3 Προγραμματισμός για το Διαδίκτυο

3.1 Σκοπός ανάπτυξης και χαρακτηριστικά της Java

- Η Java σχεδιάστηκε για την ανάπτυξη προγραμμάτων που τρέχουν σε μικροεπεξεργαστές ενσωματωμένους σε καταναλωτικές συσκευές, πχ τηλεοράσεις, ραδιόφωνα, τηλέφωνα – όχι απαραίτητα σε υπολογιστές. Ο σχεδιαστική αυτή επιλογή υποστηρίζεται με την τεχνική της εκτέλεσης των προγραμμάτων πάνω στην ιδεατή μηχανή Java. Ήδη από το 1995, η ιδεατή μηχανή Java λειτουργεί και μέσα σε περιβάλλον επόπτη ιστοσελίδων (web browser). Ως απόρροια του παραπάνω, τα προγράμματα σε Java μπορούν να χρησιμοποιούνται στο Διαδίκτυο.
  - Ο Java bytecode μπορεί να φορτώνεται δυναμικά από το δίκτυο και να εκτελείται τοπικά στο μηχάνημα που τον φιλοξενεί, χωρίς να απαιτείται μόνιμη εγκατάσταση ή αποθήκευση στο μηχάνημα όπου εκτελείται.

3.2 Εφαρμογίδια

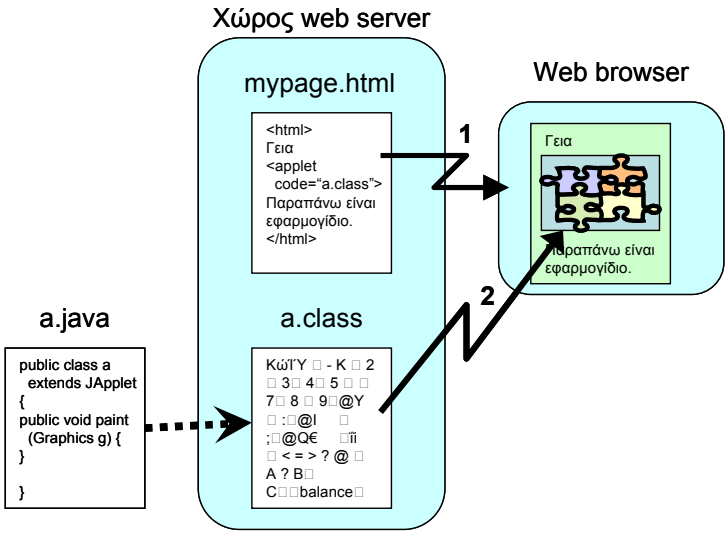
- Τα εφαρμογίδια (applets) είναι προγράμματα που εκτελούνται στο περιβάλλον ενός επόπτη ιστοσελίδων με ενσωματωμένη ιδεατή μηχανή Java. Όλοι οι σύγχρονοι επόπτες, πχ Internet Explorer, Opera, Mozilla έχουν ή μπορούν να ενσωματώσουν μια ιδεατή μηχανή Java με τη μορφή πρόσθετου (plug-in).
  - Στην Εικόνα 43 βλέπετε μια «ελάχιστη» ιστοσελίδα (αρχείο HTML) που περιέχει την ειδική ετικέτα `<applet>`. Αυτή δεσμεύει ένα παράθυρο διαστάσεων 300X200 εικονοστοιχείων (pixels) στην ιστοσελίδα για να προβάλει αυτό που θα εμφανιστεί με την εκτέλεση του αρχείου `SomeApplet.class`[1].

```
<html>
<p>Ακολουθεί ένα εφαρμογίδιο.</p>
<applet code="SomeApplet.class" width=300 height=200>
</applet>
<p>Τέλος της ιστοσελίδας</p>
</html>
```

Εικόνα 43 Ιστοσελίδα που ενσωματώνει ένα εφαρμογίδιο

  - Είναι καλή πρακτική οι ιστοσελίδες που καλούν εφαρμογίδια να δοκιμάζονται (προβάλλονται) σε όλους τους επόπτες ιστοσελίδων που μπορεί να χρησιμοποιηθούν. Επίσης, να ελέγχεται αν ο επόπτης έχει ιδεατή μηχανή που υποστηρίζει Java 2 ή την παλιότερη Java 1.1.

