. Ο χειρισμός της μπορεί να μην περιλαμβάνει κάποια εντολή. Για αυτά τα δύο βήματα χρησιμοποιούνται οι δεσμευμένες λέξεις **try**, **catch**, **finally** σε συνδυασμό, ή όλες μαζί, όπως θα αναλυθεί στην συνέχεια. Κατά την πρόκληση μιας εξαίρεσης γίνεται ανακοπή του προγράμματος. Η σύλληψη των εξαιρέσεων βοηθάει ώστε να συνεχισθεί η λειτουργία του προγράμματος, ακόμα και εάν προκληθούν εξαιρέσεις.

---

### 9.4.1 try μπλοκ

---

Το πρώτο βήμα στο χειρισμό μιας εξαίρεσης είναι να περιλαμβάνεται ο κώδικας που μπορεί να προκαλέσει εξαίρεση ανάμεσα στα μπλοκ της λέξη **try**, δηλαδή:

```
try {

// Κώδικας που μπορεί να προκαλέσει κάποια εξαίρεση

}
```

Το μπλοκ της **try** ορίζει το πεδίο που κάθε χειριστής εξαιρέσεων μπορεί να συνδεθεί με αυτό. Η δήλωση **try** πρέπει να συνοδεύεται τουλάχιστον από ένα μπλοκ **catch** ή ένα μπλοκ **finally**.

---

### 9.4.2 Μπλοκ catch

---

Το μπλοκ **catch** μπορεί να δηλωθεί μετά το μπλοκ **try** μία ή και περισσότερες φορές όπως στην παρακάτω σύνταξη.

```
try{
…
} catch( ){
…
} catch( ){
…
}
```

Πιο αναλυτικά η σύνταξη ενός μπλοκ **catch** είναι:

```
catch(ένα αντικείμενο μιας κλάσης εξαίρεσης){

...

}
```

Το αντικείμενο της κλάσης εξαίρεσης μπορεί να είναι οποιοδήποτε ομότυπο των υποκλάσεων της κλάσης `Throwable`. Στο μπλοκ, κάθε **catch** γίνεται ο χειρισμός της εξαίρεσης που δηλώνεται στις παρενθέσεις της δήλωσης catch. Δηλαδή, κάθε κλάση εξαίρεσης έχει μεθόδους που μπορούν να δώσουν πληροφορίες για τους λόγους που προκλήθηκε ένα σφάλμα.

Αναλυτικότερα, όταν προκληθεί μια εξαίρεση μέσα στο μπλοκ **try**, τότε οι υπόλοιπες εντολές του δεν εκτελούνται και ο έλεγχος περνάει στο μπλοκ **catch** που έχει στις παρενθέσεις του την κατάλληλη κλάση που προκλήθηκε.

---

### 9.4.3 Μπλοκ finally

---

Το **finally** μπλοκ χρησιμοποιείται με την παρακάτω σύνταξη.

```
try{
…
}
catch(...){
...
}
finaly{
```

…
}

Στην παραπάνω σύνταξή μπορούν να υπάρχουν περισσότερα από ένα μπλοκ **catch** ή και κανένα μπλοκ **catch**. Ο κώδικας του μπλοκ **finally** εκτελείται είτε προκληθεί εξαίρεση, είτε δεν προκληθεί. Αν δεν προκληθεί εξαίρεση εκτελείται ο κώδικας του μπλοκ **try** και στην συνέχεια ο κώδικας του μπλοκ **finally**. Αν προκληθεί εξαίρεση μετά το μπλοκ **catch**, εκτελείται ο κώδικας του μπλοκ **finally**. Όπως αναφέρθηκε παραπάνω, μπορεί να υπάρχει σύνταξη του μπλοκ **finally** χωρίς κανένα μπλοκ **catch**. Σε αυτήν την περίπτωση δεν έχουμε σύλληψη των εξαιρέσεων, αλλά το πρόγραμμα πριν τερματιστεί βιαία εκτελεί τον κώδικα του μπλοκ **finally**.

Συμπερασματικά αν υπάρχει κάποιος κώδικας που πρέπει να εκτελεστεί οπωσδήποτε, είτε προκληθεί εξαίρεση είτε όχι αυτός τοποθετείται στο μπλοκ **finally**. Έτσι, αντί να γραφεί ο κώδικας δύο φορές, δηλαδή στο μπλοκ **try** και στο μπλοκ **catch**, συντάσσεται κατευθείαν στο μπλοκ **finally**. Διευκρινιστικό παράδειγμα υπάρχει σε επόμενο κεφάλαιο.

---

## 9.5 Η λέξη κλειδί throws

---

Η λέξη κλειδί **throws** δηλώνει ότι ο κώδικας που υπάρχει στο σώμα μιας μεθόδου μπορεί να προκαλέσει μια ή περισσότερες εξαιρέσεις. Η σύνταξη της είναι:

```
… methodos(…) throws Exception1,…ExceptionN{
…
}
```

Η λέξη κλειδί **throws** δεν σημαίνει ότι θα προκληθεί οπωσδήποτε εξαίρεση αλλά μόνο αν χρειασθεί. Η λέξη **throws** παρέχει χρήσιμες πληροφορίες για τις εξαιρέσεις που μπορεί να προκαλέσει μια μέθοδος. Επιπρόσθετα, δίνει την δυνατότητα στην `Java` να επιβεβαιώσει τον σωστό χειρισμό της.

Όταν μια μέθοδος έχει στο σώμα της μια μέθοδο που δηλώνει ποιες εξαιρέσεις μπορεί να προκαλέσει, τότε πρέπει να γίνει σύλληψη των εξαιρέσεων αυτών. Μια άλλη περίπτωση είναι να μην επιθυμείται η σύλληψη των εξαιρέσεων της στο σώμα μιας άλλης μεθόδου, αλλά ο χειρισμός τους σε αυτήν. Τα παραπάνω αναλύονται στα παραδείγματα που ακολουθούν:

Π.χ.

…

```
public void methodos() throws IOException{
…
}
public void methos(){
try{
    methodos();
}
catch(IOException e)
{…}
```

---

```
…
public void methodos() throws IOException{
…
}
public void methos()throws IOException{
    methodos();
}
```

Στο πρώτο παράδειγμα η μέθοδος `methodos()` δηλώνει ότι μπορεί να προκαλέσει την εξαίρεση `IOException`. Στην συνέχεια, η μέθοδος `methos()` καλεί την μέθοδο `methodos()` και συλλαμβάνει την εξαίρεση, αφού περιλαμβάνει με τα μπλοκ **try** και **catch** την μέθοδο `methodos()`. Στο δεύτερο παράδειγμα η μέθοδος `methdodos()` μπορεί να προκαλέσει την εξαίρεση `IOException`. Η μέθοδος `methos()` καλεί την `methodos()`, αλλά επειδή δεν θέλει να πιάσει και να χειριστεί την εξαίρεση `IOException` μετά την δήλωση της τοποθετείται η λέξη κλειδί **throws** με το όνομα της εξαίρεσης. Με αυτόν τον τρόπο ο χειρισμός της εξαίρεσης δίνεται στην μέθοδο `methodos()`.

---

# 9.6 Παράδειγμα εξαιρέσεων

---

Στο παράδειγμα 9.6 ορίζεται ένα αντικείμενο της κλάση `FileReader` (ανήκει στο πακέτο java.io) η οποία ανοίγει αρχεία για ανάγνωση. Αν το αρχείο προς ανάγνωση δεν βρεθεί τότε η `FileReader` προκαλεί την εξαίρεση `FileNotFoundException`. Ο δημιουργός της `FileReader(…)` είναι: "public FileReader(String str) throws FileNotFoundException" δηλαδή μπορεί να προκαλέσει εξαίρεση. Ανεξάρτητα από το αν προκληθεί εξαίρεση το περιεχόμενο του μπλοκ **finally** εκτελείται δηλαδή προβάλεται στην Dos-Prompt το μήνυμα "TELOS". Η μέθοδος toString() κληρονομείται από την κλάση Throwable.



```
import java.io.*;


public class AppException {

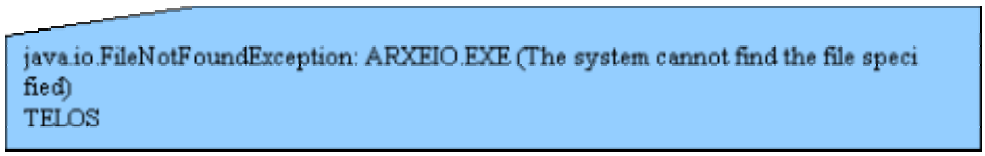
    public static void main(String[] args) {
        try{
            FileReader fr=new FileReader("ARXEIO.EXE");
        }
        catch(FileNotFoundException e)
        {
            System.out.println(e.toString());
        }
        finally{
            System.out.println("TELOS");
        }
   }

}
```

*Παράδειγμα 9.6 – AppException.java*

- source code

Αν δεν υπάρχει το arxeio.exe προκαλείται εξαίρεση και η έξοδος είναι:




Γενικά, υπάρχει η δυνατότητα να χρησιμοποιηθεί σαν όρισμα στην `catch(...)` μια υπερκλάση της υπάρχουσας κλάσης. Δηλαδή, στο παράδειγμα 9.6 είναι σωστή και η σύνταξη `catch(IOException e)`, διότι η `IOException` είναι υπερκλάση της `FileNotFoundException`. Πάντως δεν συνιστάται η χρήση υπερκλάσεων σε υψηλή βαθμίδα, διότι μπορούν να πιάσουν εξαιρέσεις που δεν προβλέφθηκαν, άρα δεν υπάρχει χειρισμός για αυτές.

---

## 9.7 Δημιουργία εξαιρέσεων

---

Η δημιουργία νέων εξαιρέσεων είναι αρκετά εύκολη υπόθεση. Πρέπει η εξαίρεση που δημιουργείται να κληρονομήσει κάποια εξαίρεση από την ιεραρχία των εξαιρέσεων της `Java`. Οι υπερκλάσεις των καινούργιων κλάσεων δεν επιτρέπεται να ανήκουν στην ιεραρχία `Error`, αλλά μπορούν να ανήκουν σε οποιαδήποτε άλλη κλάση της ιεραρχίας `Throwable`.

Ο δημιουργός μιας νέας εξαίρεσης διαλέγει ποια από τις υπάρχουσες εξαίρεση ταιριάζει πιο πολύ με αυτή που θέλει να φτιάξει και κληρονομεί από αυτήν. Οι εξαιρέσεις συνήθως έχουν δύο δημιουργούς έναν χωρίς όρισμα και έναν με όρισμα ένα **String**. Επομένως η σύνταξη μια εξαίρεσης που κληρονομεί από την κλάση `IOException` θα είναι:

Π.χ.

```
public class NewIOException extends IOException{
    NewIOException(){…}

    NewIOException(String str){
        super(str);
        …
    }

}
```







# 10.1 Γενικά για τα streams

Για την αλληλεπίδραση ενός προγράμματος με ένα αρχείο, δηλαδή για την αποθήκευση ή την ανάγνωση δεδομένων από αυτό χρησιμοποιείται ένα σύστημα επικοινωνίας που καλείται ***streams***. Τα streams δεν είναι τίποτα άλλο από κανάλια μεταφοράς δεδομένων μεταξύ αρχείου και προγράμματος. Τα αρχεία για την αποθήκευση ή ανάγνωση δεδομένων μπορεί να είναι αποθηκευμένα οπουδήποτε και σε οποιοδήποτε μέσο.

Τα streams χωρίζονται σε streams εισόδου που χρησιμοποιούνται για την μεταφορά δεδομένων από ένα αρχείο σε ένα πρόγραμμα και streams εξόδου για την μεταφορά δεδομένων από ένα πρόγραμμα σε ένα αρχείο.

Οι κλάσεις των streams ανήκουν στο πακέτο `java.io`. Οι κλάσεις αυτές διαιρούνται σε δύο κατηγορίες, ανάλογα με τον τύπο δεδομένων που μεταφέρουν. Υπάρχουν τα ***byte streams*** και τα ***streams χαρακτήρων***. Τα byte streams μεταφέρουν ακεραίους με τιμές από 0 έως 255. Κάθε τύπος δεδομένων μπορεί να εκφραστεί με την μορφή bytes. Τα streams χαρακτήρων όπως μαρτυρεί και το όνομα τους χειρίζονται μόνο δεδομένα κειμένου. Το κείμενο αυτό θεωρείται ως ένα σύνολο χαρακτήρων Unicode. Στα κεφάλαια που ακολουθούν αναλύονται τα streams χαρακτήρων.

Ο τρόπος χρήσης των streams σε γενικές γραμμές είναι ο εξής: Αρχικά, καλείται ένα αντικείμενο του τύπου stream (δηλαδή byte streams ή streams χαρακτήρων) που σχετίζεται με το αρχείο που επιθυμούμε να επεξεργαστούμε. Μετά γίνεται ανάγνωση ή εγγραφή στο αρχείο, ανάλογα με το αν έχουμε streams εισόδου ή streams εξόδου με τις μεθόδους του αντικειμένου stream. Τέλος, καλείται η κατάλληλη μέθοδος για να σταματήσει την χρήση του stream.

# 10.2 Streams χαρακτήρων

### 10.2.1 Ανάγνωση κειμένου μέσω streams χαρακτήρων

Όπως αναφέρθηκε στην εισαγωγή, υπάρχουν τρία βήματα για την ανάγνωση ή εγγραφή σε αρχείο. Το πρώτο βήμα είναι η συσχέτιση του προγράμματος με το αρχείο που επιθυμούμε, δηλαδή η δημιουργία του καναλιού μεταφοράς δεδομένων. Αυτό στην περίπτωση ανάγνωσης κειμένου μέσω streams χαρακτήρων γίνεται με την κλάση `FileReader`, της οποία η ιεραρχία βρίσκεται στο διάγραμμα 3. Η πιο σημαντική δημιουργός της `FileReader` είναι:

- `FileReader(String str)`: Το όρισμα `str` είναι το όνομα του αρχείου που θα συσχετισθεί με το τρέχον πρόγραμμα για ανάγνωση των δεδομένων του. Η `FileReader` συνοδεύεται από την δήλωση "throws FileNotFoundException" που σημαίνει ότι μπορεί να προκαλέσει την εξαίρεση `FileNotFoundException`. Αυτή η εξαίρεση θα προκληθεί αν δεν βρεθεί το αρχείο όρισμα της `FileReader`.

Δεύτερο βήμα είναι η ανάγνωση των δεδομένων του αρχείου. Υπάρχουν δύο μέθοδοι για αυτήν την ενέργεια που κληρονομούνται από την υπερκλάση `InputStreamReader` της `FileReader`.

- `read()`: Η μέθοδος αυτή επιστρέφει τον επόμενο χαρακτήρα του επιθυμητού αρχείου ως ακέραιο.
- `read(char[] ch, int a, int b)`: Η μέθοδος αυτή διαβάζει ένα πίνακα χαρακτήρων `ch` από τον `a` χαρακτήρα ως τον `b` ενός αρχείου. Επιστρέφει τον αριθμό των χαρακτήρων που διαβάστηκαν.

Και οι δύο παραπάνω μέθοδοι συνοδεύονται από την δήλωση "throws IOException" που σημαίνει ότι μπορούν να προκαλέσουν την εξαίρεση `IOException`. Επιπλέον επιστρέφουν τον αριθμό –1 αν το αρχείο φτάσει στο τέλος του. Ο αριθμός που επιστρέφει η μέθοδος `read()` αντιπροσωπεύει κάποιον χαρακτήρα στο σύστημα Unicode.

Το τρίτο βήμα είναι το κλείσιμο του `stream` το οποίο γίνεται με την μέθοδο `close()` που κληρονομείται από την κλάση `InputStreamReader`.

Στο παρακάτω παράδειγμα γίνεται ανάγνωση του αρχείου κείμενου `FileReader.txt` και εκτυπώνεται το περιεχόμενο του στην κονσόλα εξόδου. Χρησιμοποιείται ένας βρόχος, ώστε να σταματάει η ανάγνωση του αρχείου όταν φτάνει στο τέλος του, δηλαδή όταν ο ακέραιος που διαβάζεται είναι –1. Οι ακέραιοι που διαβάζει η μέθοδος `read()` μετατρέπονται σε χαρακτήρες με την πρόταση `c=(char)i` και οι χαρακτήρες σε **String** με την μέθοδο `valueOf(...)`. Το πρόγραμμα χρησιμοποιεί το μπλοκ `try...catch...` για την σύλληψη των πιθανών εξαιρέσεων. Στο συγκεκριμένο παράδειγμα θα προκληθεί εξαίρεση αν δεν βρεθεί το αρχείο κειμένου File.txt.




```
import java.io.*;


public class AppFileReader {

    public static void main(String[] args){
        try{
            FileReader fr=new FileReader("File.txt");
            int i=0;
            char c;
```

```
            String str;
            while(i!=-1){
                i=fr.read();
                if(i!=-1)
                {
                    c=(char)i;
                    System.out.print(String.valueOf(c));
                }
            }
            fr.close();
        }
        catch(IOException e)
        {
            System.out.println(e.toString());
        }
    }

}
```

*Παράδειγμα 10.1- AppFileReader.java*

- source code

---

### 10.2.2 Εγγραφή σε κειμένου μέσω streams χαρακτήρων

---

Για την εγγραφή σε αρχείο μέσω streams χαρακτήρων χρησιμοποιείται η κλάση `FileWriter`. Οι υπερκλάσεις της υπάρχουν στο διάγραμμα 4. Οι κυριότεροι δημιουργοί της `FileWriter` είναι:

- `FileWriter(String str)`: Το όρισμα `str` δηλώνει το αρχείο αποθήκευσης.
- `FileWriter(String str, boolean bl)`: Το όρισμα `str` δηλώνει το αρχείο αποθήκευσης. Αν το όρισμα `bl` είναι **true** τότε το κείμενο προσαρτάται στο είδη υπάρχον.

Και οι δύο δημιουργοί συνοδεύονται από την φράση "throws IOException" που σημαίνει ότι μπορούν να προκαλέσουν την εξαίρεση `IOException`.

Για την εγγραφή χαρακτήρων σε ένα αρχείο χρησιμοποιούνται οι παρακάτω μέθοδοι που κληρονομούνται από την κλάση `OutputStreamWriter`.

- `write(int x)`: Η μέθοδος αυτή αποθηκεύει ένα χαρακτήρα σε ένα αρχείο, πιο σωστά αποθηκεύει τον αριθμό Unicode του χαρακτήρα.
- `write(char[] ch, int x, int y)`: Γράφει τους χαρακτήρες του πίνακα `ch` από τον χαρακτήρα που βρίσκεται στην θέση `x` και `y` πλήθος ακόλουθων χαρακτήρων σε ένα αρχείο.
- `write(String str, int a, int b)`: Ίδια με την προηγούμενη μέθοδο με την διαφορά ότι αντί για πίνακας χαρακτήρων υπάρχει **String** χαρακτήρων.

Όπως και οι παραπάνω δημιουργοί, έτσι και αυτές οι μέθοδοι συνοδεύονται από την φράση “throws IOException” που σημαίνει ότι μπορούν να προκαλέσουν την μέθοδο `IOException`.

Επιπλέον υπάρχει η μέθοδος `close()` που κληρονομεί η `FileWriter` από την υπερκλάση της `OutputStreamWriter` και δηλώνει το τέλος της χρήσης του `stream`.

---

### 10.2.3. Buffers Streams

---

Ένας πιο αποδοτικός τρόπος χειρισμού των streams είναι μέσω των ***buffers streams***. Ένας ***buffer*** είναι μια περιοχή αποθήκευσης. Στην περίπτωση των streams εισόδου δημιουργείται ένας buffer streams εισόδου και τα δεδομένα διαβάζονται κατευθείαν από αυτόν.

Η διαδικασία ανάγνωσης ενός αρχείου με buffers streams είναι: Αρχικά δημιουργείται ένα αντικείμενο της κλάσης FileReader. Στην συνέχεια δημιουργείται ένα αντικείμενο buffer stream εισόδου της κλάσης `BufferReader`. Οι δημιουργοί της `BufferReader` είναι:

- `BufferedReader(Reader r)`: Το όρισμα `r` του δημιουργού είναι ένα αντικείμενο των υποκλάσεων της `Reader`, π.χ. όπως της κλάσης `FileReader`.
- `BufferedReader(Reader r, int a)`: Ισχύουν τα ίδια με την παραπάνω μέθοδο και επιπλέον ορίζεται το μέγεθος της buffer.

Ακολούθως, χρησιμοποιούνται οι μέθοδοι της κλάσης `BufferReader` για την ανάγνωση των χαρακτήρων ενός αρχείου. Οι μέθοδοι αυτοί είναι οι `read()`, `read(char[] ch, int x, int y)`, οι οποίες είναι όμοιες στην λειτουργία με τις αντίστοιχες μεθόδους της κλάσης `FileReader`. Μια ιδιαίτερα χρήσιμη μέθοδος είναι η `readline()` που επιστρέφει μια γραμμή κειμένου. Τέλος, υπάρχει και η μέθοδος `close()` για το κλείσιμο του stream.

Εκτός από το buffer stream εισόδου υπάρχει και το buffer stream εξόδου που χρησιμοποιείται για να γράψει ένα buffer σε ένα αρχείο. Η διαδικασία είναι όμοια με αυτήν τον buffer stream εισόδου. Δηλαδή, δημιουργείται ένα αντικείμενο της κλάσης FileWriter και ακολούθως ένα αντικείμενο της κλάσης buffer stream εξόδου `BufferWriter`. Οι δημιουργοί της `BufferWriter` είναι:

- `BufferedWriter(Writer w)`: Το όρισμα `w` μπορεί να είναι από οποιαδήποτε κλάση stream χαρακτήρων εξόδου, όπως η `FileWriter`.
- `BufferedWriter(Writer w, int x)`: Ισχύουν τα παραπάνω με μέγεθος buffer `x`.

Οι μέθοδοι της `BufferWriter` για την εγγραφή χαρακτήρων σε ένα αρχείο είναι οι `write(int x)`,`write(char[] ch, int x, iny y)`,`write(String str, int x, int y)` οι οποίες είναι όμοιες με τις αντίστοιχες μεθόδους της κλάσης `FileWriter`. Στο τέλος καλείται η μέθοδος `close()` για να κλείσει το stream.

Στο ακόλουθό παράδειγμα το κείμενο που υπάρχει στο αντικείμενο `str` αποθηκεύεται στο αρχείο new.txt. Για την αποδοτικότερη αποθήκευση του κειμένου χρησιμοποιείται ένας buffer stream εξόδου. Η μέθοδος που αποθηκεύει το κείμενο στο αρχείο είναι η `bw.write(str,0,str.length())`, οπού το δεύτερο όρισμα είναι το σημείο εκκίνησης του κειμένου και το τρίτο όρισμα το πλήθος των

χαρακτήρων μετά το σημείο εκκίνησης. Το όρισμα `str.length()` ισούται με το συνολικό πλήθος χαρακτήρων του `str`. Επομένως, με αυτόν τον τρόπο όλο το κείμενο του `str` γράφεται στο αρχείο new.txt.




```
import java.io.*;


public class AppFileWriter {

    public static void main(String[] args){
        try{
            String str="Το συγκεκριμένο πρόγραμμα \n"+
            " αποθηκεύει αυτό το κείμενο \n"+
            " στο αρχείο με τίτλο new.txt\n";
            FileWriter fw=new FileWriter("new.txt");
            BufferedWriter bw=new BufferedWriter(fw);
            bw.write(str,0,str.length());
            bw.close();
        }
        catch(IOException e)
        {
            System.out.println(e.toString());
        }
   }

}
```

*Πρόβλημα 10.2 – AppFileWriter.java*

- source code

---

# 10.3 Ερωτήσεις - Προβλήματα

---

Α) Να γίνει πρόγραμμα το οποίο θα διαβάζει ένα αρχείο κειμένου από τον σκληρό δίσκο και θα το εμφανίζει σε μία περιοχή κειμένου.

**Απαντήσεις**

Α) Στο ακόλουθο πρόγραμμα ορίζεται ένα πλαίσιο στο οποίο τοποθετούνται τρία κουμπιά και ένα πεδίο κειμένου. Το πρόγραμμα επιλέχθηκε να είναι εφαρμογή και όχι μικροεφαρμογή, διότι η δεύτερη στο συγκεκριμένο πρόγραμμα εμπίπτει στους κανόνες ασφάλειας των μικροεφαρμογών Στην

εφαρμογή ορίζεται ένα αντικείμενο της κλάσης FileDialog το οποίο διαλέγει το αρχείο που θα επιλεχθεί. Το αντικείμενο της `FileDialog` προσαρτάται στο υπάρχον πλαίσιο.

Οι λειτουργίες των τριών κουμπιών περιγράφονται στη συνέχεια. Το πρώτο με ετικέτα "Άνοιγμα αρχείου" χρησιμοποιείται για να κάνει ορατό το αντικείμενο της `FileDialog`. Στην συνέχεια αν επιλεχθεί ένα αρχείο, το κείμενο του γράφεται στην περιοχή κειμένου. Σε αυτό το σημείο εισέρχεται η λειτουργία του τρίτου κουμπιού. Όταν αυτό έχει τίτλο "Προσάρτηση κειμένου", τότε το κείμενο του αρχείου που επιλέγεται προσαρτάται στην αρχή της περιοχής κειμένου, αλλιώς αν έχει τίτλο "Μη προσάρτηση κειμένου" το νέο κείμενο αντικαθιστά το είδη υπάρχοντα. Η αλλαγή της λειτουργίας του τρίτου κουμπιού που συνοδεύεται και από την αλλαγή ετικέτας επιτυγχάνεται με το πάτημα του. Το δεύτερο κουμπί χρησιμοποιείται για το καθάρισμα της περιοχής κειμένου.

Η μέθοδος `read()` της κλάσης `BufferedReader` επιστρέφει τον Unicode ενός χαρακτήρα ο οποίος μετατρέπεται σε **String** με την πρόταση "str+=""+(char)i". Σε αυτήν την εντολή ο αριθμός Unicode μετατρέπεται πρώτα σε τύπο **char** και στην συνέχεια σε **String**, διότι ισχύει ότι η πρόσθεση ενός **String** με άλλον τύπο δίνει αποτέλεσμα τύπου **String**. Τέλος οι ακέραιοι `a`, `x` χρησιμοποιούνται ως σημαίες.




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

public class AppFileDialog{
    Button button1;
    Button button2;
    Button button3;
    TextArea textarea;
    Frame frame;
    FileDialog file;
    int a=0;
    int x=0;

    public AppFileDialog() {
        textarea=new TextArea(10,50);
        button1=new Button("Ανοιγμα αρχειου");
        button2=new Button("Καθαρισμα Textarea");
        button3=new Button(" Προσαρτηση κειμενου ");
        frame=new Frame("Application FileDialog");
        frame.setLayout(new FlowLayout());
        frame.setSize(400,240);
        file=new FileDialog(frame,"FileDialog");
        frame.add(button1);
        frame.add(button2);
        frame.add(button3);
        frame.add(textarea);
          /* Ενέργειες του button1. Γίνεται ορατό το παράθυρο διαλόγου και το αρχείο που θα επιλεγεί
          από τον χρήστη εμφανίζει το περιεχομενό του στην περιοχή κειμένου. */
        button1.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e){
                file.setVisible(true);
                try{
                    String str1=file.getDirectory()+file.getFile();
```

```
                FileReader fr=new FileReader(str1);
                BufferedReader br=new BufferedReader(fr);
                int i=0;
                char c;
                String str="";
                while(i!=-1){
                    i=br.read();
                    if(i!=-1)
                    {
                        str+=""+(char)i;
                    }
                }
                if (a==1)
                    textarea.setText(str);
                else if(a==0)
                    textarea.insert(str,0);
                br.close();
            }
            catch(IOException e1)
            {
                System.out.println("Please choose something file");
            }
    }});
      // Ενέργειες του button2. Καθαρίζει την περιοχή κειμένου.
    button2.addActionListener(new ActionListener(){
        public void actionPerformed(ActionEvent e){
            textarea.setText("");
    }});
      /* Ενέργειες του button3. Καθορίζει αν το κείμενο του αρχείου που επιλέγεται θα προσαρτηθεί
      στο είδη υπάρχον κείμενο της περιοχής κειμένου ή όχι. */
    button3.addActionListener(new ActionListener(){
        public void actionPerformed(ActionEvent e){
            if (x==0)
            {
                a=1;
                button3.setLabel("Μη προσαρτηση κειμενου");
                x=1;
            }
            else
            {
                a=0;
                button3.setLabel(" Προσαρτηση κειμενου ");
                x=0;
            }
    }});
      /* Κλείσιμο της εφαρμογής. Η πρώτη πρόταση στην μέθοδο windowClosing κάνει το πλαίσιο
      αόρατο. Η δεύτερη ελευθερώνει τους πόρους του συστήματος και η τρίτη τερματίζει
      την λειτουργία της εικονικής μηχανής */
    frame.addWindowListener(new WindowAdapter(){
        public void windowClosing(WindowEvent e){
            frame.setVisible(false);
            frame.dispose();
            System.exit(0);
    }});
    frame.setVisible(true);
}

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

```
        AppFileDialog appfiledialog=new AppFileDialog();
    }

}
```

*Παράδειγμα 10.3 – AppFileDialog.java*

- source code

# ΚΕΦΑΛΑΙΟ 11

# ΔΙΚΤΥΩΣΗ

## 11.1 Βασικά στοιχεία για τα δίκτυα

Οι υπολογιστές που επικοινωνούν στο διαδίκτυο χρησιμοποιούν τα πρωτόκολλα ***Transport Control Protocol*** (TCP) και ***User Datagram Protocol*** (UDP). Στην `Java` οι κλάσεις που είναι υπεύθυνες για αυτά τα πρωτόκολλα ανήκουν στο πακέτο `java.net`.

Το TCP είναι ένα πρωτόκολλο επικοινωνίας που παρέχει αξιόπιστη ροή δεδομένων ανάμεσα σε δύο συνδεδεμένους υπολογιστές. Είναι ανάλογο με το τηλέφωνο. Δηλαδή δύο χρήστες είναι συνδεδεμένοι και μπορούν να στέλνουν και να λαμβάνουν δεδομένα ο ένας από τον άλλο.

Τα πρωτόκολλα ***hypertext transfer protocol*** (HTTP – πρωτόκολλο μεταφοράς υπερκειμένου), ***file transfer protocol*** (FTP – πρωτόκολλο μεταφοράς αρχείων), χρησιμοποιούν το TCP διότι απαιτούν αξιοπιστία στην επικοινωνία.

Το UDP είναι ένα πρωτόκολλο που στέλνει ανεξάρτητα πακέτα δεδομένων, χωρίς βεβαιότητα για την άφιξη τους, δηλαδή πακέτα που μπορεί να χρησιμοποιούν διαφορετική διαδρομή για να φτάσουν στον παραλήπτη, τα οποία ονομάζονται datagrams. Μεταξύ των υπολογιστών που χρησιμοποιούν το UDP δεν υπάρχει επικοινωνία. Το UDP είναι ανάλογο του ταχυδρομείου. Δηλαδή, ο αποστολέας στέλνει πακέτα, τα οποία δεν είναι σίγουρο πως θα φτάσουν στον παραλήπτη και μπορεί να φτάσουν από διαφορετικές διαδρομές.

## 11.2 Αναφορά στα URL

Ένα συχνά χρησιμοποιούμενος όρος στο διαδίκτυο είναι τα αρχικά ***url*** που είναι το ακρωνύμιο των λέξεων uniform resource locator. Το url είναι μια αναφορά (μια διεύθυνση) σε μια πηγή στο Internet. Δηλαδή ο χρήστης παρέχει ένα url σε έναν εξεταστή και αυτός εντοπίζει το ανάλογο αρχείο στο διαδίκτυο. Το url είναι ανάλογο με την παροχή διεύθυνσης σε ένα γράμμα ώστε το ταχυδρομείο να μπορεί να εντοπίσει το παραλήπτη του γράμματος.

Ένα παράδειγμα ενός url είναι:

http://www.egnatia.ee.auth.gr/

Το παραπάνω url αποτελείται από δύο μέρη. Το πρώτο είναι ο ***προσδιοριστής πρωτοκόλλου*** (protocol identifier) που στη περίπτωση αυτή είναι το http (hypertext text protocol). Το ***http*** είναι πρωτόκολλο για την προσπέλαση υπερκειμένου, δηλαδή ιστοσελίδων. Άλλα συχνά χρησιμοποιούμενα πρωτόκολλα είναι το ***ftp*** (file transfer protocol), ***Gopher***, ***File*** και ***News***.

Το δεύτερό μέρος του url είναι το όνομα της ***πηγής*** που προσδιορίζει την διεύθυνση της πηγής. Στο συγκεκριμένο παράδειγμα το όνομα της πηγής είναι egnatia.ee.auth.gr. Το όνομα του πόρου εξαρτάται από το πρωτόκολλο που χρησιμοποιείται αλλά συνήθως περιλαμβάνει τα παρακάτω στοιχεία:

- Όνομα οικοδεσπότη (host name): Είναι το όνομα του υπολογιστή στον οποίο λειτουργεί η πηγή η οποία αναζητείται.
- Όνομα αρχείου (filename): Είναι η πλήρης διαδρομή στον κατάλογο που υπάρχει το επιθυμητό αρχείο στον κεντρικό υπολογιστή. Δηλαδή, στην διεύθυνση http://www.mywebsite.gr/allfile/file/index.html το όνομα αρχείου είναι allfile/file.
- Αριθμός θύρας (port number): Είναι η θύρα του κεντρικού υπολογιστή στον οποία αναζητείται ο πόρος. Αν παραλληλιστεί η διεύθυνση url με την διεύθυνση ενός γράμματος τότε το host name προσδιορίζει την οδό του παραλήπτη, ενώ η θύρα τον αριθμό της οδού. Για το πρωτόκολλο http η προεπιλεγμένη θύρα σύνδεσης είναι η 80 και συνήθως δεν αναφέρεται στο url. Δηλαδή τα ακόλουθα urls έχουν το ίδιο αποτέλεσμα http://www.egnatia.ee.auth.gr/ και http://www.egnatia.ee.auth.gr/.
- Αναφορά (reference): Είναι αναφορά σε συγκεκριμένο σημείο μιας σελίδας.

Για να αποκτήσουν πρόσβαση τα προγράμματα της `Java` σε μια πηγή στο διαδίκτυο χρησιμοποιούν url. Αυτό το επιτυγχάνουν καλώντας την `URL` κλάση της Java που ανήκει στο πακέτο java.net. Οι πιο σημαντικοί δημιουργοί της κλάσης `URL` είναι:

- `URL(String str)`: Δημιουργεί ένα αντικείμενο URL με διεύθυνση `str`.
- `URL(String prot, String host, String file)`: Δημιουργεί ένα αντικείμενο URL με πρωτόκολλο `prot`, με όνομα οικοδεσπότη `host` και όνομα αρχείου `file`.
- `URL(String prot, String host, int port, String file)`: Είναι ίδια με την παραπάνω και επιπλέον προσδιορίζεται η θύρα της διεύθυνσης από την μεταβλητή `port`.

Ακολούθως, μια διεύθυνση δίνεται και με τους τρεις δημιουργούς.

new URL(“http://www.egnatia.ee.auth.gr/allfiles/file/index.html”);

new URL(“http”,”www.egnatia.ee.auth.gr”,”allfiles/file/index.html”);

new URL(“http”,”www.egnatia.ee.auth.gr”,80,”allfiles/file/index.html”);

Χρήσιμοι μέθοδοι της κλάσης URL είναι:

- `getProtocol()`: Επιστρέφει τον προσδιοριστή πρωτοκόλλου ενός αντικειμένου `URL`.
- `getHost()`: Επιστρέφει το host name ενός αντικειμένου `URL`.

- `getPort()`: Επιστρέφει την θύρα ενός αντικειμένου `URL`.
- `getFile()`: Επιστρέφει το όνομα αρχείου ενός αντικειμένου `URL`.
- `openStream()`: Ανοίγει μια σύνδεση με ένα url και επιστρέφει ένα αντικείμενο `InputStream` για ανάγνωση από αυτή τη σύνδεση.

Ένα αντικείμενο `URL` μπορεί να προκαλέσει την εξαίρεση `MalformedURLException` αν το όρισμα (ή ορίσματα) που αναφέρεται στον δημιουργό δεν μπορεί να αναλυθεί ή υπάρχει σε αυτό δήλωση άγνωστου πρωτοκόλλου. Για το πιάσιμο και το χειρισμό της εξαίρεσης πρέπει να τοποθετηθεί ο δημιουργός `URL` ανάμεσα στο try…catch… μπλοκ όπως φαίνεται παρακάτω:

```
try{
    URL url=new URL(...)
}
catch (MalformedURLException e){
...
}
```

Στο ακόλουθο πρόγραμμα το πρώτο μπλοκ `try...catch...` τυπώνει τον προσδιοριστή πρωτοκόλλου, το host name και την θύρα της διεύθυνσης http://www.egnatia.auth.gr. Στο δεύτερο μπλοκ `try...catch...` η μέθοδος `openStream()` ανοίγει μια σύνδεση με την παραπάνω διεύθυνση και επιστρέφει ένα αντικείμενο `InputStream` το οποίο είναι stream εισόδου. Το περιεχόμενο του υπερκείμενου (δηλαδή τον κώδικα που δημιουργεί αυτήν την ιστοσελίδα) της διεύθυνσης http://www.egnatia.ee.auth.gr διαβάζεται από το πρόγραμμα μέσω του stream εισόδου. Προφανώς για να έχει αποτέλεσμα το stream εισόδου ο χρήστης πρέπει να είναι on-line. Οι μέθοδοι της κλάσης `InputStream` επιστρέφουν bytes, για αυτό χρησιμοποιείται ένα αντικείμενο της κλάσης `InputStreamReader` για να μετατρέψει τα bytes σε χαρακτήρες. Τέλος, οι χαρακτήρες αποθηκεύονται σε ένα buffer stream εισόδου (κλάση `BufferedReader`) και με την βοήθεια της μεθόδου `readLine()` τυπώνονται στην περιοχή κειμένου. Το παρακάτω πρόγραμμα, καθώς και τα άλλα δύο του κεφαλαίου χρησιμοποιούν την γραφική διασύνδεση `Swing` που αναλύεται στο κεφάλαιο 12. Επιπλέον και τα τρία προγράμματα είναι πλαίσια, δηλαδή στην δήλωση του προγράμματος περιλαμβάνουν την δήλωση “extends JFrame”. Η γραφική διασύνδεση του ακόλουθου προγράμματος αποτελείται από ένα πάνελ κύλισης πάνω στο οποίο υπάρχει μια περιοχή κειμένου.




```
import java.net.*;
import java.io.*;
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class AppUrl extends JFrame{
    JTextArea textarea;
    JScrollPane scroll;
```

```
 // Γραφική διασύνδεση του προγράμματος
public AppUrl(){
     super("Url example");
     getContentPane().setLayout(null);
     setSize(400,400);
     scroll=new JScrollPane();
     textarea=new JTextArea(4,80);
     scroll.getViewport().add(textarea);
     getContentPane().add(scroll);
     scroll.setBounds(1,1,390,370);
}

public void startProgram(){
     BufferedReader buf;
     URL url;
      // Ανάγνωση του πρωτοκόλλου, του host name και της θύρας της ιστοσελίδας
     try {
          url=new URL("http://www.egnatia.ee.auth:80");
          String prot=url.getProtocol();
          String host=url.getHost();
          int port=url.getPort();
          textarea.setText("Πρωτοκολλο:"+prot+" Διευθυνση:"+
                                        host+" Θυρα:"+port+"\n");
          System.out.println(prot+"://"+host+":"+port);
     }
     catch(MalformedURLException e){
     }
       // Ανάγνωση και τύπωση του υπερκειμένου της ιστοσελίδας
     try{
          url=new URL("http://www.egnatia.ee.auth.gr");
          buf=new BufferedReader(new InputStreamReader(url.openStream()));
          String input;
          while((input=buf.readLine())!=null)
               textarea.setText(textarea.getText()+"\n"+input);
          buf.close();
     }
     catch(MalformedURLException e1)
     {
          System.out.println("Το όρισμα του δημιουργου
                              Url δεν μπορεί να αναλυθεί");
     }
     catch(IOException e){System.out.println("Δεν
                     μπορεί να βρεθεί το url για να γίνει η
                     σύνδεση του το stream εισόδου");}
     }

       // Δημιουργία του προγράμματος και εμφάνιση του πλαισίου
     public static void main(String[] args)
     {
          AppUrl app=new AppUrl();
          WindowListener l = new WindowAdapter() {
               public void windowClosing(WindowEvent e) {
                    System.exit(0);
               }
          };
          app.addWindowListener(l);
          app.setVisible(true);
          app.startProgram();
```

```
        }
}
```

*Παράδειγμα 11.1 – AppUrl.java*

- source code

---

# 11.3 Sockets

---

Εκτός από την κλάση URL για πιο γενικές εφαρμογές δικτύου χρησιμοποιούνται οι κλάσεις sockets της Java. Γενικά ένα ***socket*** είναι το ένα άκρο μιάς διπλής κατεύθυνσης επικοινωνιακού συστήματος μεταξύ δύο προγραμμάτων που εκτελούνται στο δίκτυο. Οι κλάσεις sockets της `Java` αναπαριστούν την επικοινωνία ανάμεσα στον πελάτη (client) και στον διακομιστή (server – εξυπηρετητή).

Για την μεταξύ τους επικοινωνία τα sockets χρησιμοποιήσουν τα πρωτόκολλά TCP και UDP. Στο τρέχον εγχειρίδιο, δηλαδή στην συνέχεια του κεφαλαίου, εξετάζεται η επικοινωνία μέσω του πρώτου πρωτοκόλλου. Η ανάγνωση και η εγγραφή κειμένου στα sockets γίνεται με την βοήθεια των streams εισόδου και εξόδου.

Στο πακέτο `java.net` υπάρχουν οι κλάσεις για τα sockets. Αυτές είναι η `Socket` και η `ServerSocket` που υλοποιούν τα sockets του πελάτη και του διακομιστή αντίστοιχα.

Ιδιαίτερη προσοχή χρειάζεται στις λέξεις που είναι ίδιες αλλά έχουν διαφορετικό νόημα δηλαδή socket είναι ο ορισμός που ειπώθηκε παραπάνω ενώ `Socket` είναι μια κλάση μέσω της οποίας η Java προσπαθεί να πετύχει μια σύνδεση socket.

Η διαδικασία επικοινωνίας των υπολογιστών μέσω sockets περιγράφεται ακολούθως. Ο διακομιστής χρησιμοποιεί ένα αντικείμενο της κλάσης `ServerSocket` το οποίο περιμένει κάποιο αίτημα ενός αντικείμενου της κλάσης `Socket` του πελάτη. Ο πελάτης προσπαθεί να δημιουργήσει μια socket σύνδεση δημιουργώντας ένα αντικείμενο της κλάσης `Socket`. Αυτή η δημιουργία του αντικειμένου σημαίνει αίτηση για socket σύνδεσης στον διακομιστή. Ο διακομιστής δέχεται την αίτηση του πελάτη (μέσω της μεθόδου `accept(...)` της κλάσης `ServerSocket` της οποίας ο επιστρεφόμενος τύπος είναι Socket), απαντώντας με ένα αντικείμενο της κλάσης `Socket` μέσω του οποίου θα λαμβάνει χώρα η επικοινωνία στην πλευρά του διακομιστή. Επομένως, γίνεται φανερό ότι επιτυγχάνεται μια σύνδεση ενός αντικείμενου `Socket` της πλευράς του διακομιστή και ενός αντικειμένου `Socket` της πλευράς του πελάτη, δηλαδή υπάρχει επικοινωνία `Socket` με `Socket`. Η διαφορά των κλάσεων `Socket` και `ServerSocket` είναι ότι η δεύτερη μπορεί να δεχθεί μόνο καινούργιες αιτήσεις για σύνδεση και όχι πακέτα δεδομένων.