- Η **διαδικασία ανάπτυξης** μιας ιστοσελίδας με εφαρμογίδιο έχει τα παρακάτω βήματα. (i) Φτιάχνουμε τον HTML κώδικα για το στατικό περιεχόμενο της ιστοσελίδας που μπορεί να περιέχει μορφοποιημένο κείμενο, εικόνες, κλπ. Η σχεδίαση της ιστοσελίδας μπορεί να γίνει συγγράφοντας το περιεχόμενο και ενσωματώνοντας τις HTML ετικέτες ή χρησιμοποιώντας κάποιο εργαλείο συγγραφής ιστοσελίδων, όπως τα Macromedia Dreamweaver, Microsoft FrontPage, κά. (ii) Συγγράφουμε την κλάση του εφαρμογίδιου, και την μεταγλωττίζουμε στο ομώνυμο αρχείο bytecode. (iii) Σε κάποιο σημείο της ιστοσελίδας εισάγουμε την ετικέτα `<applet>` με αναφορά προς το μεταγλωττισμένο εφαρμογίδιο. (iv) Μεταφέρουμε τα δύο αρχεία (.html και .class) στον χώρο που έχει ορίσει ο εξυπηρέτης ιστοσελίδων (web server), πχ Apache, IIS, κλπ για την τοποθέτηση διαμοιραζόμενων αρχείων. Από οποιοδήποτε πλέον μηχάνημα μπορούμε να ζητήσουμε μέσα από έναν επόπτη ιστοσελίδων να μας προβάλει την ιστοσελίδα. Η διαδικασία απεικονίζεται στην Εικόνα 44.



Εικόνα 44 Διαδικασία εκτέλεσης εφαρμογίδιου

  - Κατά τη φάση ανάπτυξης του εφαρμογίδιου μπορούμε να χρησιμοποιήσουμε το πρόγραμμα `appletviewer` του J2SDK για να εκτελέσουμε εφαρμογίδια από μια ιστοσελίδα. Έτσι γλιτώνουμε από τη διαδικασία μεταφοράς των αρχείων (upload) στον εξυπηρέτη ιστοσελίδων και την προβολή τους μέσα από τον επόπτη ιστοσελίδων. Σε κάθε περίπτωση, ο κώδικας Java με τη μορφή αρχείου `.class` πρέπει να φορτωθεί μέσα από μια ιστοσελίδα `.html`.
  - Αν δεν χρησιμοποιούμε κάποιο εργαλείο ανάπτυξης λογισμικού, οι εντολές που εκτελούμε είναι:

```
> edit mypage.html          συγγραφή κώδικα ιστοσελίδας
> edit a.java               συγγραφή κώδικα εφαρμογίδιου
> javac a.java              μεταγλώττιση εφαρμογίδιου
> appletviewer mypage.html  προβολή ιστοσελίδας  εκτέλεση εφαρμογίδιου
```

[1] Σημειώστε ότι η ετικέτα `<applet>` είναι depreciated («υπό απόσυρση», ας πούμε) στην έκδοση 4.0 της HTML. Ο ενδεδειγμένος τρόπος για την ενσωμάτωση εφαρμογίδιου σε ιστοσελίδα είναι με την ετικέτα `<object>`, ως εξής:

```
<object codetype="application/java" classid="java:a.class"  width="300" height="200">
```

Δείτε στην τεκμηρίωση της HTML για τους διακόπτες που δέχεται.