Οι σημαντικότεροι δημιουργοί των κλάσεων `Socket` και `ServerSocket` είναι:

- `Socket(String host, int port)`: Δημιουργεί ένα socket πελάτη και ανοίγει μια σύνδεση με τον διακομιστή (όρισμα host) στην θύρα port. Δηλαδή, το όρισμα host είναι η διεύθυνση του διακομιστή. Αυτός ο δημιουργός μπορεί να προκαλέσει τις εξαιρέσεις `IOException` και `UnknownHostException`. Η πρώτη προκαλείται από λάθος εισόδου εξόδου, ενώ η δεύτερη από λάθος host name.
- `ServerSocket(int port)`: Δημιουργεί ένα socket διακομιστή στην θύρα `port`. Δεν χρειάζεται ο προσδιορισμός της διεύθυνσης που βρίσκεται ένα αντικείμενο της `ServerSocket`, διότι ήδη προσδιορίζεται από τον υπολογιστή που βρίσκεται. Μπορεί να προκαλέσει την εξαίρεση `IOException`.

Σημαντικές μέθοδοι της κλάσης `Socket` είναι:

- `close()`: Χρησιμοποιείται για να κλείσει ένα αντικείμενο της κλάσης `Socket`. Η μέθοδος αυτή μπορεί να προκαλέσει την εξαίρεση `IOException`, αν συμβεί ένα λάθος εισόδου/εξόδου όταν κλείνει το `Socket`.
- `getInputStream()`: Επιστρέφει ένα stream εισόδου από το αντικείμενο της κλάσης `Socket` που προσδιορίζει. Ο τύπος επιστροφής της μεθόδου είναι `InputStream`. Και αυτή η μέθοδος μπορεί να προκαλέσει την εξαίρεση `IOException`.
- `getOutputStream()`: Επιστρέφει ένα stream εξόδου από το αντικείμενο της κλάσης `Socket` που προσδιορίζει. Ο τύπος επιστροφής της μεθόδου είναι `OutputStream`. Και αυτή η μέθοδος μπορεί να προκαλέσει την εξαίρεση `IOException`.

Αξιοσημείωτες μέθοδοι της κλάσης `ServerSocket` είναι:

- `accept()`: Κάνει δεκτή την αίτηση ενός socket πελάτη. Ο τύπος επιστροφής της είναι τύπου `Socket`. Η μέθοδος αυτή μπορεί να προκαλέσει τις εξαιρέσεις `IOException` και `SecurityException`.
- `close()`: Χρησιμοποιείται για να κλείσει ένα αντικείμενο της κλάσης `ServerSocket`. Και αυτή η μέθοδος μπορεί να προκαλέσει την εξαίρεση `IOException`.
- `getInetAddress()`: Επιστρέφει την διεύθυνση του `ServerSocket`.
- `getLocalPort()`: Επιστρέφει την θύρα στην οποία ο `ServerSocket` αναμένει τις αιτήσεις των πελατών.

Στην πλευρά του διακομιστή μια σύνδεση socket επιτυγχάνεται με τον πελάτη με τις παρακάτω εντολές:

```
ServerSocket server=new ServerSocket(8888);

Socket connection;

connection= server.accept();
```

Αρχικά δημιουργείται ένα αντικείμενο της κλάσης `ServerSocket` το οποίο αναμένει κάποια αίτηση ενός πελάτη (client). Αν ο πελάτης κάνει αίτηση, αυτή γίνεται δεκτή με την μέθοδο `accept()`. Το αντικείμενο connection της κλάσης `Socket` δεν προσδιορίζει ούτε την διεύθυνση,

ούτε την θύρα που θα συνδεθεί, δηλαδή φαίνεται ότι δεν περιέχει την διεύθυνση και την θύρα του πελάτη. Αυτές οι πληροφορίες δίνονται στο αντικείμενο connection από το αντικείμενο `Socket` που επιστρέφει η μέθοδος `accept()`.

Αντίστοιχα, ο πελάτης προσπαθεί να συνδεθεί με τον διακομιστή μέσω socket με τις παρακάτω εντολές:

```
Socket connection_client;

connection_client=new Socket("Address",port);
```

Αφού γίνει η σύνδεση διακομιστή πελάτη μέσω sockets, οι συνδεδεμένοι υπολογιστές μπορούν να πάρουν και να στείλουν πληροφορίες με την βοήθεια των streams. Ακολούθως δίνεται ένα παράδειγμα με streams.

Π.χ.

```
Socket one_socket;
try{
    one_socket=new Socket("address",8888);
    BufferedReader in;
    PrintWriter out;
    in=new BufferedReader(new
    InputStreamReader(one_socket.getInputStream()));
    out =new PrintWriter(one_socket.getOutputStream(),true);
}
catch(UnknownHostException e)
{
    System.out.println("Δεν βρέθηκε η διεύθυνση address");
    System.exit(-1);
}
catch(IOException e){
    System.out.println("Λάθος στην είσοδο/έξοδο");
    System.exit(-1);
}
out.println("Hello");
String line;
while(true){
    try{
        line = in.readLine();
        System.out.println(line);
    } catch (IOException e) {
        System.out.println("Η προσπάθεια ανάγνωσης απέτυχε");
        System.exit(-1);
    }
```

Στο παραπάνω παράδειγμα δημιουργείται ένα αντικείμενο της κλάσης `Socket` (πλευρά πελάτη) το οποίο προσπαθεί να συνδεθεί με τον διακομιστή στην διεύθυνση address και στην θύρα 8888. Αν δεν μπορέσει να βρει την διεύθυνση, προκαλεί την εξαίρεση `UnkonwHostException`.

Αφού γίνει η σύνδεση με τον διακομιστή, δημιουργείται ένας buffer εισόδου (το αντικείμενο `in`) και ένα αντικείμενου stream εξόδου κειμένου (το αντικείμενο `out`). Η μέθοδος `getInputStream()` επιστρέφει ένα stream εισόδου από το αντικείμενο `one_socket`, δηλαδή λαμβάνει ό,τι κείμενο στέλνει ο διακομιστής. Η κλάση `InputStreamReader` χρησιμοποιείται για να μετατρέψει τον επιστρεφόμενο τύπο της μεθόδου `getInputStream` (ο επιστρεφόμενος τύπος είναι InputStream) σε `Reader` που είναι ο τύπος κλάσης που δέχεται η κλάση BufferedReader. Η μέθοδος `getOutputStream()` επιστρέφει ένα stream εξόδου από το αντικείμενο `one_socket`. Το αντικείμενο `out` της κλάσης `PrintWriter` έχει όρισμα αυτήν την μέθοδο, άρα είναι υπεύθυνο (το αντικείμενο out) για την αποστολή κειμένου σε άλλο socket. Η πρόταση `out.println("Hello")` στέλνει την λέξη "Hello" στο άλλο socket με την βοήθεια της μεθόδου `println(String str)` της κλάσης `PrintWriter`. Στην συνέχεια, με την βοήθεια της μεθόδου readline() της κλάσης `BufferedReader` γίνεται η ανάγνωση του κειμένου που στέλνει το άλλο socket. Οι προτάσεις `System.exit(-1)` προκαλούν τον τερματισμό του προγράμματος.

Ακολουθούν δύο προγράμματα ένα για τον διακομιστή (server) και ένα για τον πελάτη (client) που στην ουσία είναι ένα πρόγραμμα κουβέντας (chat) μεταξύ δύο προγραμμάτων. Η γραφική διασύνδεση διακομιστή και πελάτη είναι όμοια, για αυτό αναλύεται μια φορά:

```
JButton button;
JTextField textfield;
JTextArea area;
Socket socket = null;
PrintWriter out = null;
BufferedReader in = null;
JScrollPane scroll;

AppClient(){
    super("Client Program");
    getContentPane().setLayout(null);
    setSize(250,250);
    scroll=new JScrollPane();
    button = new JButton("Send");
    button.addActionListener(this);
    textfield=new JTextField(40);
    area= new JTextArea();
    area.setLineWrap(true);
    area.setEditable(false);
    getContentPane().add(button);
    getContentPane().add(textfield);
    scroll.getViewport().add(area);
    getContentPane().add(scroll);
    button.setBounds(80,150,80,30);
    scroll.setBounds(4,1,230,100);
    textfield.setBounds(1,105,230,30);
}
```

Αυτό το κομμάτι του κώδικα είναι από το πρόγραμμα του πελάτη και η μόνη διαφορά του από το πρόγραμμα του διακομιστή είναι στις παρακάτω εντολές:

Στο πρόγραμμα του πελάτη:

```
AppClient(){

super("Client Program");
```

Στο πρόγραμμα του διακομιστή:

```
AppServer( ){

super("Server Program");
```

Βασικά, και τα δύο προγράμματα είναι πλαίσια. Αποτελούνται από πάνω προς τα κάτω από μια περιοχή κειμένου, ένα πεδίο κειμένου και ένα κουμπί. Η περιοχή κειμένου περιλαμβάνει, ότι έχει γραφεί μεταξύ διακομιστή, πελάτη. Η μέθοδος `setLineWrap(true)` τοποθετεί το κείμενο του αντικειμένου `area` στην επόμενη γραμμή όταν αυτό φτάσει στην άκρη της σειράς. Αν η επικοινωνία μεταξύ διακομιστή-πελάτη γεμίσει όλη την περιοχή κειμένου, τότε δεν θα είναι δυνατή η συνέχιση της, επειδή δεν θα είναι ορατή η νέα πληροφορία. Για να ξεπεραστεί αυτό το πρόβλημα η περιοχή κειμένου εισέρχεται σε ένα πάνελ κύλισης, δηλαδή ένα αντικείμενο της κλάση JScrollPane. Έτσι όταν γεμίσει η περιοχή κειμένου, με την μετακίνηση της μπάρας κύλισης θα φαίνεται το καινούργιο κείμενο του chat. Στο πεδίο κειμένου (αντικείμενο textfield) γράφονται τα μηνύματα που θέλει να στείλει ο ένας στον άλλο (διακομιστής, πελάτης). Τα μηνύματα στέλνονται με το πάτημα του κουμπιού με ετικέτα `send`. Όλα τα συστατικά ορίζονται πάνω στην γραφική διασύνδεση με ακριβείς συντεταγμένες με την βοήθεια της μέθοδο `setBounds(...)`.

Ακολουθεί το μέρος του κώδικα του προγράμματος της δημιουργίας του socket πελάτη:

```
public void runSocket(){
     // Δημιουργία socket
    try{
        socket = new Socket("127.0.0.1", 4444);
        out = new PrintWriter(socket.getOutputStream(), true);
        in = new BufferedReader(new
        InputStreamReader(socket.getInputStream()));
    } catch (UnknownHostException e) {
        System.out.println("Unknown host");
        System.exit(-1);
    } catch (IOException e) {
        System.out.println("No I/O");
        System.exit(-1);
    }
    String line;
    while(true){
          /* Τα δεδομένα που λαμβάνονται από τον διακομιστή γράφονται στην περιοχή κειμένου του πελάτη */
        try{
            line = in.readLine();
            area.setText(area.getText()+"\n"+line);
        } catch (IOException e) {
            System.out.println("Read failed");
            System.exit(-1);
        }
    }
```

```
}
```

Η μέθοδος `runSocket()` είναι ίδια με του προηγούμενου παραδείγματος. Στον παραπάνω κώδικα το αντικείμενο socket της κλάσης `Socket` κάνει μια αίτηση στην διεύθυνση 127.0.0.1 στην θύρα 4444. Ο αριθμός 127.0.0.1 είναι η προεπιλεγμένη διεύθυνση για έναν υπολογιστή που είναι offline. Αν ο υπολογιστής είναι online πρέπει να βρεθεί η διεύθυνση του και αυτή θα αντιπροσωπεύει την διεύθυνση του διακομιστή. Η θύρα του πελάτη είναι 4444 γιατί ο διακομιστής όπως θα φανεί στον κώδικα του αναμένει αίτηση από τον πελάτη σε αυτή την θύρα. Στην συνέχεια ορίζονται streams εισόδου, εξόδου. Ακολουθεί ένα ατέρμονος βρόχος **while**{…} ο οποίος διαβάζει το κείμενο που του στέλνει ο διακομιστής. Το κείμενο που στέλνει ο διακομιστής τυπώνεται στην περιοχή κειμένου με την πρόταση `area.setText(area.getText()+"\n"+line)`. Η περιοχή κειμένου συντηρεί το κείμενο που είδη έχει με την μέθοδο `getText()`. Στην συνέχεια αλλάζει σειρά και τέλος τυπώνει το νέο κείμενο που στάλθηκε από τον διακομιστή. Σε περίπτωση που ο διακομιστής κλείσει, προκαλείται εξαίρεση στην μέθοδο `readLine()` και εκτελείται η πρόταση `System.exit(-1)` που τερματίζει την λειτουργία του πελάτη.

Με το πάτημα του μοναδικού κουμπιού του προγράμματος εκτελείται η παρακάτω μέθοδος.:

```
public void actionPerformed(ActionEvent e){
    if(e.getSource() == button){
        String text = "Client : "+textfield.getText();
        out.println(text);
        area.setText(area.getText()+"\n"+text);
    }
}
```

Η μέθοδος `actionPerformed(...)` εκτελείται όταν πατιέται το κουμπί με ετικέτα `send`. Η πρόταση `out.printlnl(text)` στέλνει το κείμενο που υπάρχει στο πεδίο κείμενου του πελάτη, στον διακομιστή. Επιπλέον, αυτό το κείμενο τυπώνεται και στην περιοχή κειμένου του πελάτη (μέσω της πρότασης `area.setText(area.getText()+"\n"+text)`. Η μέθοδος `actionPerformed(...)` καλείται αυτόματα όταν πατιέται το συγκεκριμένο κουμπί.

Ολόκληρο το πρόγραμμα του πελάτη παρατίθεται παρακάτω:




```
import java.awt.event.*;
import javax.swing.*;
import java.io.*;
import java.net.*;

public class AppClient extends JFrame implements ActionListener {
    JButton button;
    JTextField textfield;
    JTextArea area;
    Socket socket = null;
```

```
PrintWriter out = null;
BufferedReader in = null;
JScrollPane scroll;

 // Δημιουργία της γραφικής διασύνδεσης του προγράμματος
AppClient(){
    super("Client Program");
    getContentPane().setLayout(null);
    setSize(250,250);
    scroll=new JScrollPane();
    button = new JButton("Send");
    button.addActionListener(this);
    textfield=new JTextField();
    area= new JTextArea();
    area.setLineWrap(true);
    getContentPane().add(button);
    getContentPane().add(textfield);
    scroll.getViewport().add(area);
    getContentPane().add(scroll);
    button.setBounds(80,150,80,30);
    scroll.setBounds(4,1,230,100);
    textfield.setBounds(1,105,230,30);
}

 // Εντολές που εκτελούνται με το πάτημα του κουμπιού του προγράμματος
public void actionPerformed(ActionEvent e){
    if(e.getSource() == button){
        String text = "Client : "+textfield.getText();
        out.println(text);
        area.setText(area.getText()+"\n"+text);
    }
}

 // Αρχικοποίηση του socket και ανάγνωση του κειμένου που στέλνει ο διακομιστής
public void runSocket(){
    try{
        socket = new Socket("127.0.0.1", 4444);
        out = new PrintWriter(socket.getOutputStream(), true);
        in = new BufferedReader(new
                    InputStreamReader(socket.getInputStream()));
    } catch (UnknownHostException e) {
        System.out.println("Unknown host");
        System.exit(-1);
    } catch (IOException e) {
        System.out.println("No I/O");
        System.exit(-1);
    }
    String line;
    while(true){
        try{
            line = in.readLine();
            area.setText(area.getText()+"\n"+line);
        } catch (IOException e) {
            System.out.println("Read failed");
            System.exit(-1);
        }
    }
}
```

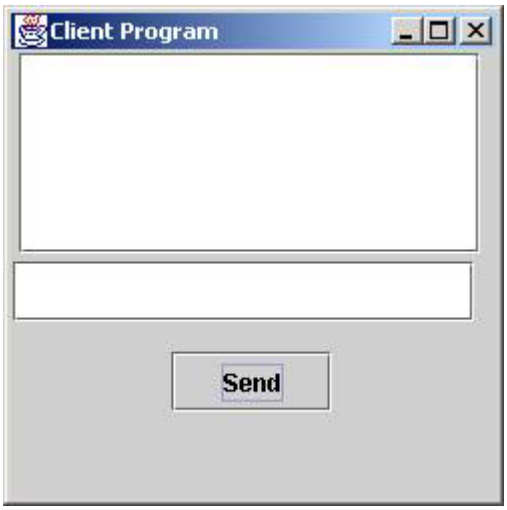
```
    public static void main(String[] args){
        AppClient appclient = new AppClient();
        WindowListener l = new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        };
        appclient.addWindowListener(l);
        appclient.setVisible(true);
        appclient.runSocket();
    }

}
```

*Παράδειγμα 11.2 – AppClient.java*

- source code

Ακολουθεί ένα στιγμιότυπο της εφαρμογής `AppClient.java`.




Στην μέθοδο `main(...)` η διασύνδεση `WindowListener` είναι υπεύθυνη για τα συμβάντα παραθύρου. Η μέθοδος της διασύνδεσης `windowClosing(...)` καλείται όταν κλείνει το πρόγραμμα και έχει ως αποτέλεσμα την εκτέλεση της πρότασης `System.exit(0)` που προκαλεί την διακοπή του socket. Τέλος, η μέθοδος `setVisible(...)` κάνει ορατό το πλαίσιο του προγράμματος.

Στην συνέχεια αναλύεται το πρόγραμμα του διακομιστή. Η γραφική διασύνδεση του αναλύθηκε πιο πάνω, για αυτό ακολούθως παρουσιάζεται το υπόλοιπο πρόγραμμα του. Το πρόγραμμα του διακομιστή έχει την ιδιότητα όταν ο πελάτης κλείσει την σύνδεση του, να περιμένει άλλο πελάτη να συνδεθεί μαζί του.

```
public void runServerSocket(){
    startOfServer();
    int i=1;
    while(true){
        if (i==0) {
```

```
            end();
            startOfServer();
         }
        try{
            line = in.readLine();
            area.setText(area.getText()+"\n"+line);
            i=1;
        }catch (IOException e) {
            client=null;
            i=0;
        }
    }
}
void startOfServer(){
    try{
        server = new ServerSocket(4444);
        String text="Wait to connect ...";
        area.setText(area.getText()+"\n"+text);
    } catch (IOException e) {
        System.out.println("Could not listen on port 1111");
        System.exit(-1);
    }
    try{
        client = server.accept();
        String text="Connection established";
        area.setText(area.getText()+"\n"+text);
    }
    catch (IOException e) {
        System.out.println("Accept failed: 4444");
        System.exit(-1);
    }
    try{
        in = new BufferedReader(new
        InputStreamReader(client.getInputStream()));
        out = new PrintWriter(client.getOutputStream(), true);
   }catch (IOException e) {
        System.out.println("Accept failed: 4444");
        System.exit(-1);
    }
}

void end(){
    try{
        in.close();
        out.close();
        server.close();
        String text="Connection closed";
        area.setText(area.getText()+"\n"+text);
    }catch (IOException e) {
        System.out.println("Could not close.");
        System.exit(-1);
    }
}
```

Αρχικά η μέθοδος `runServerSocket()` εκτελεί την μέθοδο `startOfServer()`. Η μέθοδος αυτή δημιουργεί ένα αντικείμενο της κλάσης `ServerSocket`. Στην συνέχεια αυτό το αντικείμενο βρίσκεται σε κατάσταση αναμονής μέχρι ο πελάτης να κάνει κάποια αίτηση. Η αίτηση που κάνει ο πελάτης γίνεται δεκτή με την μέθοδο `accept()`, δηλαδή αν δεν συμβεί κάποια εξαίρεση, πελάτης και διακομιστής συνδέονται μεταξύ τους. Ακολουθεί η δημιουργία streams εισόδου και εξόδου στα αντικείμενα `in` και `out` αντίστοιχά. Επιστρέφοντας στην μέθοδο `runServerSocket()`, ορίζεται η ακέραια μεταβλητή i=1 που παίζει το ρόλο σημαίας. Έχει την τιμή ένα όταν υπάρχει σύνδεση μεταξύ πελάτη και διακομιστή. Αν διακοπεί η σύνδεσή τους τότε γίνεται μηδέν. Κάτω από τον ορισμό της ακέραιας μεταβλητής υπάρχει ένας ατέρμονος βρόχος ο οποίος διαβάζει μέσω του αντικειμένου `in` το κείμενο που του έστειλε ο πελάτης. Το κείμενο αυτό καταχωρείται στην περιοχή κειμένου με την πρόταση `area.setText(area.getText()+"\n"+line)`. Αν ο πελάτης κλείσει την σύνδεσή του μεταξύ του διακομιστή, τότε προκαλείται μια εξαίρεση και εκτελείται το μπλοκ **catch** το οποίο θέτει στην μεταβλητή `i` την τιμή μηδέν. Όταν συμβεί αυτό εκτελείται το μπλοκ "if(i==0) {end(); startOfServer();}". Η μέθοδος `end()` κλείνει τα stream εισόδου εξόδου και το αντικείμενο της κλάσης `ServerSocket`. Στην συνέχεια, καλείται εκ νέου η μέθοδος `startOfServer()` για να δημιουργήσει ένα αντικείμενο της κλάσης `ServerSocket` που θα δεχθεί μια αίτηση από τον πελάτη και θα δημιουργήσει streams εισόδου και εξόδου. Όλες οι εντολές που μπορούν να προκαλέσουν εξαίρεση βρίσκονται μέσα σε **try** …**catch** … μπλοκ.

Οι εντολές που εκτελούνται όταν πατηθεί το κουμπί με ετικέτα `send` δίνονται από την παρακάτω μέθοδο.

```
public void actionPerformed(ActionEvent e) {
    if((e.getSource() == button) & (client!=null))
    {
        String text="Server : "+textfield.getText();
        out.println(text);
        area.setText(area.getText()+"\n"+text);
    }
}
```

Το κείμενο (getText()) που υπάρχει στο πεδίο κειμένου του διακομιστή με πρόθεμα την λέξη "Server : " στέλνεται στον πελάτη με την πρόταση `out.println(text)` και τυπώνεται στην περιοχή κειμένου του διακομιστή με την πρόταση `area.setText(area.getText()+"\n"+text)`.