- Το εφαρμογίδιο υλοποιείται από μια κλάση που επεκτείνει (κληρονομεί) την κλάση `JApplet` του πακέτου `javax.swing`[2], όπως φαίνεται στην Εικόνα 45.
  - Η κλάση `JApplet` προσφέρει κάποιες τυποποιημένες μεθόδους που εκτελούνται σε συγκεκριμένα χρονικά σημεία της προβολής της ιστοσελίδας στο περιβάλλον του επόπτη ιστοσελίδων. Στην κλάση του εφαρμογίδιου επαναπροσδιορίζουμε τις μεθόδους αυτές και τοποθετούμε στο σώμα τους τις εντολές που θέλουμε να εκτελεστούν κατά τα συγκεκριμένα χρονικά σημεία.
  - Η μέθοδος `init` εκτελείται κατά το φόρτωμα της ιστοσελίδας, ενώ και η `paint` κάθε φορά που ανανεώνεται (refresh) η ιστοσελίδα. Η `paint` απαιτεί ένα αντικείμενο της κλάσης `Graphics` επί του οποίου εφαρμόζονται οι μέθοδοι των γραφικών.

```
import java.awt.*;
import javax.swing.*;

public class AnApplet extends JApplet {

  String sOnoma; //τη χρησιμοποιούν και οι δύο

  public void init() {
    sOnoma = JOptionPane.showInputDialog("Όνομα:");
  }

  public void paint(Graphics g) {
    for (int i = 15; i <= 165; i += 15)
      g.drawString(sOnoma, i, i);
  }
}
```

Εικόνα 45 Ένα απλό εφαρμογίδιο

- Μπορούμε να περάσουμε δεδομένα από την ιστοσελίδα προς το εφαρμογίδιο. Στον HTML κώδικα της ιστοσελίδας που καλεί το εφαρμογίδιο Java τοποθετούμε μία ή περισσότερες ετικέτες `<param>` μέσα στις `<applet>` … `</applet>`. Στο εφαρμογίδιο καλούμε τη μέθοδο `getParameter("paramName")` που αντλεί την τιμή από την αντίστοιχη `<param>` (δείτε Εικόνα 46).

```
<html>
<head>
  <title>Ιστοσελίδα με παραμέτρους</title>
</head>
<body>
<p>Θα εμφανιστεί σε έναν επόπτη που έχει JVM.</p>
<applet code = "AnApplet.class" name = "TestApplet"
        width = "400" height = "300">
  <param name="etos" value="2004">
</applet>
<p>Από πάνω βρίσκεται ο χώρος του εφαρμογίδιου.</p>
</body>
</html>
```

```
import java.awt.*;
import javax.swing.*;

public class AnApplet extends JApplet {
  public void paint(Graphics g) {
    String sEtos = getParameter("etos");
    g.drawString("Χρησιμοποιείται " +sEtos,15,15);
  }
}
```

Εικόνα 46 Παράμετροι εφαρμογίδιου

- Τα αρχεία `.class` των εφαρμογίδιων φορτώνονται δυναμικά από το τοπικό δίκτυο ή το Διαδίκτυο (από τον web server που σερβίρει και τα αρχεία `.html` των ιστοσελίδων) και διερμηνεύονται / εκτελούνται τοπικά, δηλαδή στον υπολογιστή που τρέχει ο επόπτης.
  - Για λόγους ασφάλειας τα εφαρμογίδια, επειδή μπορούν να εκτελούνται μέσα από επόπτες ιστοσελίδων, έχουν μειωμένες δυνατότητες. Για παράδειγμα, δεν μπορούν να προσπελάσουν αρχεία του υπολογιστή-πελάτη και δεν μπορούν να επικοινωνήσουν με άλλον υπολογιστή.
  - Πριν την εκτέλεση, ο bytecode verifier ελέγχει την εγκυρότητα των κλάσεων και επιβάλει περιορισμούς ασφάλειας[3]. Για ασφάλεια, δεν επιτρέπεται ανάγνωση ή εγγραφή σε αρχεία του υπολογιστή που φιλοξενεί το εφαρμογίδιο.
  - Επειδή ο bytecode διερμηνεύεται, τα εφαρμογίδια συνήθως είναι μικρά και όχι έντασης υπολογισμών. Για να επιτύχουμε επιδόσεις, απαιτείται μεταγλώττιση από πηγαίο κώδικα ή bytecode σε γλώσσα μηχανής. Ως ενδιάμεση προσέγγιση υπάρχουν οι μεταγλωττιστές Just In Time που παράγουν κώδικα σε γλώσσα μηχανής και τον εκτελούν. Έτσι αποφεύγεται η επανάληψη της διερμηνείας των μεταγλωττισμένων εντολών. Για περισσότερες πληροφορίες, δείτε στο http://java.sun.com/products/hotspot/.
  - Σε σχέση με μια κλασική εφαρμογή που απαιτεί εγκατάσταση στον υπολογιστή όπου τρέχει, ένα εφαρμογίδιο έχει το πλεονέκτημα ότι αν δημοσιευτεί νέα έκδοσή του στο web server, οι σταθμοί εργασίας ενημερώνονται με τη νέα έκδοση αυτόματα μόλις κάνουν ανανέωση της ιστοσελίδας. Έτσι ο διαχειριστής του συστήματος είναι σίγουρος ότι όλοι οι πελάτες τρέχουν την ίδια (ενημερωμένη) έκδοση του προγράμματος, και δεν χρειάζεται να τους διανείμει τη νέα έκδοση και να επιβλέψει την εγκατάστασή της.
  - Η Java συχνά συγχέεται με τη JavaScript που είναι μια γλώσσα σεναρίων (scripting language) για HTML ιστοσελίδες. Οι δύο γλώσσες παρέχουν δυναμικό περιεχόμενο σε ιστοσελίδες, αλλά πέρα από κάποιες ομοιότητες στη σύνταξη και στο όνομα, δεν έχουν άλλη σχέση μεταξύ τους.