Όλος ο κώδικας του προγράμματος διακομιστή είναι:




```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.io.*;
import java.net.*;

public class AppServer extends JFrame implements ActionListener{
    JButton button;
    ServerSocket server = null;
```

```
Socket client = null;
BufferedReader in = null;
PrintWriter out = null;
JTextField textfield;
JTextArea area;
JScrollPane scroll;
String line;

 // Δημιουργία της γραφική διασύνδεσης του προγράμματος
AppServer( ){
    super("Server Program");
    getContentPane().setLayout(null);
    setSize(250,250);
    scroll=new JScrollPane();
    button = new JButton("Send");
    button.addActionListener(this);
    textfield=new JTextField();
    area= new JTextArea();
    area.setLineWrap(true);
    area.setEditable(false);
    getContentPane().add(button);
    getContentPane().add(textfield);
    scroll.getViewport().add(area);
    getContentPane().add(scroll);
    button.setBounds(80,150,80,30);
    scroll.setBounds(4,1,230,100);
    textfield.setBounds(1,105,230,30);
}

 // Εντολές που εκτελούνται με το πάτημα του κουμπιού
public void actionPerformed(ActionEvent e) {
    if((e.getSource() == button) & (client!=null))
    {
        String text="Server : "+textfield.getText();
        out.println(text);
        area.setText(area.getText()+"\n"+text);
    }
}

 // Ανάγνωση του κειμένου που στέλνει ο πελάτης
public void runServerSocket(){
    startOfServer();
    int i=1;
    while(true){
        if (i==0) {
           end();
           startOfServer();
        }
        try{
            line = in.readLine();
            area.setText(area.getText()+"\n"+line);
            i=1;
        } catch (IOException e) {
            client=null;
            i=0;
        }
    }
```

```
}

 /* Δημιουργία του αντικειμένου της κλάσης ServerSocket, αναμονή για αίτηση από τον πελάτη,
 δημιουργία streams εισόδου και εξόδου */
void startOfServer(){
    try{
        server = new ServerSocket(4444);
        String text="Wait to connect ...";
        area.setText(area.getText()+"\n"+text);
    } catch (IOException e) {
        System.out.println("Could not listen on port 1111");
        System.exit(-1);
    }
    try{
        client = server.accept();
        String text="Connection established";
        area.setText(area.getText()+"\n"+text);
    }
    catch (IOException e) {
        System.out.println("Accept failed: 4444");
        System.exit(-1);
    }
    try{
        in = new BufferedReader(new
                    InputStreamReader(client.getInputStream()));
        out = new PrintWriter(client.getOutputStream(), true);
    }catch (IOException e) {
        System.out.println("Accept failed: 4444");
        System.exit(-1);
    }
}

 /* Τερματισμός stream εισόδου, εξόδου και του αντικειμένου της κλάσης ServerSocket */
void end(){
    try{
        in.close();
        out.close();
        server.close();
        String text="Connection closed";
        area.setText(area.getText()+"\n"+text);
    }catch (IOException e) {
        System.out.println("Could not close.");
        System.exit(-1);
    }
}

public static void main(String[] args){
    AppServer server=new AppServer();
    WindowListener l = new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
            System.exit(0);
        }
    };
    server.addWindowListener(l);
    server.setVisible(true);
    server.runServerSocket();
}
```

}

*Παράδειγμα 11.3 – AppServer.java*

- source code

Ακολουθεί ένα στιγμιότυπο της εφαρμογής `AppServer.java`.




Στην μέθοδο `main(...)` δημιουργείται ένα αντικείμενο του προγράμματος, γίνεται ορατό με την μέθοδο `setVisible(true)` και στην συνέχεια καλείται η μέθοδος `runServerSocket()` η οποία εκτελεί τις περισσότερες λειτουργίες του προγράμματος.

Για να τρέξουν τα προγράμματα ταυτόχρονα πρέπει ο χρήστης να εκτελέσει το καθένα από αυτά σε διαφορετική κονσόλα εισόδου.







## 12.1 Εισαγωγή στα Swing

***JFC*** είναι το ακρωνύμιο των λέξεων Java Foundation Classes το οποίο περιλαμβάνει μια ομάδα χαρακτηριστικών για την δημιουργία γραφικής διασύνδεσης χρήστη (GUI). Δύο από τα χαρακτηριστικά του JFC είναι το Swing και το Look and Feel (εμφάνιση και αίσθηση).

Το ***Swing*** είναι ουσιαστικά μια επέκταση του AWT. Περιλαμβάνει ότι και το AWT (κουμπιά , ετικέτες, πεδία κειμένου κ.α) αλλά με νέα χαρακτηριστικά. Οι καινοτομίες αυτές είναι νέα συστατικά, επεκταμένα χαρακτηριστικά συστατικών και καλύτερη αίσθηση και εμφάνιση. Το Swing ενσωματώθηκε στην Java στην έκδοση 2. Τα στοιχεία του Swing είναι μέρος του πακέτου `javax.swing`.

Το ***Look and Feel*** δίνει την δυνατότητα σε ένα πρόγραμμα που χρησιμοποιεί στοιχεία Swing να διαλέξει την εμφάνιση και την αίσθησή του, δηλαδή αν θα μοιάζει οπτικά με πρόγραμμα των Windows, Solaris ή θα έχει ένα στυλ της Java που ονομάζεται Metal. Το Look and Feel αναλύεται στο υποκεφάλαιο 12.7.

Όπως έγινε αναφορά πιο πάνω, όλα τα συστατικά του AWT υπάρχουν και στο Swing. Η διαφορά στον ορισμό των αντικειμένων των κλάσεων είναι ότι τα συστατικά του Swing αρχίζουν από το γράμμα J. Δηλαδή το αντίστοιχο συστατικό του `Button` στο AWT θα είναι στο Swing το `JButton` κ.ο.κ.

Άλλες σημαντικές διαφορές είναι: Τα κουμπιά και ετικέτες του Swing εκτός από κείμενο μπορούν να περιέχουν και εικονίδια. ***Εικονίδιο*** είναι ένα γραφικό μικρών διαστάσεων. Ένα ακόμα χαρακτηριστικό είναι η δυνατότητα πρόσθεσης ή αλλαγής περιθωρίων γύρω από τα περισσότερα συστατικά.

## 12.2 Γραφική διασύνδεση χρήστη με χρήση Swing

Η δημιουργία μιας γραφικής διασύνδεσης χρήστη με Swing είναι περίπου ίδια με αυτήν του AWT και δημιουργείται με τα παρακάτω βήματα .

- Σύνθεση μιας γραφικής διασύνδεσης χρήστη προσθέτοντας συστατικά (components) στους ενδιάμεσους υποδοχείς (content pane – τομέας περιεχομένων). Οι ενδιάμεσοι υποδοχείς τίθενται στους ανώτερους υποδοχείς.
- Δημιουργία χειριστών συμβάντων που ανταποκρίνονται στην αλληλεπίδραση χρήστη με το GUI.

Για την σύνθεση μιας γραφικής διασύνδεσης υπάρχει εκτενής αναφορά στα παρακάτω κεφάλαια. Για την δημιουργία χειριστών συμβάντων δεν υπάρχει κάποιο ξεχωριστό κεφάλαιο, αλλά στα κεφάλαια των συστατικών υπάρχουν επιγραμματικά οι ακροατές συμβάντων που ταιριάζουν με αυτά. Γενικά για τους χειριστές συμβάντων ισχύουν οι έννοιες του κεφαλαίου 8.4 (AWT) και στο Swing.

---

# 12.3 Σύνθεση γραφικής διασύνδεσης με χρήση Swing

---

Η σύνθεση γραφικής διασύνδεσης με Swing είναι ίδια με την AWT με την διαφορά ότι υπάρχουν δύο επίπεδα υποδοχέων. Η διαδικασία σύνθεσης είναι η εξής: Αρχικά τα συστατικά προστίθενται σε ένα ***ενδιάμεσο υποδοχέα*** που καλείται τομέας περιεχομένων (content pane). Ο ***τομέας περιεχομένων*** είναι υπεύθυνος για την διάταξη των συστατικών. Οι διαχειριστές διάταξης ορίζονται σε αυτόν. Για τους διαχειριστές διάταξης ισχύουν οι έννοιες που εξετάστηκαν στο κεφάλαιο 8.3.2. Τέλος οι ενδιάμεσοι υποδοχείς προστίθενται στους κύριους (ανώτερους) υποδοχείς.

Οι ανώτεροι υποδοχείς είναι οι κλάσεις `JFrame`, `JApplet`, `JOptionPane`. Οι ενδιάμεσοι υποδοχείς είναι οι `JPanel`, `JScrollPane`, `JSplitPane`, `JTabbedPane`, `JToolBar`. Οι ενδιάμεσοι υποδοχείς προστίθενται στους ανώτερους υποδοχείς με την μέθοδο `setContentPane(...)`. Οι διαχειριστές διάταξης ορίζονται στους ενδιάμεσους υποδοχείς με την μέθοδο `setLayout(...)`. Τα συστατικά προστίθενται στους ενδιάμεσους υποδοχείς με την μέθοδο `add(...)`. Οι ενδιάμεσοι υποδοχείς μπορούν να προστεθούν σε άλλους ενδιάμεσους υποδοχείς.

Οι ανώτεροι υποδοχείς έχουν προεπιλεγμένους τομείς περιεχομένων, δηλαδή σε ένα πρόγραμμα μπορεί να δείτε την εντολή `getContentPane()` (δηλαδή επιστροφή του τομέα περιεχομένων) χωρίς να έχει οριστεί ενδιάμεσος υποδοχέας.

---

# 12.4 Συστατικά

---

Στα ακόλουθα κεφάλαια θα αναλυθούν τα πιο σημαντικά χαρακτηριστικά του Swing. Όλα τα συστατικά κληρονομούν την κλάση `JComponent`. Επίσης κληρονομούν και τις κλάσεις `Component` και `Container` όπως τα συστατικά του AWT.

Στα παραδείγματα που υπάρχουν στα κεφάλαια των συστατικών, ως ανώτερος υποδοχέας θεωρείται ένα αντικείμενο της κλάσης `JApplet` η οποία είναι δευτερεύουσα της κλάσης Applet. Διεξοδική αναφορά στην κλάση `JApplet` υπάρχει στο υποκεφάλαιο 12.6.3.

Σημαντικές μέθοδοι που κληρονομούνται από τα συστατικά του Swing είναι της κλάσης Component και της κλάσης `JComponent` της οποίας οι κυριότερες μέθοδοι επεξήγονται ακολούθως:

- `setOpaque(boolean bl)`: Θέτει πότε ένα συστατικό θα είναι αδιαφανές. Δηλαδή ένα αδιαφανές συστατικό θέτει την έκταση που κατέχει σε ένα ενδιάμεσο υποδοχέα με το χρώμα του παρασκηνίου του.
- `isOpaque()`: Επιστρέφει τύπο **boolean** που προσδιορίζει πότε ένα συστατικό είναι αδιαφανές.
- `setToolTipText(String str)`: Θέτει το κείμενο που θα εμφανίζεται σε ένα συστατικό σε ξεχωριστό κουτάκι πάνω από αυτό όταν το ποντίκι βρίσκεται πάνω από το συστατικό. Συνήθως χρησιμοποιείται σαν ετικέτα βοήθειας για την λειτουργία του συστατικού.

---

### 12.4.1 Κουμπιά και πλαίσια ελέγχου (κλάση JCheckBox)

---

Αυτό το υποκεφάλαιο αναφέρεται στα συστατικά που είναι δευτερεύουσες κλάσεις της `AbstractButton`. Αυτές είναι οι `JButton`, `JToggleButton`. Υποκλάση της `JToggleButton` είναι η `JCheckBox`.

Σημαντικές μέθοδοι της κλάσης `AbstractButton` είναι:

- `addActionListener(ActionListener al)`: Η μέθοδος αυτή συσχετίζει τα συμβάντα ενέργειας (ActionListener) με ένα συστατικό `JButton`.
- `addItemListener(ItemListener il)`: Προσθέτει έναν ακροατή στοιχείων (ItemListener) σε ένα συστατικό JCheckBox.
- `setEnabled(boolean bl)`: Ανάλογα με την τιμή της μεταβλητής `bl` ενεργοποιεί ή απενεργοποιεί ένα συστατικό.
- `setHorizontalAlignment(int hl)`: Θέτει την οριζόντια διεύθυνση του κειμένου και του εικονιδίου (icon). Οι τιμές της μεταβλητής `hl` μπορεί να είναι `CENTER`, `LEFT`, `RIGHT`, `LEADING` ή `TRAILING`. Η προεπιλεγμένη τιμή είναι η `CENTER`.
- `setHorizontalTextPosition(int h)`: Θέτει την οριζόντια θέση του κειμένου σε σχέση με το εικονίδιο. Οι τιμές της μεταβλητές `h` μπορεί να είναι `RIGHT`, `LEFT`, `CENTER`, `LEADING` ή `TRAILING`. Η προεπιλεγμένη τιμή είναι η `RIGHT`.
- `setIcon(Icon i)`: Θέτει ένα εικονίδιο σε ένα συστατικό.
- `setMnemonic(char ch)`: Θέτει το κουμπί του πληκτρολογίου που μαζί με το alt θα ενεργοποιούν το συστατικό. Η τιμή της μεταβλητής `ch` έχει την τύπο `KeyEvent.VK_X` όπου `X` οποιοδήποτε γράμμα.

- `setText(String str)`: Θέτει το κείμενο του συστατικού.
- `setVerticalAlignment(int va)`: Θέτει την κάθετη σύνταξη του κειμένου και του εικονιδίου. Οι τιμές της μεταβλητής `va` μπορεί να είναι `CENTER`, `TOP`, `BOTTOM`. Η προεπιλεγμένη τιμή είναι η `CENTER`.
- `setVerticalTextPosition(int v)`: Θέτει την κάθετη θέση του κειμένου σε σχέση με το εικονίδιο. Η μεταβλητή `v` μπορεί να πάρει της εξής τιμές: `CENTER`, `TOP`, `BOTTOM`. Η προεπιλεγμένη τιμή είναι η `CENTER`.

Οι παραπάνω μέθοδοι κληρονομούνται σε όλα τα αντικείμενα των υποκλάσεων της `AbstractButton`. Επιπρόσθετα κληρονομούνται οι μέθοδοι της κλάσης `Component` που αναλύθηκαν στο κεφάλαιο 8.3.1.

Η κλάση `JButton` ορίζει ένα κουμπί του οποίου οι πιο σημαντικοί δημιουργοί είναι:

- `JButton(String str)`: Δημιουργεί ένα αντικείμενο της `JButton` με κείμενο `str`.
- `JButton(String str, Icon i)`: Δημιουργεί ένα αντικείμενο της `JButton` με κείμενο `str` και εικονίδιο `i`.

Η κλάση `JCheckBox` ορίζει ένα πλαίσιο ελέγχου. Οι πιο σημαντικοί δημιουργοί του είναι:

- `JCheckBox(String str, boolean bl)`: Δημιουργεί ένα πλαίσιο ελέγχου με κείμενο `str`. Η μεταβλητή `bl` καθορίζει αν το πλαίσιο θα είναι επιλεγμένο.
- `JCheckBox(String str, Icon i, boolean bl)`: Η ίδια με την παραπάνω και επιπλέον έχει εικονίδιο `i`.

Στο παρακάτω παράδειγμα ένα αντικείμενο `JButton` και ένα `JChechBox` προστίθενται στον ενδιάμεσο υποδοχέα JPanel. Το κουμπί ορίζεται έτσι ώστε να ενεργοποιείται και με το ταυτόχρονο πάτημα του alt και του J.




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

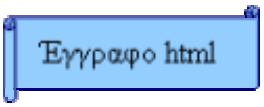
public class AppJButton extends JApplet{

    public void init() {
        JPanel jp=new JPanel();
        JButton jb=new JButton("JButton");
        JCheckBox jcb=new JCheckBox("CheckBox",true);
        jb.setMnemonic(KeyEvent.VK_J);
        jp.add(jb);
        jp.add(jcb);
        setContentPane(jp);
    }

}
```

*Παράδειγμα 12.1 – AppJButton.java*

- source code




```
<APPLET CODE = "AppJButton.class" WIDTH = 200 HEIGHT= 100>
</APPLET>
```

*appjbuttonviewer.html*

Προσοχή: Το παραπάνω έγγραφο html τρέχει μόνο με το εργαλείο **appletviewer** της Java. Ενώ στις μικροεφαρμογές του AWT το έγγραφο html μπορεί να φορτωθεί και από έναν εξεταστή, στις μικροεφαρμογές Swing δεν είναι δυνατό, χωρίς κάποια τροποποίηση του. Στο πρωτότυπο έγγρφο html πρέπει να προστεθεί κώδικας μέσω του εργαλείου html converter έτσι ώστε να μπορεί να φορτωθεί από έναν εξεταστή. Η χρήση του εργαλείου είναι απλή. Επιλέγουμε με την βοήθεια του εργαλείου τον κατάλογο που βρίσκεται το έγγραφο html και στην συνέχεια πατώντας το κουμπί convert προστίθεται ο απαραίτητος κώδικας στο έγγραφο. Σε αυτό το κεφάλαιο τα πρωτότυπα έγγραφα html που τρέχουν μόνο με το εργαλείο appletviewer έχουν κατάληξη viewer ενώ τα έγγραφα που έχουν επιπλεόν κώδικα έχουν το όνομα του προγράμματος. Π.χ. το παραπάνω πρόγραμμα AppJButton.java έχει πρωτότυπο έγγραφο html το appjbuttonviewer.html και τροποποιημένο το appjbutton.html.

---

### 12.4.2 Επιλογέας χρωμάτων (κλάση JColorChooser)

---

Η κλάση `JColorChooser` δημιουργεί μια παλέτα χρωμάτων. Η παλέτα της `JColorChooser` περιλαμβάνει τρία πλαίσια μέσα από τα οποία μπορεί να επιλεχθεί ένα επιθυμητό χρώμα. Τα τρία αυτά πλαίσια είναι το Swatches και τα μοντέλα χρωμάτων HSB και RGB. Τα αρχικά του μοντέλου RGB σημαίνουν red, green, blue και εκφράζουν την αναλογία στην παρασκευή του τελικού χρώματος. Οι πιο σημαντικοί δημιουργοί της κλάσης `JColorChooser` είναι:

- `JColorChooser()`: Δημιουργεί ένα αντικείμενο της κλάσης `JColorChooser` με προεπιλεγμένη τιμή το άσπρο χρώμα.
- `JColorChooser(Color c)`: Δημιουργεί ένα αντικείμενο της κλάσης `JColorChooser` με προεπιλεγμένη τιμή το χρώμα `c`.

Αξιομνημόνευτες μέθοδοι της `JColorChooser` είναι:

- `getColor()`: Επιστρέφει το επιλεγμένο χρώμα της παλέτας.
- `setColor(Color c)`: Θέτει το χρώμα της παλέτας.
- `setColor(int r, int g, int b)`: Θέτει το χρώμα της παλέτας. Κάθε ακέραιο όρισμα εκφράζει το ποσοστό του κόκκινου του πράσινου και του μπλε χρώματος αντίστοιχα στις μεταβλητές `r`, `g`, `b`. Οι τιμές των μεταβλητών μπορεί να είναι από 0 έως 255.

### 12.4.3 Επιλογέας αρχείων (κλάση JFileChooser)

Ο επιλογέας αρχείων παρέχει ένα περιβάλλον γραφικής διασύνδεσης στο χρήστη για την περιήγηση του στα αρχεία του υπολογιστή. Αυτή η διαδικασία μπορεί να περιλαμβάνει επιλογή αρχείου, αλλαγή καταλόγου, δημιουργία καταλόγου κ.ο.κ.. Ο επιλογέας αρχείων υλοποιείται με την κλάση `JFileChooser` της οποίας οι πιο σημαντικοί δημιουργοί είναι:

- `JFileChooser()`: Δημιουργεί ένα επιλογέα αρχείων με επιλεγμένο κατάλογο τον default του συστήματος.
- `JFileChooser(String str)`: Δημιουργεί έναν επιλογέα αρχείων με επιλεγμένο τον κατάλογο που υπάρχει στην γραμματοσειρά `str`.

Μια σημαντική λεπτομέρεια είναι ότι ένα αντικείμενο της κλάσης `JFileChooser` είναι υπεύθυνο μόνο για την παρουσίαση του επιλογέα αρχείων. Οι εντολές για το άνοιγμα ή το σώσιμο ενός αρχείου πρέπει να γίνουν από τον προγραμματιστή.

Ο επιλογέας αρχείων γίνεται ορατός με τις μεθόδους:

- `showDialog(Component c, String str)`: Εμφανίζει τον επιλογέα αρχείων. Η μεταβλητή `c` είναι το όνομα του αντικειμένου της κλάσης στο οποία θα εμφανιστεί ο επιλογέας αρχείων. Η μέθοδος αυτή επιστρέφει ακέραια τιμή, που μπορεί να είναι η `CANCEL_OPTION` ή `APPROVE_OPTION`. Για τις τιμές αυτές είναι υπεύθυνα δύο κουμπιά που βρίσκονται πάνω στο επιλογέα αρχείων. Το ένα έχει πάντα την τιμή `Cancel` και όταν επιλέγεται επιστρέφει την τιμή `CANCEL_OPTION`. Το δεύτερο έχει την τιμή που ορίζεται από την μεταβλητή `str` και όταν επιλέγεται η μέθοδος επιστρέφει τον ακέραιο `APPROVE_OPTION`.
- `showSaveDialog(Component c)`: Ισχύει ότι και στην παραπάνω μόνο που ως τιμή της μεταβλητής `str` είναι η τιμή `Save`.
- `showOpenDialog(Component c)`: Ισχύει ότι και στην πρώτη με την διαφορά ότι η μεταβλητή `str` έχει την τιμή `Open`.

Χρήσιμες μέθοδοι του επιλογέα αρχείων είναι:

- `getFileSelectionMode()`: Επιστρέφει ένα ακέραιο με την τρέχουσα κατάσταση του επιλογέα αρχείων, δηλαδή εάν θα εμφανίζονται μόνο τα αρχεία, μόνο οι κατάλογοι ή και τα δύο μαζί . Οι αντίστοιχες ακέραιες τιμές για κάθε κατάσταση είναι `FILES_ONLY`, `DIRECTORIES_ONLY`, `FILES_AND_DIRECTORIES`.
- `getSelectedFile()`: Επιστρέφει το αρχείο που είναι επιλεγμένο. Ο τύπος επιστροφής δεν είναι **String** αλλά `File`.
- `getSelectedFiles()`: Επιστρέφει τα επιλεγμένα αρχεία σε ένα πίνακα τύπου `File`.
- `setFileSelectionMode(int x)`: Είναι η συμπληρωματική της `getFileSelectionMode()`, δηλαδή θέτει την κατάσταση του επιλογέα αρχείων. Έτσι η μεταβλητή `x` μπορεί να πάρει τις τιμές `FILES_ONLY`, `DIRECTORIES_ONLY`, `FILES_AND_DIRECTORIES`.
- `setSelectedFile(File f)`: Θέτει ένα αρχείο `f` ως επιλεγμένο.

- `setSelectedFiles(File[] f)`: Θέτει έναν πίνακα αρχείων `f` ως πίνακα επιλεγμένων αρχείων.

Για τον χειρισμό ενός γεγονότος που παράγεται από ένα αντικείμενο της κλάσης `JFileChooser` χρησιμοποιείται ο ακροατής συμβάντων ActionListener.

---

### 12.4.4 Ετικέτες (κλάση JLabel)

---

Η ετικέτα είναι ένα συστατικό που περιέχει κείμενο ή κάποιο εικονίδιο. Η υπεύθυνη κλάση για αυτήν είναι η `JLabel`. Η κλάση `JLabel` κληρονομεί απευθείας την `JComponent`. Οι πιο σημαντικοί δημιουργοί της είναι:

- `JLabel(String str)`: Δημιουργεί ένα αντικείμενο `JLabel` με κείμενο `str`.
- `JLabel(String str, Icon i, int x)`: Δημιουργεί ένα αντικείμενο `JLabel` με καθορισμένο κείμενο `str`, εικονίδιο `i` και στοίχιση `x`. Οι πιθανές τιμές του ακέραιου `x` είναι οι `LEFT`, `CENTER`, `RIGHT`, `LEADING` ή `TRAILING`.
- `JLabel(String str, int x)`: Είναι ίδια με την παραπάνω, απλώς δεν περιέχει κάποιο εικονίδιο.

Χρήσιμες μέθοδοι της κλάσης `JLabel` είναι:

- `getIcon()`: Επιστρέφει το εικονίδιο που εμφανίζει η ετικέτα.
- `getText()`: Επιστρέφει το κείμενο που υπάρχει στην ετικέτα
- `setIcon(Icon i)`: Θέτει το εικονίδιο `i` σε ένα αντικείμενο της `JLabel`.
- `setText(String str)`: Θέτει το κείμενο `str` σε μια ετικέτα.

---

### 12.4.5 Λίστες (κλάση JList)

---

Η ***λίστα*** είναι μια στήλη από στοιχεία τα οποία μπορεί να επιλέξει ο χρήστης. Αρμόδια κλάση για τις λίστες είναι η `JList` η οποία επιτρέπει την επιλογή περισσότερων τους ενός στοιχείων. Οι σημαντικότεροι δημιουργοί της `JList` είναι:

- `JList()`: Δημιουργεί μια άδεια λίστα.
- `JList(ListModel lm)`: Δημιουργεί μια λίστα με στοιχεία από το `lm` μοντέλο. Το `lm` μοντέλο περιέχει τα στοιχεία που προστίθενται στην λίστα.

Όταν αρχικοποιείται μια λίστα ο κατασκευαστής δημιουργεί το προεπιλεγμένο μοντέλο λίστας. Αυτό το μοντέλο λίστας είναι όμως αμετάβλητο. Στην θέση του μπορεί να χρησιμοποιηθεί ένα μοντέλο λίστας που επιτρέπει την αλλαγή της κατάστασης των στοιχείων, όπως είναι ένα αντικείμενο της κλάσης `DefaultListModel`.

Χρήσιμές μέθοδοι της κλάσης `JList` είναι:

- `addListSelectionListener(ListSelectionListener l)`: Η μέθοδος αυτή συσχετίζει μια λίστα με έναν ακροατή συμβάντων `ListSelectionListener`. Αυτός ο ακροατής είναι υπεύθυνος για γεγονότα αλλαγής τιμής του επιλεγμένου στοιχείου της λίστας. Η μέθοδος του συγκεκριμένου ακροατή είναι η `valueChanged(ListSelectionEvent e)`.
- `getSelectedIndex()`: Επιστρέφει έναν ακέραιο που δείχνει την θέση του πρώτου επιλεγμένου στοιχείου στην λίστα. Αν δεν υπάρχει επιλεγμένο στοιχείο επιστρέφει την τιμή –1.
- `getSelectedIndices()`: Επιστρέφει έναν πίνακα ακεραίων με την θέση των επιλεγμένων στοιχείων στην λίστα.
- `getSelectedValue()`: Επιστρέφει την τιμή του πρώτου επιλεγμένου στοιχείου στην λίστα. Αν δεν υπάρχει επιλεγμένο στοιχείο επιστρέφει την τιμή **null**.
- `getSelectedValues()`: Επιστρέφει τις τιμές των επιλεγμένων στοιχείων της λίστας.
- `getSelectionMode()`: Επιστρέφει μια ακέραια τιμή που δείχνει αν επιτρέπεται η πολλαπλή επιλογή στοιχείων.
- `setSelectedIndex(int x)`: Επιλέγει το στοιχείο που βρίσκεται στην θέση `x` της λίστας
- `setSelectedIndices(int[] x)`: Θέτει ως επιλεγμένα, τα στοιχεία που βρίσκονται στις θέσεις της λίστας που προσδιορίζονται από τον πίνακα `x`.
- `setSelectedValue(Object obj, boolean b)`: Θέτει την τιμή του επιλεγμένου στοιχείου. Το όρισμα `b` είναι υπεύθυνο για την εμφάνιση ή μη μιας λίστας κύλισης.
- `setSelectionMode(int x)`: Προσδιορίζει πότε θα είναι απλή ή πολλαπλή η επιλογή στοιχείων. Το όρισμα `x` μπορεί να πάρει τις τιμές `INGLE_SELECTION`, `SINGLE_INTERVAL_SELECTION`, `MULTIPLE_INTERVAL_SELECTION`. Ως πρόθεμα των τιμών αυτών υπάρχει η `ListSelectionModel`.

Στην συνέχεια αναφέρονται κάποιες μέθοδοι της κλάσης `DefaultListModel`.

- `addElement(Object obj)`: Προσθέτει ένα συστατικό στο τέλος της λίστας.
- `getElementAt(int x)`: Επιστρέφει το συστατικό που βρίσκεται στην θέση `x` του πίνακα. Ο τύπος επιστροφής της μεθόδου είναι `Object`.
- `getSize()`: Επιστρέφει τον αριθμό των στοιχείων στην λίστα.
- `insertElementAt(Object obj, int x)`: Εισάγει ένα συστατικό στην θέση `x` της λίστας.
- `remove(int x)`: Αφαιρεί το συστατικό που βρίσκεται στην θέση `x` της λίστας.
- `toString()`: Επιστρέφει ένα **String**, το οποίο δείχνει και πιστοποιεί τις ιδιότητες του αντικειμένου που προσδιορίζει η μέθοδος.

Στην συνέχεια υπάρχει ένα πρόγραμμα που χρησιμοποιεί μια λίστα για την πρόσθεση και την αφαίρεση ονομάτων από αυτή.




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

```
import javax.swing.event.*;

public class AppJList extends JApplet {

    JList list;
    DefaultListModel listModel;
    static final String inString = "Εισαγωγή";
    static final String delString = "Διαγραφή";
    JButton delButton;
    JTextField friendName;

    public void init() {
        listModel = new DefaultListModel();
        listModel.addElement("Μιχάλης Παπαγιαννάκης");
        listModel.addElement("Κώστας Τζιμόπουλος");
        listModel.addElement("Δημήτρης Παπαδόπουλος");
        listModel.addElement("Γιάννης Καραδήμος");
        /* Δημιουργείται μια λίστα και τοποθετείται σε ένα πάνελ κύλισης. Ορίζονται δύο κουμπιά
        το inButton και το delButton. Το πρώτο συσχετίζεται με την εσωτερική κλάση inListener
        η οποία υλοποιεί την διασύνδεση ActionListener. Το κουμπί inButton προσθέτει το όνομα
        που βρίσκεται στο πεδίο κειμένου στην λίστα. Το δεύτερο κουμπί συσχετίζεται με την κλάση
        delButton που υλοποιεί την διασύνδεση ActionListener. Το κουμπί delButton αφαιρεί το
        όνομα που βρίσκεται επιλεγμένο στην λίστα από αυτή. */
        list = new JList(listModel);
        list.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
        list.setSelectedIndex(0);
        JScrollPane listScrollPane = new JScrollPane(list);
        JButton inButton = new JButton(inString);
        inButton.setActionCommand(inString);
        inButton.addActionListener(new inListener());
        delButton = new JButton(delString);
        delButton.setActionCommand(delString);
        delButton.addActionListener(new delListener());
        friendName = new JTextField(10);
        friendName.addActionListener(new inListener());
        String name = listModel.getElementAt(
        list.getSelectedIndex()).toString();
        friendName.setText(name);
        JPanel buttonPane = new JPanel();
        buttonPane.add(friendName);
        buttonPane.add(inButton);
        buttonPane.add(delButton);
        getContentPane().add(listScrollPane, BorderLayout.CENTER);
        getContentPane().add(buttonPane, BorderLayout.SOUTH);
    }

    /* Χειρισμός συμβάντων για το κουμπί delButton. Διαγράφεται όποιο κουμπί είναι επιλεγμένο
    στην λίστα. Αν η λίστα δεν έχει κανένα στοιχείο τότε το κουμπί delButton απενεργοποιείται.
    Αν το όνομα που διαγραφεί είναι το τελευταίο στην σειρά της λίστας τότε ως επιλεγμένο (όνομα)
    τίθεται το αμέσως προηγούμενο. */
    class delListener implements ActionListener {
        public void actionPerformed(ActionEvent e) {
            int index = list.getSelectedIndex();
            listModel.remove(index);
            int size = listModel.getSize();
            if (size == 0) {
                delButton.setEnabled(false);
            } else {
```

```
                if (index == listModel.getSize())
                    index--;
                list.setSelectedIndex(index);
            }
        }
    }

    /* Χειρισμός συμβάντων για το κουμπί inButton. Αν δεν υπάρχει κανένα στοιχείο στην λίστα
    ή το επιλεγμένο στοιχείο είναι το τελευταίο, τότε προστίθεται το νέο όνομα ως καινούργιο στοιχείο
    και επιλέγεται. Διαφορετικά, εισάγεται ένα στοιχείο μετά το τρέχον και επιλέγεται. */
    class inListener implements ActionListener {
        public void actionPerformed(ActionEvent e) {
            int index = list.getSelectedIndex();
            int size = listModel.getSize();
            if (index == -1 || (index+1 == size)) {
                listModel.addElement(friendName.getText());
                list.setSelectedIndex(size);
            } else {
                    listModel.insertElementAt(friendName.getText(),index+1);
                    list.setSelectedIndex(index+1);
            }
        }
    }

}
```

*Πρόγραμμα 12.2 – AppJList.java*

- source code

Έγγραφο html

```
<APPLET CODE = "AppJList.class" WIDTH = 330 HEIGHT = 200>
</APPLET>
```

*appjlistviewer.html*

Το παραπάνω πηγαίο αρχείο μετά την μεταγλώττισή του, θα παράγει τα τρία αρχεία AppJList.class, AppJList$delListener.class, AppJList$inListener.class. Τα δύο τελευταία παράγονται από τις εσωτερικές κλάσεις της μικροεφαρμογής.

---

### 12.4.6 Μενού (κλάση Menu)

---

Τα μενού είναι συστατικά που παρέχουν την δυνατότητα στον χρήστη να διαλέξει μία από τις πολλές επιλογές τους. Τα μενού σώζουν χώρο του GUI του προγράμματος, αφού δεν εμφανίζουν όλες τις

επιλογές τους σε αυτό όπως οι λίστες. Για να γίνουν ορατές οι επιλογές ενός μενού πρέπει ο χρήστης να κάνει κλικ στην μπάρα του μενού.

Οι σημαντικότερες κλάσεις που είναι υπεύθυνες για την δημιουργία ενός μενού είναι οι `JMenuBar` , `JMenu`, `JMenuItem`, `JSeperator`.

Η κλάση `JMenubar` είναι η βάση για την δημιουργία ενός μενού. Σε αυτήν προστίθενται αντικείμενα της κλάσης `JMenu` τα οποία δημιουργούν μια αναδυόμενη λίστα από αντικείμενα της `JMenuItem`. Τα αντικείμενα `JMenuItem` είναι οι επιλογές του μενού. Επίσης στην `JMenu` μπορεί να προστεθεί ένα αντικείμενο της κλάσης `JSeperator` το οποίο δημιουργεί μια διαχωριστική γραμμή στο μενού.

Οι πιο σημαντικοί δημιουργοί των παραπάνω κλάσεων είναι:

- `JMenuBar()`: Δημιουργεί μια μπάρα μενού.
- `JMenu()`: Δημιουργεί ένα αντικείμενο `JMenu` χωρίς κείμενο.
- `JMenu(String str)`: Δημιουργεί ένα αντικείμενο `JMenu` με κείμενο `str`.
- `JMenuItem()`: Δημιουργεί ένα αντικείμενο `JMenuItem` χωρίς κείμενο.
- `JMenuItem(String str)`: Δημιουργεί ένα αντικείμενο `JMenuItem` με κείμενο `str`.
- `JSeperator()`: Δημιουργεί ένα αντικείμενο της κλάσης `JSeperator`.

Αξιομνημόνευτες μέθοδοι για τις παραπάνω κλάσεις είναι:

Για την `JMenuBar`:

- `add(JMenu menu)`: Προσθέτει ένα αντικείμενο `JMenu` στο τέλος της μπάρας.
- `setJMenuBar(JMenuBar menubar)`: Αυτή η μέθοδος θέτει την μπάρα `menubar` σε ένα αντικείμενο των κλάσεων `JApplet`, `JDialog`, `JFrame`. Δεν ανήκει στην κλάση `JMenuBar`, αλλά υπάρχει στις κλάσεις `JApplet`, `JDialog`, `JFrame`.

Για την `JMenu`:

- `add(JMenuItem menuitem)`: Επισυνάπτει το αντικείμενο `menuitem` στο τέλος του μενού.
- `addMenuListener(MenuListener menul)`: Συσχετίζει ένα μενού με τον ακροατή συμβάντων `MenuListener` που είναι υπεύθυνος για γεγονότα που προκαλούν τα μενού. Οι μέθοδοι του ακροατή συμβάντων είναι οι `menuCanceled(MenuEvent e)`, `menuDeselected(MenuEvent e)`, `menuSelected(MenuEvent e)`.
- `addSeperator()`: Επισυνάπτει ένα διαχωριστή στο τέλος του μενού.

Στο παρακάτω παράδειγμα ορίζεται μια μπάρα μενού που έχει δύο αντικείμενα της κλάσης `JMenu`. Το πρώτο αντικείμενό της αποτελείται από 5 επιλογές, ενώ το δεύτερο από μία. Όταν ο χρήστης επιλέγει κάποιο στοιχείο, τότε στην περιοχή κειμένου υπάρχει αναφορά της ταυτότητάς του. Το πρώτο στοιχείο του πρώτου μενού μπορεί να επιλεχθεί και με τον συνδυασμό των κουμπιών alt+1.




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

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

public class AppMenu extends JApplet implements ActionListener{
    JTextArea textarea;
    JScrollPane scrollPane;
    JMenuItem it1,it2,it3,it4,it5,it6;

     /* Στην γραφική διασύνδεση της μικροεφαρμογής χρησιμοποιείται ο προεπιλεγμένος
     διαχειριστής διάταξης BorderLayout (διάταξη περιγράμματος). */
    public void init() {
        JMenuBar menuBar;
        JMenu menu1,menu2;
        textarea = new JTextArea(5, 30);
        textarea.setEditable(false);
        scrollPane = new JScrollPane(textarea);
        getContentPane().add(scrollPane, BorderLayout.CENTER);

         // Δημιουργία μενού μπάρας
        menuBar = new JMenuBar();
        setJMenuBar(menuBar);

         // Δημιουργία του πρώτου μενού
        menu1 = new JMenu("Πρώτο μενού");
        menuBar.add(menu1);

          // Δημιουργία στοιχείων μενού και πρόσθεση τους στο κυρίως μενού
        it1 = new JMenuItem("Πρώτη επιλογή");
        it1.addActionListener(this);
        it1.setAccelerator(KeyStroke.getKeyStroke(
                    KeyEvent.VK_1, ActionEvent.ALT_MASK));
        menu1.add(it1);
        it2 = new JMenuItem("Δεύτερη επιλογή");
        it2.addActionListener(this);
        menu1.add(it2);
        it3 = new JMenuItem("Τρίτη επιλογή");
        it3.addActionListener(this);
        menu1.add(it3);
        menu1.addSeparator();
        it4 = new JMenuItem("Τέταρτη επιλογή");
        it4.addActionListener(this);
        menu1.add(it4);
        it5 = new JMenuItem("Πέμπτη επιλογή");
        it5.addActionListener(this);
        menu1.add(it5);

         // Δημιουργία του δεύτερου μενού
        menu2 = new JMenu("Δεύτερο μενού");
        menuBar.add(menu2);
        it6 = new JMenuItem("Πρώτη επιλογή");
        it6.addActionListener(this);
        menu2.add(it6);
    }

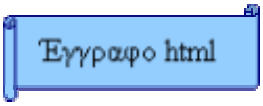
     /* Χειρισμός συμβάντων μενού. Το μήνυμα που εμφανίζεται στην περιοχή κειμένου προσδιορίζει
     το στοιχείο μενού που επιλέχθηκε. */
    public void actionPerformed(ActionEvent e){
        Object source = e.getSource();
```

```
        if(source==it1)
            textarea.setText("Πρώτο μενού-πρώτο στοιχείο ");
        if(source==it2)
            textarea.setText("Πρώτο μενού-δεύτερο στοιχείο ");
        if(source==it3)
            textarea.setText("Πρώτο μενού-τρίτο στοιχείο ");
        if(source==it4)
            textarea.setText("Πρώτο μενού-τέταρτο στοιχείο ");
        if(source==it5)
            textarea.setText("Πρώτο μενού-πέμπτο στοιχείο ");
        if(source==it6)
            textarea.setText("Δεύτερο μενού-πρώτο στοιχείο ");
    }

}
```

*Παράδειγμα 12.3 – AppMenu.java*

- source code




```
<APPLET CODE = "AppMenu.class" WIDTH = 350 HEIGHT = 150>
</APPLET>
```

*appmenuviewer.html*

---

### 12.4.7. Γραμμή εξέλιξης (κλάση JProgressBar)

---

Μερικές φορές ορισμένες εργασίες σε ένα πρόγραμμα χρειάζονται ορισμένη ώρα για να ολοκληρωθούν. Μια φιλική ένδειξη προς το χρήστη είναι να παρέχει το πρόγραμμα ένδειξη για το ποσοστό της εργασίας που έχει ολοκληρωθεί. Αυτό επιτυγχάνεται με τις ***γραμμές εξέλιξης*** που δημιουργούνται από την κλάση `JProgressBar`. Στις γραμμές εξέλιξης προσδιορίζονται δύο τιμές που δηλώνουν τα σημεία που αρχίζει και τελειώνει η επιθυμητή εργασία. Οι κυριότεροι δημιουργοί της `JProgressBar` είναι:

- `JProgressBar()`: Δημιουργεί μια οριζόντια γραμμή εξέλιξης
- `JProgrssBar(int x)`: Δημιουργεί μια γραμμή εξέλιξης με προσαναταλισμό που προσδιορίζεται από το όρισμα `x` το οποίο μπορεί να πάρει τις τιμές `JProgressBar.VERTICAL`, `JProgressBar.HORIZONTAL` για την κατακόρυφη και την οριζόντια περίπτωση αντίστοιχα.
- `JProgrssBar(int x, int min, int max)`: Δημιουργεί μια γραμμή εξέλιξης με `min` την αρχική και `max` την τελική τιμή της. Ο προσανατολισμός καθορίζεται από το όρισμα `x`.

Χρήσιμες μέθοδοι της κλάσης `JProgressBar` είναι:

- `setMaximum(int x)`: Θέτει την τελική τιμή της γραμμής εξέλιξης.
- `setMinimum(int x)`: Θέτει την αρχική τιμή της γραμμής εξέλιξης.
- `setOrientation(int x)`: Θέτει τον προσανατολισμό της γραμμής εξέλιξης. Το όρισμα `x` μπορεί να πάρει τις τιμές `JProgressBar.VERTICAL`, `JProgressBar.HORIZONTAL` για κατακόρυφη και οριζόντια κατεύθυνση αντίστοιχα.
- `setStringPainted(boolean bl)`: Αν το όρισμα `bl` είναι **true** θα εμφανίζεται ένα **String** επάνω στην γραμμή εξέλιξης με το ποσοστό της εργασίας που έχει ολοκληρωθεί.
- `setValue(int x)`: Το όρισμα `x` θέτει την τιμή από όπου θα αρχίζει η γραμμή εξέλιξης.

Ακολούθως δίνεται σχηματικά ένα στιγμιότυπο της γραμμής εξέλιξης:




---

### 12.4.8 Μηχανισμός ολίσθησης (κλάση JSlider)

---

Ο ***μηχανισμός ολίσθησης*** επιτρέπει στον χρήστη να θέσει μία αριθμητική τιμή ανάμεσα σε μια μέγιστη και μια ελάχιστη. Με αυτόν τον τρόπο μειώνονται τα λάθη λόγω του περιορισμένου εύρους των τιμών. Ο μηχανισμός ολίσθησης υλοποιείται με την κλάση `JSlider`. Οι σημαντικότεροι δημιουργοί της `JSlider` είναι:

- `JSlider(int min, int max)`: Δημιουργεί ένα μηχανισμό ολίσθησης με `min` ελάχιστη τιμή και `max` μέγιστη.
- `JSlider(int min, int max, int v)`: Ίδια με την παραπάνω αλλά με αρχική τιμή `v`.
- `JSlider(int x, int min, int max, int v)`: Δημιουργεί ένα μηχανισμό ολίσθησης με ελάχιστη τιμή `min`, μέγιστη τιμή `max`, αρχική τιμή `v` και προσανατολισμό `x`. Το όρισμα `x` μπορεί να πάρει τις τιμές `JSlider.VERTICAL`, `JSlider.HORIZONTAL` για κάθετο και οριζόντιο προσανατολισμό αντίστοιχα.

Χρήσιμες μέθοδοι της κλάσης `JSlider` είναι οι παρακάτω:

- `addChangeListener(ChangeListener c)`: Συσχετίζει τον μηχανισμό ολίσθησης με τον ακροατή συμβάντων `ChangeListener` που είναι υπεύθυνος να αντιλαμβάνεται αλλαγές τιμών στον μηχανισμό ολίσθησης. Η μοναδική μέθοδος του ακροατή συμβάντων είναι η `stateChanged(ChangeEvent e)`.
- `getValue()`: Επιστρέφει την τρέχουσα τιμή του μηχανισμού ολίσθησης
- `setMajorTickSpacing(int x)`: Αυτή η μέθοδος θέτει μια απόσταση `x` μεταξύ των κύριων συμβόλων υποδιαίρεσης. Δηλαδή, αν το πεδίο ορισμού του μηχανισμού ολίσθησης είναι από 0 έως 50 και το x=10 τότε τα κύρια σύμβολα υποδιαίρεσης θα υπάρχουν στις τιμές 0,10,20,30,40,50.
- `setMaximun(int max)`: Θέτει την τιμή `max` μέγιστη τιμή του μηχανισμού ολίσθησης.

- `setMinimum(int min)`: Θέτει την τιμή `min` ελάχιστη τιμή του μηχανισμού ολίσθησης.
- `setMinorTickSpacing(int x)`: Θέτει μια απόσταση `x` μεταξύ των δευτερευόντων συμβόλων υποδιαίρεσης.
- `setPaintTicks(boolean bl)`: Αν το όρισμα `bl` είναι **true** θα εμφανίζονται τα σύμβολα υποδιαίρεσης.
- `setPaintLabels(boolean bl)`: Αν το όρισμα `bl` είναι **true** θα εμφανίζεται η αριθμητική τιμή του μηχανισμού ολίσθησης.
- `setValue(int v)`: Θέτει την τιμή `v` ως τρέχουσα τιμή του μηχανισμού ολίσθησης.

Στο ακόλουθο παράδειγμα ορίζονται δύο μηχανισμοί ολίσθησης και ένα πεδίο κειμένου. Για το πεδίο κειμένου υπάρχει αναφορά στο επόμενο κεφάλαιο. Ο πρώτος μηχανισμός ολίσθησης έχει περιοχή τιμών από 0 έως 100 με αρχική τιμή 10. Ο δεύτερος μηχανισμός ολίσθησης έχει περιοχή τιμών από 0 έως 1000 με αρχική τιμή 100. Το πεδίο κειμένου έχει το άθροισμα των τρεχόντων τιμών των μηχανισμών ολίσθησης. Για να επιτευχθεί αυτό το πρόγραμμα υλοποιεί την διασύνδεση `ChangeListener`. Ο ακροατής συμβάντων `ChangeListener` χειρίζεται την αλλαγή των τιμών των μηχανισμών ολίσθησης. Η μέθοδος `String.valueOf(x+y)` χρησιμοποιείται για να μετατρέψει το ακέραιο άθροισμα `x+y` σε τύπο **String**. Επίσης χρησιμοποιείται ο προεπιλεγμένος διαχειριστής διάταξης του ενδιάμεσου υποδοχέα της κλάσης `JApplet` που είναι ο BorderLayout.




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

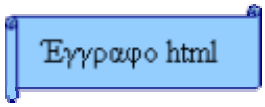
public class AppSlider extends JApplet implements ChangeListener{
     // Ορισμός των συστατικών του προγράμματος
    JSlider slide,slide1;
    JTextField textfield;
    int x=10,y=100;

    public void init(){
        textfield=new JTextField(String.valueOf(x+y),50);
         // Ορισμός χαρακτηριστικών του πρώτου μηχανισμού ολίσθησης
        slide=new JSlider(JSlider.HORIZONTAL,0,100,10);
        slide.setMajorTickSpacing(10);
        slide.setMinorTickSpacing(5);
        slide.setPaintTicks(true);
        slide.setPaintLabels(true);
        slide.addChangeListener(this);
         // Ορισμός χαρακτηριστικών του δεύτερου μηχανισμού ολίσθησης
        slide1=new JSlider(JSlider.HORIZONTAL,0,1000,100);
        slide1.setMajorTickSpacing(100);
        slide1.setMinorTickSpacing(50);
        slide1.setPaintTicks(true);
        slide1.setPaintLabels(true);
        slide1.addChangeListener(this);
         // Τοποθέτηση των συστατικών σε κατάλληλη θέση στον ενδιάμεσο υποδοχέα
        getContentPane().add(slide,BorderLayout.NORTH);
        getContentPane().add(slide1,BorderLayout.CENTER);
        getContentPane().add(textfield,BorderLayout.SOUTH);
    }
```

```
    // Χειρισμός συμβάντων της αλλαγής τιμής των μηχανισμών ολίσθησης
    public void stateChanged(ChangeEvent e){
        if(slide==e.getSource())
        {
            x=slide.getValue();
            textfield.setText(String.valueOf(x+y));
        }
        if(slide1==e.getSource())
        {
            y=slide1.getValue();
            textfield.setText(String.valueOf(x+y));
        }
    }

}
```

*Παράδειγμα 12.4 – AppSlider.java*




```
<APPLET CODE="AppSlider.class" WIDTH=350 HEIGHT=150>
</APPLET>
```

*appsliderviewer.html*

---

### 12.4.9. Συστατικά κειμένου

---

Τα ***συστατικά κειμένου*** εμφανίζουν κείμενο το οποίο μπορεί ο χρήστης επιλεκτικά να το αλλάξει. Η υπερκλάση των συστατικών κειμένου είναι η `JTextComponent` του πακέτου `javax.swing.text`. Τα πιο σημαντικά συστατικά κειμένου δημιουργούνται από τις κλάσεις `JTextArea`, `JTextField`, `JPasswordField`. Οι κλάσεις `JTextArea` και `JTextField` κληρονομούν απευθείας την `JTextComponent`, ενώ η `JPasswordField` κληρονομεί την `JTextField`.

Τα πεδία κειμένου υλοποιούνται από τις κλάσεις `JTextField` και `JPasswordField`. Έχουν μόνο μια γραμμή κειμένου. Η κλάση `JTextField`, αντίθετα με την αντίστοιχη κλάση της στο AWT (TextField), δεν μπορεί να αποκρύψει χαρακτήρες. Για την δουλειά αυτή είναι υπεύθυνη η κλάση `JPasswordField`.

Οι περιοχές κειμένου υλοποιούνται με την κλάση `JTextArea`. Η κλάση αυτή μπορεί να εμφανίζει πολλές γραμμές.

Οι δημιουργοί των παραπάνω συστατικών κειμένου είναι:

- `JTextField()`: Δημιουργεί ένα κενό αντικείμενο της κλάσης `JTextField`.

- `JTextField(String str)`: Δημιουργεί ένα πεδίο κειμένου με το περιεχόμενο του ορίσματος `str`.
- `JTextField(String str, int x)`: Δημιουργεί ένα πεδίο κειμένου με το περιεχόμενο του ορίσματος `str` και μέγεθος `x` στήλες.
- `JPasswordField(String str)`: Δημιουργεί ένα ομότυπο της `JPasswordField` με κείμενο `str`.
- `JPasswordField(String str, int x)`: Δημιουργεί ένα ομότυπο της `JPasswordField` με κείμενο `str` και μέγεθος `x` στήλες.
- `JTextArea(int x, int y)`: Δημιουργεί μια κενή περιοχή κειμένου με `x`, `y` σειρές και στήλες αντίστοιχα.
- `JTextArea(String str)`: Δημιουργεί μια περιοχή κειμένου με κείμενο `str`.
- `JTextArea(String str, int x, int y)`: Δημιουργεί μια περιοχή κειμένου με `x`, `y` σειρές, στήλες και κείμενο `str`.

Μέθοδοι της κλάσης `JTextComponent` που κληρονομούνται από τις υποκλάσεις της είναι:

- `getText()`: Επιστρέφει το κείμενο ενός συστατικού κειμένου.
- `replaceSelection(String str)`: Αντικαθιστά το επιλεγμένο κείμενο ενός συστατικού με το περιεχόμενο του ορίσματος `str`.
- `setEditable(boolean bl)`: Αν το όρισμα `bl` είναι **true** τότε το κείμενο του συστατικού μπορεί να αλλάξει.
- `setEnabled(boolean bl)`: Ενεργοποιεί ή απενεργοποιεί το συστατικό κειμένου ανάλογα με την τιμή του ορίσματος `bl`.
- `setText(String str)`: Θέτει το περιεχόμενο του συστατικού κειμένου.

Χρήσιμες μέθοδοι της κλάσης `JTextField` είναι:

- `addActionListener(ActionListener)`: Συσχετίζει ένα πεδίο κειμένου με τον ακροατή συμβάντων ActionListener. Η μοναδική μέθοδος του ακροατή συμβάντων είναι η `actionPerformed(ActionEvent e)`.
- `setColumns(int x)`: Θέτει σε `x` τον αριθμό των στηλών του πεδίου κειμένου.

Χρήσιμες μέθοδοι της κλάσης `JPasswordField` είναι:

- `getPassword()`: Επιστρέφει έναν πίνακα τύπου **char** με το κείμενο του `JPasswordField`.
- `setEchoChar(char c)`: Θέτει σε `c` τον χαρακτήρα που θα εμφανίζεται στο πεδίο κειμένου, οτιδήποτε και αν πληκτρολογεί ο χρήστης.

Χρήσιμες μέθοδοι της κλάσης `JTextArea` είναι:

- `replaceRange(String str, int x, int y)`: Αντικαθιστά το κείμενο από την θέση `x` έως την θέση `y` με το κείμενο `str`.
- `setColumns(int x)`: Θέτει σε `x` τον αριθμό των στηλών της περιοχής κειμένου.
- `setRows(int x)`: Θέτει σε `x` τον αριθμό των σειρών της περιοχής κειμένου.

Στο παρακάτω πρόγραμμα υπάρχουν δύο πεδία κειμένου, δύο κουμπιά και μια περιοχή κειμένου. Το πρώτο πεδίο κειμένου χρησιμοποιείται για το όνομα του χρήστη, ενώ το δεύτερο για τον κωδικό του. Το κουμπί με τίτλο `submit` χρησιμοποιείται για να συγκρίνει τα δεδομένα που υπάρχουν στα πεδία με τα σωστά και, ανάλογα με την ορθότητα τους, επιστρέφει κάποιο μήνυμα στην περιοχή κειμένου. Στο συγκεκριμένο πρόγραμμα οι μόνες σωστές τιμές είναι username για το όνομα του χρήστη και

kodikos για τον κωδικό του χρήστη. Ό,τι κείμενο εισάγεται στο πεδίο για τον κωδικό εμφανίζεται με αστεράκια. Η εντολή `JTextArea1.setLineWrap(true)` κάνει το κείμενο να αλλάξει σειρά όταν γεμίσει αυτήν που είναι. Η μέθοδος `setActionCommand(String str)` θέτει ένα όνομα εντολής σε ένα συστατικό για γεγονότα που θα συμβούν από αυτό. Αντίθετα, η `getActionCommand()` παίρνει το όνομα της εντολής.




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

public class AppJText extends JApplet implements ActionListener {
    JTextField text1;
    JLabel JLabel1;
    JLabel JLabel2;
    JPasswordField text2;
    JButton JButton1;
    JButton JButton2;
    JTextArea JTextArea1;

    public void init()
    {
        getContentPane().setLayout(null);
        getContentPane().setBounds(0,0,260,210);
        text1=new JTextField("username");
        JLabel1= new JLabel("Ονομα χρήστη");
        JLabel2=new JLabel("Κωδικός χρήστη");
        text2=new JPasswordField("kodikos");
        text2.setEchoChar('*');
        JButton1=new JButton("Submit");
        JButton2=new JButton("Clear");
        JTextArea1=new JTextArea();
        JTextArea1.setLineWrap(true);
        getContentPane().add(text1);
        text1.setBounds(110,12,121,24);
        getContentPane().add(JLabel1);
        JLabel1.setBounds(12,12,95,26);
        getContentPane().add(JLabel2);
        JLabel2.setBounds(12,48,100,21);
        getContentPane().add(text2);
        text2.setBounds(110,48,120,24);
        JButton1.setActionCommand("ok");
        JButton1.addActionListener(this);
        getContentPane().add(JButton1);
        JButton1.setBounds(48,96,76,27);
        JButton2.setActionCommand("clear");
        JButton2.addActionListener(this);
        getContentPane().add(JButton2);
        JButton2.setBounds(144,96,79,26);
        getContentPane().add(JTextArea1);
        JTextArea1.setBounds(12,132,240,72);
    }
```

/* *Στην μέθοδο χειρισμού συμβάντων συγκρίνονται τα ονόματα εντολής γεγονότων των δύο κουμπιών.*

```
        Αν το κουμπί που επιλέχθηκε είναι ίσο με την ετικέτα clear τότε καθαρίζονται τα πεδία κειμένου,
        διαφορετικά συγκρίνονται οι τιμές των πεδίων με τις προεπιλεγμένες τιμές "username" και "kodikos". */
    public void actionPerformed(ActionEvent e){
        String s=e.getActionCommand();
        if (s.equals("ok"))
        {
            if(text1.getText().equals(""))
            {
                JTextArea1.setText("Το πεδίο όνομα χρήστη είναι
                                    κενό. Παρακαλώ συμπληρώστε το.");
                text2.setText("");
            }
            else
            {
                if(text1.getText().equals("username"))
                {
                    if(String.valueOf(text2.getPassword())
                                        .equals("kodikos"))
                        JTextArea1.setText("Το όνομα του χρήστη και
                                                ο κωδικός είναι έγκυρα.");
                    else
                    {
                        JTextArea1.setText("Ο κωδικός είναι λάθος,
                                            παρακαλώ προσπαθήστε ξανά.");
                        text2.setText("");
                    }
                }
                else
                    JTextArea1.setText("Το όνομα του χρήστη είναι
                            λανθασμένο. Παρακαλώ προσπαθήστε ξανά.");
            }
        }
        if (s.equals("clear"))
        {
            text1.setText("");
            text2.setText("");
        }
    }

}
```

*Παράδειγμα 12.5 – AppJText.java*

- source code

Έγγραφο html

```
<APPLET CODE="AppJText.class" WIDTH=260 HEIGHT=210>
</APPLET>
```

---

### 12.4.10 Δέντρο (κλάση JTree)

Σε ένα δέντρο τα δεδομένα εμφανίζονται ιεραρχικά, όπως τα περιεχόμενα του σκληρού δίσκου με τον windows explorer στο λειτουργικό σύστημα Windows98. Ένα δέντρο δημιουργείται από την κλάση `JTree`. Ένα αντικείμενο της `JTree` δεν περιέχει πραγματικά τα δεδομένα, αλλά περιέχει μια άποψη από αυτά. Ο προσανατολισμός του συστατικού `JTree` (δέντρο) συνήθως είναι κάθετος. Κάθε σειρά από αυτό το συστατικό περιέχει ένα στοιχείο δεδομένων που καλείται κόμβος. Κάθε δέντρο έχει μια ρίζα κόμβου από όπου όλοι οι απόγονοι κόμβοι εξαρτώνται. Προεπιλεγμένα, τα δέντρα εμφανίζουν την ρίζα κόμβου. Οι κόμβοι που εξαρτώνται από έναν άλλο κόμβο είναι παιδιά του κόμβου αυτού ο οποίος λέγεται κλάδος (branch node), ή πατέρας τους. Οι κόμβοι που δεν έχουν απογόνους λέγονται φύλλα (leaf).

Η λειτουργία ενός αντικειμένου της κλάσης `JTree` είναι απλή. Κάνοντας διπλό κλικ σε ένα κόμβο εμφανίζονται οι επόμενοι κόμβοι της ιεραρχίας. Αν γίνει πάλι διπλό κλικ στον ίδιο κόμβο οι απόγονοι του κόμβοι σταματούν να είναι ορατοί.

Σε ένα αντικείμενο της κλάσης `JTree` η ρίζα του και οι υπόλοιποι κόμβοι προστίθενται με την κλάση `DefaultMutableTreeNode`. Οι κόμβοι προστίθενται σε άλλους κόμβους με την μέθοδο `add(...)`.

Για την ρίζα, τους κλάδους και τα φύλλα υπάρχουν προεπιλεγμένα εικονίδια δίπλα στους κόμβους. Κάθε φορά που γίνεται δίπλο κλικ σε ένα κλάδο αυτός αλλάζει εικονίδιο ανάλογα αν κρύβει ή εμφανίζει τους απογόνους του. Η κλάση που αλλάζει τα προεπιλεγμένα εικονίδια είναι η `DefaultTreeCellRenderer`. Οι μέθοδοι της κλάσης αυτής είναι οι `setLeafIcon(Icon i)`, `setOpenIcon(Icon i)`, `setClosedIcon(Icon i)` που θέτουν τα εικονίδια σε ένα φύλλο, σε έναν κλάδο που εκτείνεται και σε έναν κλάδο που συρρικνώνεται αντίστοιχα. Η μέθοδος που ενεργοποιεί την αλλαγή εικονιδίων στο δέντρο είναι η `setCellRender(DefaultTreeCellRenderer d)`.

Οι δημιουργοί των κλάσεων `JTree` και `DefaultMutableTreeNode` είναι:

- `JTree(TreeNode tree)`: Δημιουργεί ένα δέντρο και σαν ρίζα έχει ένα αντικείμενο την κλάσης `TreeNode`. Η κλάση `DefaultMutableTreeNode` υλοποιεί τη διασύνδεση `MutableTreeNode`, η οποία κληρονομεί τη διασύνδεση `TreeNode`.
- `DefaultMutableTreeNode(Object obj)`: Δημιουργεί ένα κόμβο δέντρου.

Χρήσιμες μέθοδοι των παραπάνω κλάσεων είναι:

Για την `JTree`:

- `addTreeSelectionListener(TreeSelectionListener tree)`: Προσθέτει τον ακροατή συμβάντων `TreeSelectionListener` που είναι υπεύθυνος για την επιλογή ενός κόμβου.
- `getLastSelectedPathComponent()`: Επιστρέφει τον τελευταία επιλεγμένο κόμβο. Ο τύπος επιστροφής είναι Object.
- `getSelectionMode()`: Η μέθοδος αυτή ανήκει στην κλάση `ThreeSelectionModel` και επιστρέφει ένα ακέραιο. Η τιμές του ακέραιου μπορεί να είναι `SINGLE_TREE_SELECTION` και `CONTIGUOUS_THREE_SELECION`. Η πρώτη τιμή επιτρέπει μόνο την επιλογή ενός κόμβου ενώ η δεύτερη την πολλαπλή επιλογή.

- `getSelectionModel()`: Επιστρέφει το μοντέλο (TreeSelectionModel) που είναι υπεύθυνο για τον έλεγχο της επιλογής των κόμβων.
- `getSelectionPath()`: Επιστρέφει τον κατάλογο του πρώτου επιλεγμένου κόμβου.
- `getSelectionPaths()`: Επιστρέφει τους καταλόγους των επιλεγμένων κόμβων.
- `setSelectionMode(int x)`: Είναι η αντίθετη της μεθόδου `getSelectionMode()`.
- `setSelectionModel(TreeSelectionModel tree)`: Θέτει το μοντέλο που είναι υπεύθυνο για τον έλεγχο της επιλογής των κόμβων.
- `setSelectionPath(ThreePath tree)`: Θέτει τον κατάλογο `tree` σε έναν κόμβο.
- `setSelectionPaths(ThreePath[] tree)`: Θέτει τον πίνακα καταλόγων `tree` σε ορισμένους κόμβους.

Για την `DefaultMutableTreeNode`

- `getParent()`: Επιστρέφει τον πατρικό κόμβο ενός κόμβου.
- `getRoot()`: Επιστρέφει την ρίζα του δέντρου.
- `isLeaf()`: Επιστρέφει την τιμή **true** αν ο κόμβος είναι φύλλο.

---

# 12.5 Ενδιάμεσοι υποδοχείς (Intermediate Containers)

---

Σε αυτή την ενότητα θα αναλυθούν οι ενδιάμεσοι υποδοχείς στους οποίους τοποθετούνται όλα τα συστατικά. Οι υποδοχείς που θα σχολιασθούν είναι οι πάνελ, πάνελ κύλισης, πάνελ καρτέλλας και μπάρα εργαλείων.

---

### 12.5.1 Πάνελ (κλάση JPanel)

---

Η κλάση `JPanel` είναι ο πιο ευέλικτος και ο πιο συχνά χρησιμοποιούμενος ενδιάμεσος υποδοχέας. Η `JPanel` μπορεί να χρησιμοποιήσει οποιονδήποτε διαχειριστή διάταξης. Προεπιλεγμένα, η `JPanel` δεν χρωματίζει τίποτα εκτός από το παρασκήνιο (background) και έχει σαν διαχειριστή διάταξης την διάταξη ροής (κλάση `FlowLayout`).

Οι βασικοί δημιουργοί της κλάσης `JPanel` είναι:

- `JPanel()`: Δημιουργεί ένα αντικείμενο της κλάσης `JPanel`.
- `JPanel(LayoutManager layout)`: Δημιουργεί ένα αντικείμενο της κλάσης `JPanel` με διαχειριστή διάταξης το αντικείμενο `layout`.

Τα συστατικά προστίθενται στον ενδιαμέσο υποδοχέα με την μέθοδο `add(…)`. Μέσα στην μέθοδο αυτή μπορεί να προσδιοριστεί και ο προσανατολισμός των συστατικών.

Π.χ.

…

```
JButton button=new JButton();

pane.add(button,BorderLayout.NORTH);
```

…

Η μέθοδος `setLayout(LayoutManaget layout)` θέτει τον διαχειριστή διάταξης σε ένα ομότυπο της `JPanel`. Ένα αντικείμενο της κλάσης `JPanel` όταν δημιουργείται είναι αδιαφανές. Για να γίνει διαφανές χρησιμοποιείται η μέθοδος `setOpaque(boolean bl)` με τιμή ορίσματος **false**.

---

### 12.5.2 Πάνελ κύλισης (κλάση JScrollPane)

---

Ο ενδιάμεσος υποδοχέας κύλισης υλοποιείται με την κλάση `JScrollPane`. Χρησιμοποιείται όταν ένα συστατικό που θέλουμε να βάλουμε σε ένα πρόγραμμα καταλαμβάνει περισσότερη από την προσφερόμενη επιφάνεια.

Ένα αντικείμενο της κλάσης `JScrollPane` έχει μπάρες κύλισης στις άκρες του και έτσι είναι δυνατή η εμφάνιση όλων των σημείων του συστατικού που περικλείει. Βασικά η `JScrollPane` αποτελείται από τα στοιχεία δύο κλάσεων της `JScrollBar` και της `JViewport`. Η κλάση `JViewport` είναι υπεύθυνη για το τι εμφανίζεται στην οθόνη όταν μετακινούνται οι λίστες κύλισης. Οι δημιουργοί της κλάσης `JScrollPane` είναι:

- `JScrollPane()`: Δημιουργεί ένα άδειο αντικείμενο της κλάσης `JScrollPane` με τις κυλιόμενες μπάρες να εμφανίζονται όταν χρειάζονται.
- `JScrollPane(int x, int y)`: Δημιουργεί ένα άδειο αντικείμενο της κλάσης `JScrollPane` όπου με `x`, `y` ορίζεται η κατάσταση των μπαρών. Το όρισμα `x` αναφέρεται στην οριζόντια μπάρα και μπορεί να πάρει τις τιμές `HORIZONTAL_SCROLLBAR_ALWAYS`,`HORIZONTAL_SCROLL_NEVER`, `HORIZONTAL_SCROLLBAR_AS_NEEDED` που αντιστοιχούν στην εμφάνιση της μπάρας, πάντα, όταν χρειάζεται και ποτέ αντίστοιχα. Τα ίδια ισχύουν και για το όρισμα `y` που ισχύει για τον κατακόρυφο άξονα, στην θέση όμως της λέξης `HORIZONTAL` μπαίνει η `VERTICAL`.

Χρήσιμες μέθοδοι είναι :

- `getHorizontalScrollBarPolicy()`: Επιστρέφει έναν ακέραιο με την τρέχουσα κατάσταση της οριζόντιας μπάρας.
- `getVerticalScrollBarPolicy()`: Επιστρέφει έναν ακέραιο με την τρέχουσα κατάσταση της κάθετης μπάρας.
- `getViewport()`: Επιστρέφει ένα αντικείμενο `JViewport`. Δηλαδή ό,τι μπορεί να μετακινείται από τις μπάρες κύλισης.
- `setHorizontalScrollBarPolicy(int x)`: Το όρισμα `x` θέτει την κατάσταση της οριζόντιας μπάρας.
- `setLayout(LayoutManager layout)`: Θέτει τον διαχειριστή διάταξης.
- `setVerticalScrollBarPolicy(int x)`: Το όρισμα `x` θέτει την κατάσταση της κάθετης μπάρας. Οι πιθανές τιμές του `x` είναι `VERTICAL_SCROLLBAR_AS_NEEDED`, `VERTICAL_SCROLLBAR_ALWAYS`, `VERTICAL_SCROLLBAR_NEVER` για εμφάνιση της μπάρας όταν χρειάζεται, πάντοτε και ποτέ αντίστοιχα.

Στο παρακάτω παράδειγμα ορίζεται ένα πάνελ κύλισης πάνω στο οποίο προστίθεται μια περιοχή κειμένου είκοσι γραμμών και πενήντα στηλών. Επειδή η περιοχή κειμένου είναι μεγαλύτερη σε μέγεθος από το αντικείμενο της κλάσης `JScrollPane` (το οποίο έχει μέγεθος 300x100 pixels), όταν εκτελείται το πρόγραμμα εμφανίζονται οι μπάρες κύλισης σε αυτό. Η περιοχή κειμένου προστίθεται στο πάνελ κύλισης με την εντολή `scrollpane.getViewport().add(textarea)`.

Πηγαίο αρχείο

```
import java.awt.*;
import javax.swing.*;

public class AppJScrollPane extends JApplet
{
    JScrollPane scrollpane;
    JTextArea textarea;

    public void init()
    {
        scrollpane=new JScrollPane();
        textarea=new JTextArea(20,50);
        getContentPane().add(scrollpane);
        scrollpane.setBounds(0,0,300,100);
        scrollpane.getViewport().add(textarea);
    }

}
```

*Παράδειγμα 12.6 – AppJScrollPane.java*

- source code

Έγγραφο html

```
<APPLET CODE = "AppJScrollPane.class" WIDTH = 300 HEIGHT= 100>
</APPLET>
```

---

### 12.5.3. Tabbed Pane (κλάση JTabbedPane)

---

Με την κλάση `JTabbedPane` διαφορετικά συστατικά μοιράζονται τον ίδιο χώρο. Ο χρήστης μπορεί να επιλέξει το επιθυμητό συστατικό επιλέγοντας την κατάλληλη καρτέλα tab. Δηλαδή, ανάλογα με το πόσα tabs έχει το αντικείμενό της `JTabbedPane` είναι σαν να υπάρχουν τόσες διαφορετικές επιφάνειες οι οποίες μπορούν να εναλλάσσονται.

Οι δημιουργοί της `JTabbedPane` είναι:

- `JTabbedPane()`: Δημιουργεί ένα άδειο αντικείμενο της κλάσης `JTabbedPane`.
- `JTabbedPane(int x)`: Δημιουργεί ένα αντικείμενο της κλάσης `JTabbedPane` όπου τα tabs έχουν τον προσανατολισμό του ορίσματος `x`. Το `x` μπορεί να πάρει τις τιμές `TOP`, `BOTTOM`, `LEFT`, `RIGHT`.

Χρήσιμες μέθοδοι της `JTabbedPane` είναι:

- `add(String str, Component c)`: Προσθέτει ένα συστατικό `c` σε μια καρτέλα tab με τίτλο `str`.
- `getSelectedIndex()`: Επιστρέφει μια ακέραια τιμή που δείχνει πιο tab είναι επιλεγμένο.
- `setSelectedIndex(int x)`: Θέτει ποίο tab θα είναι επιλεγμένο.

Ένα σημείο προσοχής είναι ότι σε κάθε tab μπορεί να υπάρχει μόνο ένα συστατικό το οποίο και καταλαμβάνει όλο το χώρο. Για να οριστούν περισσότερα συστατικά σε ένα tab αυτά πρέπει να προστεθούν σε ένα αντικείμενο της κλάσης JPanel και μετά αυτό το αντικείμενο να προστεθεί στο συγκεκριμένο tab.

Στο ακόλουθο πρόγραμμα ο ενδιάμεσος υποδοχέας είναι ένα αντικείμενο της κλάσης `JTabbedPane` με τρία tabs. Στο πρώτο και στο δεύτερο tab υπάρχει από μία περιοχή κειμένου. Στο τρίτο tab υπάρχει ένα αντικείμενο της κλάσης `JPanel` το οποίο περιέχει τρία συστατικά , δύο κουμπιά και μια περιοχή κειμένου. Το τρίτο tab γίνεται προεπιλεγμένο με την εντολή `tabbed1.setSelectedIndex(2)`. Όπως φαίνεται η αρίθμηση των tabs αρχίζει από τον αριθμό μηδέν.




```
import java.awt.*;
import javax.swing.*;

public class AppJTabbed extends JApplet
{
    JTabbedPane tabbed1;
    JPanel pane;
    JTextArea textarea1;
```

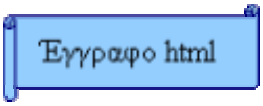
```
    JTextArea textarea2;
    JTextField textfield1;
    JButton button1;
    JButton button2;

    public void init()
    {
        tabbed1=new JTabbedPane();
        pane=new JPanel();
        textarea1=new JTextArea();
        textarea2=new JTextArea();
        textfield1=new JTextField();
        textarea1.setLineWrap(true);
        textarea2.setLineWrap(true);
        button1=new JButton("Button");
        button2=new JButton("Button2");
        getContentPane().setLayout(null);
        getContentPane().add(tabbed1);
        tabbed1.setBounds(0,0,300,100);
        tabbed1.add("first tab",textarea1);
        tabbed1.add("second tab",textarea2);
        pane.setLayout(new GridLayout(2,2));
        tabbed1.add("third tab",pane);
        pane.add(button1);
        pane.add(button2);
        pane.add(textfield1);
        tabbed1.setSelectedIndex(2);
    }

}
```

*Παράδειγμα 12.7 – AppJTabbed.java*

- source code




```
<APPLET CODE="AppJTabbed.class" WIDTH=300 HEIGHT=100>
</APPLET>
```

*appjtabbedviewer.html*

---

### 12.5.4 Μπάρα εργαλείων (κλάση JToolBar)

---

Η μπάρα εργαλείων είναι μια μπάρα που μπορεί να περιέχει περισσότερα από ένα συστατικά. Συνήθως περιέχει κουμπιά και μπορεί να είναι γραμμή ή στήλη. Ο χρήστης μπορεί να μετακινήσει τις μπάρες εργαλείων σε οποία πλευρά του προγράμματος θέλει, ακόμα και εκτός της επιφάνειας του προγράμματος.

Η μπάρα εργαλείων υλοποιείται με την κλάση `JToolBar`. Για να γίνεται σωστά η μετακίνηση της μπάρας πρέπει αυτή να βρίσκεται σε υποδοχέα που χρησιμοποιεί τον διαχειριστή διάταξης BorderLayout.

Οι δημιουργοί της `JToolBar` είναι:

- `JToolBar()`: Δημιουργεί μια μπάρα εργαλείων.
- `JToolBar(int x)`: Δημιουργεί μια μπάρα εργαλείων με προσανατολισμό που δίνεται από το όρισμα `x` που μπορεί να πάρει τις τιμές `HORIZONTAL`, `VERTICAL`.

Χρήσιμες μέθοδοι είναι:

- `addSeperator()`: Επισυνάπτει ένα προεπιλεγμένο διαχωριστή στο τέλος της μπάρας εργαλείων.
- `getOrientation()`: Επιστρέφει έναν ακέραιο που δείχνει τον προσανατολισμό της μπάρας εργαλείων.
- `setOrientation(int x)`: Θέτει τον προσανατολισμό της μπάρας εργαλείων με το όρισμα `x` το οποίο μπορεί να πάρει τις τιμές `HORIZONTAL`, `VERTICAL`.

---

# 12.6 Κύριοι υποδοχείς (Top-level container)

---

Η βάση των προγραμμάτων GUI με Swing είναι οι κύριοι υποδοχείς. Πάνω σε αυτούς μπαίνουν οι ενδιάμεσοι υποδοχείς με τα συστατικά. Οι κλάσεις των κύριων υποδοχών είναι οι `JFrame`, `JDialog`, `JApplet`. Οι κύριοι υποδοχείς έχουν την δυνατότητα να περιέχουν και άλλους κύριους υποδοχείς. Οι κύριοι υποδοχείς έχουν, προεπιλεγμένα, έναν ενδιάμεσο υποδοχέα όταν ορίζονται. Δηλαδή, μπορεί σε ένα πρόγραμμα να χρησιμοποιηθεί κατευθείαν η μέθοδος `getContentPane().add(Component c)` χωρίς να έχει προηγηθεί η μέθοδος που θέτει ένα ενδιάμεσο υποδοχέα (`setContentPane(...)`). Στην περίπτωση που επιθυμούμε την ακριβή τοποθέτηση των συστατικών σε έναν ενδιάμεσο υποδοχέα με την μέθοδο `setBounds(...)` πρέπει πρώτα να απενεργοποιούμε τον προεπιλεγμένο διαχειριστή διάταξης του, διαφορετικά δεν θα αποδίδεται σωστά το αποτέλεσμα. Η απενεργοποίηση των διαχειριστών διάταξης γίνεται με την εντολή `getContentPane().setLayout(null)`.

---

### 12.6.1 Πλαίσια (JFrame)

---

Τα πλαίσια στο Swing υλοποιούνται με την κλάση `JFrame`. Ένα αντικείμενο της `JFrame` έχει περιθώρια, τίτλο και κουμπιά για το κλείσιμο, την ελαχιστοποίηση και μεγιστοποίηση του παραθύρου. Δημιουργοί της κλάσης `JFrame` είναι:

- `JFrame()`: Δημιουργεί ένα πλαίσιο το οποίο είναι αόρατο.
- `JFrame(String str)`: Δημιουργεί ένα αόρατο πλαίσιο με τίτλο `str`.

Η μέθοδος `setVisible(boolean bl)` είναι υπεύθυνη για την εμφάνιση ή μη ενός πλαισίου. Η `setSize(int x, int y)` θέτει σε `x`, `y` pixels το μέγεθος του πλαισίου. Τέλος, η μέθοδος `setTitle(String str)` θέτει σε `str` τον τίτλο ενός πλαισίου.

---

### 12.6.2. Παράθυρα διαλόγου (κλάσεις JDialog, JOptionPane)

---

Τα παράθυρα διαλόγου υλοποιούνται από τις κλάσεις `JDialog` και `JOptionPane`. Κάθε παράθυρο διαλόγου εξαρτάται από ένα πλαίσιο. Έτσι όταν τερματίζεται ένα πλαίσιο το ίδιο συμβαίνει και για το παράθυρο διαλόγου που εξαρτάται από αυτό.

Το παράθυρο διαλόγου που υλοποιείται από την κλάση `JOptionPane` είναι αποκλειστικό, ενώ της `JDialog` μπορεί να μην είναι αποκλειστικό. Αυτό σημαίνει ότι σε ένα πρόγραμμα τα παράθυρα πίσω από το αποκλειστικό παράθυρο αγνοούνται μέχρι αυτό να κλείσει.

Η `JDialog` είναι δευτερεύουσα της Dialog και η `JOptionPane` είναι δευτερεύουσα της `JComponent`.

Η κλάση `JOptionPane` παρέχει υποστήριξη για τυποποιημένα παράθυρα διαλόγου που περιέχουν εικονίδιο, τίτλο και κείμενο. Τα πρότυπα εικονίδιά της αντιστοιχούν στις περιπτώσεις ερώτησης, ενημέρωσης, προειδοποίησης, λάθους και είναι τα ακόλουθα.


ερώτηση


πληροφορία


προειδοποίηση


λάθος

Τα αποκλειστικά παράθυρα διαλόγου της `JOptionPane` υλοποιούνται με μια από τις μεθόδους `showMessageDialog`, `showOptionDialog`, `showConfirmDialog` και `showInputDialog`.

Η πρώτη μέθοδος εμφανίζει μόνο ένα κουμπί (σε ένα παράθυρο διαλόγου) με κείμενο “OK”. Εύκολα καθορίζονται σε αυτήν το εικονίδιο, το μήνυμα και ο τίτλος του παραθύρου.

- `showMessageDialog(Component c, Obgect obj, String str, int x)`: Το όρισμα `c` καθορίζει το πλαίσιο από το οποίο εξαρτάται αυτό το παράθυρο. Το όρισμα `obj` καθορίζει το μήνυμα του παραθύρου, η μεταβλητή `str` ορίζει τον τίτλο του παραθύρου και τέλος ο ακέραιος `x` εμφανίζει ένα από τα παραπάνω εικονίδια. Ο `x` μπορεί να πάρει τις τιμές `ERROR_MESSAGE`, `INFORMATION_MESSAGE`, `WARNING_MESSAGE`, `QUESTION_MESSAGE`.




Ο κώδικας του παραπάνω στιγμιότυπου είναι:

```
JOptionPane pane=new JOptionPane();

pane.showMessageDialog(frame, "The message dialog program", "Dialog",
JOptionPane.INFORMATION_MESSAGE);
```

Η μέθοδος `showOptionDialog` έχει αυξημένη λειτουργικότητα σε σχέση με την προηγούμενη μέθοδο. Η σύνταξη της είναι:

- `showOptionDialog(Component c, Object obj, String str, int x, int y, Icon icon, Object[] obj1, Object obj2)`: Το όρισμα `c` είναι το πλαίσιο από το οποίο εξαρτάται το παράθυρο. Τα ορίσματα `obj`, `str` είναι το κείμενο και ο τίτλος του παραθύρου. Ο ακέραιος `x` μπορεί να πάρει τις τιμές `YES_NO_OPTION`, `YES_NO_CANCEL_OPTION`. Στην πρώτη περίπτωση εμφανίζονται δύο κουμπιά με κείμενο `YES`, `NO`, ενώ στην δεύτερη τρία με κείμενο `YES`, `NO`, `CANCEL`. Το όρισμα `y` είναι υπεύθυνο για το πρότυπο εικονίδιο που εμφανίζεται στο παράθυρο, ενώ η μεταβλητή `icon` εμφανίζει ένα εικονίδιο της αρεσκείας μας. Ο πίνακας `obj1` περιέχει το κείμενο των κουμπιών που θα εμφανιστούν (υπερβαίνοντας τις προεπιλεγμένες τιμές) και η μεταβλητή `obj2` θέτει το επιλεγμένο κουμπί. Η μέθοδος αυτή επιστρέφει έναν ακέραιο που δείχνει το κουμπί που επιλέχτηκε.




Ο κώδικας του παραπάνω στιγμιότυπου είναι:

```
Object[] obj1={"Yes, please", "No, thanks"};

optionpane.showOptionDialog(this, "The option dialog program", "Dialog",
JOptionPane.YES_NO_OPTION, JOptionPane.INFORMATION_MESSAGE, null, obj1, obj1[0]);
```

Η μέθοδος `showConfirmDialog` μοιάζει αρκετά με την παραπάνω μέθοδο και έχει σύνταξη:

- `showConfirmDialog(Component c, Object obj, String str, int x, int y)`: Τα ορίσματα `c`, `obj`, `str` είναι για το πλαίσιο το κείμενο και τον τίτλο του παραθύρου αντίστοιχα. Το όρισμα `x` μπορεί να πάρει τις τιμές `YES_NO_OPTION` και `YES_NO_CANCEL_OPTION`. Η μεταβλητή `y` δείχνει κάποιο από το πρότυπα εικονίδια. Η μέθοδος επιστρέφει έναν ακέραιο που δείχνει ποιο κουμπί επιλέχτηκε.




Ο κώδικας του παραπάνω στιγμιότυπου είναι:

```
optionpane.showConfirmDialog(frame, "The confirm dialog program", "Dialog",
JOptionPane.YES_NO_CANCEL_OPTION, JOptionPane.QUESTION_MESSAGE);
```

Η μέθοδος `showInputDialog` ζητάει από τον χρήστη να θέσει ένα κείμενο στο πεδίο κειμένου που εμφανίζει. Η σύνταξη της είναι:

- `showInputDialog(Component c, Object o, String str, int x)`: Τα ορίσματα είναι ίδια με τις προηγούμενες μεθόδους. Η μέθοδος αυτή επιστρέφει ένα **String** που είναι αυτό που εισάγει ο χρήστης.



Ο κώδικας του παραπάνω στιγμιότυπου είναι:

```
optionpane.showInputDialog(this, "The input dialog program", "Dialog",
JOptionPane.QUESTION_MESSAGE);
```

---

### 12.6.3 Μικροεφαρμογές (κλάση JApplet)

---

Η κλάση `JApplet` είναι δευτερεύουσα κλάση της Applet και χρησιμοποιείται στην δημιουργία μικροεφαρμογών. Με τον ίδιο τρόπο που δημιουργείται μια μικροεφαρμογή με την κλάση `Applet`, δημιουργείται και με την κλάση `JApplet`. Δηλαδή, χρησιμοποιούνται οι μέθοδοι της κλάσης `Applet` `init()`, `start()`, `stop()`, `destroy()`, `paint()`. Ο προεπιλεγμένος διαχειριστής διάταξης του ενδιάμεσου υποδοχέα της κλάσης `JApplet` είναι ο BorderLayout.

---

## 12.7 Εμφάνιση και αίσθηση (Look and feel)

---

Η εμφάνιση και η αίσθηση έχουν σχέση με την απεικόνιση του προγράμματος. Δίνει την δυνατότητα σε ένα πρόγραμμα να έχει την εμφάνιση και την αίσθηση του συστήματος. Δηλαδή, κάποιος που χρησιμοποιεί τα λειτουργικά συστήματα Windows98 ή Unix μπορεί να έχει στο πρόγραμμα του στην Java, το στυλ των Windows98 ή Unix αντίστοιχα. Επιπλέον, μπορεί το πρόγραμμα να έχει την αίσθηση και εμφάνιση του Swing, που ονομάζεται Metal.

Υπεύθυνη για την εμφάνιση και αίσθηση ενός προγράμματος είναι η κλάση `UIManager` του πακέτου `javax.swing`. Η επιλογή της εμφάνισης και της αίσθησης ενός προγράμματος επιλέγεται με την μέθοδο `setLookAndFeel(...)` της `UIManager`. Δύο από τα ορίσματα της `setLookAndFeel(...)` είναι οι παρακάτω μέθοδοι:

- `getCrossPlatformLookAndFeelClassName()`: Επιστρέφει την εμφάνιση και την αίσθηση Metal.
- `getSystemLookAndFeelClassName()`: Επιστρέφει την εμφάνιση και την αίσθηση του συστήματος.

Η μέθοδος `setLookAndFeel(...)` μπορεί να προκαλέσει την εξαίρεση `UnsupportedLookAndFellException`. Εν κατακλείδι ο κώδικας που θέτει την αίσθηση και την εμφάνιση ενός προγράμματος είναι:

```
try{
    UIManager.setLookAndFeel(...);
}
```

```
catch(Exception e){
    System.out.println("Error"+e);
}
```

Ακολούθως υπάρχουν δύο στιγμιότυπα του ίδιου προγράμματος, το πρώτο με αίσθηση και εμφάνιση Metal και το δεύτερο με Windows98.






# ΠΑΡΑΡΤΗΜΑ Α

# Χρήσιμες οδηγίες για την εγκατάσταση της Java

## Α.1 Λογισμικό της Java

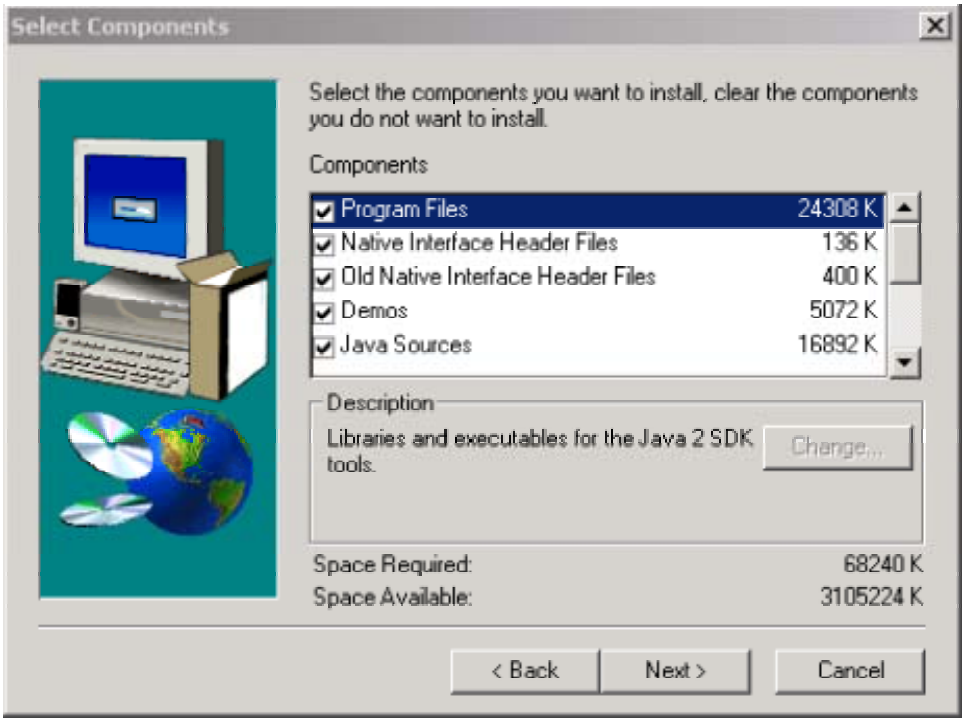
Τα προγράμματα του παρόντος εγχειριδίου έχουν μεταγλωττιστεί και διερμηνευτεί με την πλατφόρμα Java 2 της Sun Microsystems. Προηγούμενες εκδόσεις της πλατφόρμας της Java είναι η 1.0 και 1.1. Το κυρίως λογισμικό της πλατφόρμας Java είναι το JDK (Java Development Kit). Το JDK αποτελείται από βοηθήματα γραμμής εντολών που χρησιμοποιούνται για την δημιουργία, μεταγλώττιση και εκτέλεση του προγράμματος. Το JDK εκτελείται από την γραμμή εντολών δηλαδή από το MS_DOS. Τα προγράμματα του εγχειριδίου έχουν υποστεί επεξεργασία με το JDK έκδοση 1.3 που είναι και η πιο νέα έκδοση του JDK, το οποίο και πρέπει να κατεβάσετε ώστε να μπορείτε να επεξεργάζεστε τα προγράμματα. Η έκδοση 1.3 έχει μέγεθος περίπου 30 MB. Το JDK μπορεί να εγκατασταθεί σε όλες τις εκδόσεις των Windows δηλαδή Win95/98/NT/2000.
Όλες οι εκδόσεις του JDK είναι δωρεάν διαθέσιμες στην ιστοσελίδα της Sun στην διεύθυνση http://java.sun.com/. Στην σελίδα της Sun υπάρχει τεκμηρίωση για κάθε έκδοση του JDK. Επιπλέον υπάρχουν πολλά βοηθητικά προγράμματα, εγχειρίδια, παραδείγματα και ένα forum για συζήτηση με άλλους γνώστες της Java .
Εκτός από την Sun υπάρχουν και άλλες εταιρίες που έχουν δημιουργήσει λογισμικό για την Java. Παρακάτω αναφέρουμε 3 προϊόντα λογισμικού από διαφορετικές εταιρίες τα οποία χρησιμοποιούν την Java με οπτικό περιβάλλον εργασίας. Τα προγράμματα αυτά περιέχουν δικές τους κλάσεις αλλά και τις κλάσεις του JDK της SUN.

- Borland JBuilder
- Microsoft Visual J++
- Symantec Visual Cafe

## Α.2 Εγκατάσταση του Java Development Kit

Αφού κατεβάσετε (download) το JDK 1.3 το κάνετε εγκατάσταση σε έναν οποιοδήποτε κατάλογο. Στην αρχή ο wizard θα εμφανίζει ένα προεπιλεγμένο κατάλογο εγκατάστασης. Αν θέλετε μπορείτε να επιλέξετε έναν άλλο. Στο επόμενο βήμα (Σχήμα Α.1) εμφανίζονται τα αρχεία που πρόκειται να εγκαταστηθούν τα οποία είναι :

- Program files: Περιέχει τις βιβλιοθήκες και τα εκτελέσιμα αρχεία της Java και είναι απαραίτητο να εγκατασταθεί .
- Native interface header files: Δεν χρειάζεται να εγκατασταθεί.
- Old native interface header files: Δεν χρειάζεται να εγκατασταθεί.
- Demos: Περιέχει παραδείγματα εφαρμογών και μικροεφαρμογών.
- Java sources: Περιέχει τα πηγαία αρχεία των κλάσεων συμπεριλαμβανομένης και της πλατφόρμας της Java




*Σχήμα Α.1 Παράθυρο επιλογών κατά την διάρκεια εγκατάστσης της Java*

Στο τέλος της εγκατάστασης μπορείτε να πληκτρολογείστε 'javac –version' στην γραμμή εντολών του MS-DOS για να δείτε ποια έκδοση της Java χρησιμοποιείτε.

Σε περίπτωση που είστε σε διαφορετικό κατάλογο από αυτόν της εγκατάστασης και δοκιμάσετε να εκτελέσετε μια Java εφαρμογή με το εργαλείο **java** θα εμφανιστεί ένα μήνυμα λάθους στην οθόνη όπως 'File not found' στα Win95/98 και 'java' is not recognized as an internal or external command, operable program or batch file' στα Win2000. Αυτό σημαίνει ότι το σύστημα σας δεν μπορεί να βρει τα εργαλεία της `Java` . Αυτό είναι φυσιολογικό αφού δοκιμάσατε να χρησιμοποιήσετε την `Java` από διαφορετικό κατάλογο από αυτόν της εγκατάστασης .

Για να γίνουν διαθέσιμα τα εργαλεία της `Java` πρέπει να ρυθμιστεί σωστά η μεταβλητή path του συστήματος. Η μεταβλητή path περιέχει τους καταλόγους που ψάχνει το MS-DOS για να βρει ένα πρόγραμμα που εισάγεται στην γραμμή εντολών. Τους τρέχοντες κατάλογους path μπορείτε να τους δείτε γράφοντας 'path' στην κατάσταση MS-DOS.

Η εντολή path βρίσκεται στο αρχείο autoexec.bat . Ανοίξτε το αρχείο αυτό με έναν οποιονδήποτε editor και στην γραμμή path προσθέστε τον κατάλογο της γραμμής εργαλείων της `Java`. Σώστε το τροποποιημένο autoexec.bat και επανεκκινήστε το σύστημα σας .

Αν έχετε εγκαταστήσει το JDK στον κατάλογο 'D:\katalogos\java' τότε στο path πρέπει να βάλετε τον κατάλογο 'D:\katalogos\java\bin'. Ένα παράδειγμα της ρύθμισης path στο αρχείο autoexec.bat είναι το παρακάτω:

PATH=C:\WINDOWS;C:\WINDOWS\COMMAND;D:\KATALOGOS\JAVA\BIN

Οι άλλοι δύο κατάλογοι που υπάρχουν έχουν εισαχθεί από το λειτουργικό σύστημα.

Αφού ολοκληρώσετε τα παραπάνω βήματα μπορείτε να μετακινηθείτε στον κατάλογο που υπάρχουν τα πηγαία αρχεία `Java` και με την βοήθεια του μεταγλωττιστή (java) και του διερμηνευτή (javac) να τα τρέξετε.

## Α.3 Τεκμηρίωση του JDK

Μπορείτε να δείτε on-line την τεκμηρίωση της Java πατώντας εδώ (τεκμηρίωση Java 1.3). Η τεκμηρίωση της Java είναι απαραίτητη στον προγραματιστή διότι περιλαμβάνει όλες τις κλάσεις που υπάρχουν, τις μεθόδους και τις μεταβλητές αυτών, καθώς και την ιεραρχία τους. Επίσης περιλαμβάνει και εκείνες τις μεθόδους που κληρονομούν οι κλάσεις από τις υπερκλάσεις τους. Χρήσιμα βοηθήματα σε μορφή για κατέβασμα ή on-line για την εκμάθηση της Java μπορείτε να κατεβάσετε από την

ιστοσελίδα της Sun. Ένα εύχρηστο βοήθημα (η έκδοση είναι στα αγγλικά) είναι το Java Language Specification το οποίο μπορείτε να το κατεβάσετε πατώντας εδώ




---

# Εργαλεία του JDK

---

## B.1 Εργαλεία του JDK

---

Τα πιο βασικά εργαλεία του JDK είναι τα παρακάτω.

- Ο ***μεταγλωττιστής*** (compiler) της Java **javac** .
- Ο ***διερμηνευτής*** (interpreter) της Java **java** .
- Το υπεύθυνο εργαλείο για την εκτέλεση των μικροεφαρμογών ***appletviewer***
- Ο ***αποσφαλματωτής*** (debugger) της Java **jdb**
- Το υπεύθυνο εργαλείο για την δημιουργία documentation **javadoc**

Αν εισαχθούν στην γραμμή εντολών τα εργαλεία **javac**, **java**, **appletviewer** και **javadoc** χωρίς ορίσματα τότε θα εμφανιστούν οι κανόνες συντάξεις και οι επιλογές του κάθε εργαλείου. Οι επιλογές εισάγονται μετά το όνομα του εργαλείου και έχουν ως πρόθεμα τον χαρακτήρα '-'.

---

## B.2 Μεταγλωττιστής javac

---

Ο **javac** χρησιμοποιείται για την μεταγλώττιση ενός πηγαίου αρχείου της Java σε μορφή bytecode .

Ο πηγαίος κώδικας αποθηκεύεται σε ένα αρχείο με κατάληξη '.java' . Ο πηγαίος κώδικας μπορεί να συνταχθεί σε οποιονδήποτε editor (κειμενογράφο) οποιουδήποτε λειτουργικού συστήματος αρκεί να σωθεί σαν αρχείο καθαρού κειμένου δηλαδή σαν κείμενο ASCII , κείμενο DOS.

Ο πηγαίος κώδικας μπορεί να περιέχει περισσότερες από μια κλάσης (που λέγονται εσωτερικές κλάσεις) αλλά μόνο μία από αυτές θα είναι δηλωμένη ως **public** η οποία θα είναι και το όνομα του πηγαίου αρχείου με την κατάληξη .java. Σε αυτήν την περίπτωση ο μεταγλωττιστής δεν παράγει ένα

αρχείο bytecode για το αρχείο του πηγαίου κώδικα αλλά χωριστό αρχείο για κάθε κλάση το οποίο έχει την κατάληξη '.class'.

Π.χ.

```
public class MasterClass{
    …
    class SlaveClass1{
        …
    }
    …
    class SlaveClass2{
        …
    }
}
```

Κατά την μεταγλώττιση του παραπάνω προγράμματος θα παραχθούν τρία αρχεία με την κατάληξη .class που είναι τα MasterClass.class, MasterClass$SlaveClass1.class, MasterClass$SlaveClass2.class.

Η σύνταξη της εντολής για την μεταγλώττιση είναι η εξής :

```
javac <όνομα πηγαίου αρχείου>
```

Π.χ.

```
javac Onoma.java
```

Μπορούν να μεταγλωττιστούν περισσότερα από ένα αρχεία με τη βοήθεια των χαρακτήρων μπαλαντέρ .Ο χαρακτήρας '*' χρησιμοποιείται αντί πολλών χαρακτήρων ενώ ο '?' στη θέση ενός χαρακτήρα . Οι πηγαίοι κώδικες ενός καταλόγου μπορούν να μεταγλωττιστούν με την παρακάτω εντολή :

```
javac *.java
```

Μια πολύ σημαντική επιλογή του μεταγλωττιστή είναι η ***deprecation*** . Σε κάθε νέα έκδοση του JDK της `Java` ορισμένοι μέθοδοι υποβαθμίζονται και την λειτουργία τους την αναλαμβάνουν μέθοδοι άλλων κλάσεων ή και των ίδιων. Ο σκοπός αυτής της αλλαγής είναι η βελτίωση των λειτουργιών της `Java`. Μια υποβαθμισμένη μέθοδος δεν σημαίνει κατά ανάγκη ότι δεν θα λειτουργήσει. Αλλά κατά την μεταγλώττιση του πηγαίου αρχείου θα υπάρχει κάποια προειδοποίηση (warning). Για την εξακρίβωση του ονόματος της υποβαθμισμένης μεθόδου χρησιμοποιείται η επιλογή deprecation η οποία έχει την σύνταξη .

```
javac -deprecation <όνομα πηγαίου αρχείου>
```

Μια άλλη επιλογή είναι η ***verbose*** η οποία παραθέτει το χρόνο που χρειάζεται ο μεταγλωττιστής για να φορτώσει τις συναρτήσεις και τις κλάσεις .

---

## B.3 Διερμηνευτής java

---

Ο διερμηνευτής **java** χρησιμοποιείται για την εκτέλεση των εφαρμογών . Παίρνει σαν όρισμα ένα αρχείο bytecode δηλαδή ένα μεταγλωττισμένο αρχείο από το εργαλείο **javac** . Τα αρχεία που εκτελούνται έχουν κατάληξη '.class' αλλά κατά την εκτέλεση των δεν χρειάζεται να καθορίζεται η επέκταση τους. Η σύνταξη του εργαλείου είναι :

```
java <όνομα κλάσης>
```

Η κλάση που εκτελείται από τον διερμηνευτή **java** μπορεί να προέρχεται από οποιοδήποτε λειτουργικό σύστημα .

---

## B.4 Appletviewer

---

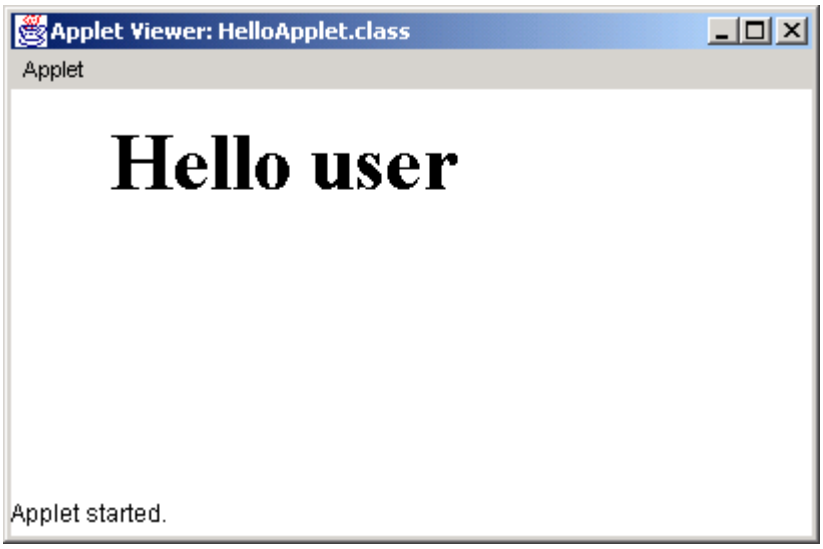
Ο **appletviewer** χρησιμοποιείται για την προβολή των μικροεφαρμογών. Οι μικροεφαρμογές μπορούν να είναι στοιχείο μιας ιστοσελίδας. Η σύνταξη του εργαλείου **appletviewer** είναι :

```
appletviewer <μικροεφαρμογή>
```

Στην θέση του ορίσματος μικροεφαρμογή μπαίνει ένα έγγραφο html το οποίο προδιορίζει τον bytecode (αρχείο με κατάληξη class) της μικροεφαρμογής.

Με το εργαλείο **appletviewer** δεν φαίνεται όλη η ιστοσελίδα αλλά μόνο η μικροεφαρμογή που υπάρχει σε αυτή. Στο παρακάτω παράδειγμα παρουσιάζεται μια μικροεφαρμογή πώς φαίνεται με τον **appletviewer** και πως με έναν φυλλομετρητή (browser). Το παράδειγμα εμφανίζει το μήνυμα 'Hello user' σε παράθυρο 400x200 .

Με τον **appletviewer**




Με έναν φυλλομετρητή



Οι επιλογές στο μενού applet του εργαλείου **appletviewer** είναι :

- `Restart` και `Reload` : Χρησιμοποιούνται για την εκκίνηση της μικροεφαρμογής .
- `Start` και `Stop` : Χρησιμοποιούνται για να καλέσουν απευθείας τις μεθόδους `start()` και `stop()` της μικροεφαρμογής .
- `Clone` : Δημιουργεί ένα δεύτερο αντίγραφο της ίδιας μικροεφαρμογής
- `Tag` : Εμφανίζει τον κώδικα του εγγράφου html.

---

# Β.5 Αποσφαλματωτής jdb

---

Ο αποσφαλματωτής **jdb** βοηθάει στην εύρεση λαθών σε ένα πρόγραμμα `Java` . Ο αποσφαλματωτής χρησιμοποιείται κυρίως στην εύρεση λογικών σφαλμάτων διότι τα άλλα σφάλματα γίνονται αντιληπτά από την μη μεταγλώττιση του πηγαίου κώδικα και με έναν editor διορθώνονται. Στην περίπτωση λογικών σφαλμάτων ο μεταγλωττιστής δεν εμφανίζει ένδειξη λάθους .

Για την χρησιμοποίηση του αποσφαλματωτή πρέπει πρώτα να μεταγλωττιστή το πηγαίο αρχείο με την επιλογή '–g' η οποία προσθέτει πληροφορίες στο αρχείο κλάσης που δημιουργείται. Στη συνέχεια χρησιμοποιείται ο αποσφαλματωτής **jdb** με όρισμα το όνομα της κλάσης που δημιουργήθηκε .Ο αποσφαλματωτής δεν εκτελεί την κλάση απλός εμφανίζει την ένδειξη :

Initializing jdb...

0xac:class(classmethodos)

>

Οι εντολές του αποσφαλματωτή πληκτρολογούνται μετά την ένδειξη '>' . Οι εντολές που θέτουν σημεία διακοπής είναι οι `stop in` και `stop at`. Η πρώτη δέχεται σαν όρισμα το όνομα της κλάσης συνοδευόμενο από το όνομα της μεθόδου, στην πρώτη σειρά της οποίας θέτετε ένα σημείο διακοπής .

Π.χ.

```
stop in MyClass.methodos
```

Η παραπάνω εντολή θέτει ένα σημείο διακοπής στην πρώτη γραμμή της μεθόδου `methodos()`.

Η δεύτερη εντολή διακοπής θέτει σημείο διακοπής σε ένα αριθμό γραμμής του κώδικα. Η σύνταξη της είναι :

```
stop at <όνομα κλάσης><:><αριθμός γραμμής>
```

Π.χ.

```
stop at MyClass:13
```

Μετά την εντολή αυτή εμφανίζεται το παρακάτω μήνυμα το οποίο επιβεβαιώνει το σημείο διακοπής .

```
breakpoint set at MyClass:13
```

Όλα τα σημεία διακοπής εμφανίζονται με την εντολή `clear`. Επιπλέον η εντολή `clear` μπορεί να σβήσει ένα σημείο διακοπής έχοντας σαν όρισμα το ίδιο με αυτό της εντολή stop at .

Η κλάση εκτελείτε με την εντολή `run`. Όταν η κλάση φτάσει σε ένα σημείο διακοπής είναι διαθέσιμες οι ακόλουθες εντολές.

- `list` : Στο σημείο διακοπής εμφανίζει τον πηγαίο κώδικα της γραμμής και αρκετές γραμμές γύρω από αυτή.
- `locals` : Παραθέτει τις τιμές των μεταβλητών σε εκείνη την διακοπή
- `step` : Εκτελεί την επόμενη γραμμή της κλάσης
- `cont` : Εκτελείτε το πρόγραμμα από το σημείο που σταμάτησε

Η αποσφαλμάτωση σταματάει με την εντολή `exit`. Μετά το πέρας της αποσφαλμάτωσης πρέπει να μεταγλωττιστεί ξανά το πηγαίο αρχείο χωρίς την επιλογή ‘-g’ .

---

## B.6 Εργαλείο javadoc

---

Το εργαλείο **javadoc** χρησιμοποιείται για την προσθήκη τεκμηρίωσης σε ένα πρόγραμμα . Η τεκμηρίωση είναι σε μορφή html. Για να δημιουργηθεί η τεκμηρίωση πρέπει να να εισαχθεί ένας συγκεκριμένος τύπος σχολίου που να μπορεί να διαβαστεί από το εργαλείο **javadoc**. Η σύνταξη αυτού του σχολίου είναι :

```
/**

* @ σήμανση(tag) Περιγραφή σήμανσης

*/
```

Τα σχόλια τεκμηρίωσης τοποθετούνται στο πηγαίο αρχείο πριν από το στοιχείο που τεκμηριώνουν . Οι κυριότερες σημάνσεις είναι οι παρακάτω .

- `@author`: Η σήμανση αυτή προσδιορίζει τον συγγραφέα του προγράμματος και για να ενεργοποιείται θα πρέπει να χρησιμοποιείται με την επιλογή '-author' κατά την εκτέλεση του **javadoc** .
- `@version`: Αυτή η σήμανση προσδιορίζει την έκδοση του προγράμματος και όπως και η σήμανση `author` απαιτεί την επιλογή '-version' για να ενεργοποιηθεί .
- `@return` : Η σήμανση αυτή επιστρέφει την μεταβλητή η το αντικείμενο που επιστρέφει η τρέχον μέθοδος .

Created in .pdf from : MoUCLAs corp.2002
http://www.mouclas.tk