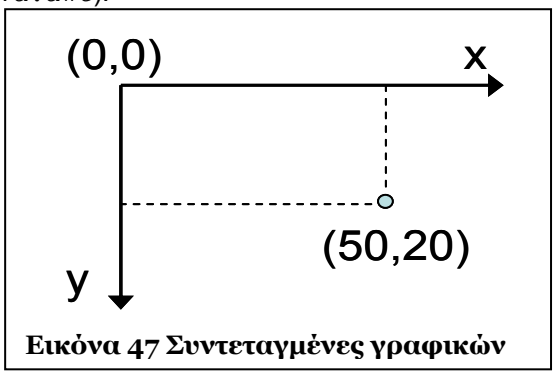
---

[2] Παλιότερα, τα εφαρμογίδια χρησιμοποιούσαν ως υπέρ-κλάση τους την `Applet` του πακέτου `java.Applet` – τώρα η `JApplet` είναι δημοφιλέστερη.

[3] Το πρόσθετο αυτό μέτρο ασφάλειας είναι απαραίτητο, διότι κάποιος θα μπορούσε να τροποποιήσει τον νόμιμο bytecode που φτιάχνει ένας java compiler, για να εκτελέσει ανασφαλείς λειτουργίες στον οικοδεσπότη υπολογιστή. Για να μην αναφέρω την περίπτωση που κάποιος θα έφτιαχνε έναν java compiler που θα επέτρεπε (ή και θα εισήγαγε) ανασφαλείς λειτουργίες στον bytecode που θα έφτιαχνε.

3.3 Γραφικά

- Τα γραφικά στη Java υλοποιούνται από τη βιβλιοθήκη AWT (πακέτο `java.awt`).
  - Στο σύστημα συντεταγμένων της Java, η αρχή των αξόνων, δηλαδή το σημείο (x, y) = (0, 0) βρίσκεται πάνω και αριστερά. Οι διαστάσεις του πλαισίου γραφικών καθορίζονται από τους διακόπτες `width` και `height` της εντολής `applet` της HTML, όπως φαίνεται στην Εικόνα 43.
  - Όλες οι μέθοδοι που εμφανίζουν κάτι, εφαρμόζονται σε κάποιο αντικείμενο `Graphics` – κατά σύμβαση εκείνο που αναφέρει η μέθοδος `paint` του εφαρμογίδιου.



Εικόνα 47 Συντεταγμένες γραφικών

- Η μέθοδος `drawString(text, x, y)` εμφανίζει **κείμενο** τοποθετώντας την κάτω – αριστερά άκρη του στο σημείο (x, y). Δείτε στην ενότητα 3.7 Εφαρμογίδια και γραφικά για ένα παράδειγμα χρήσης.
  - Υπάρχει η κλάση `Font` που διαχειρίζεται (δημιουργεί, ανακαλεί / θέτει ιδιότητες) αντικείμενα γραμματοσειρών. Ο κατασκευαστής γραμματοσειρών δέχεται ως παραμέτρους το όνομα της γραμματοσειράς (πχ Dialog, SansSerif, Serif, Monospaced, DialogInput), το στιλ γραφής (`Font.PLAIN`, `Font.BOLD`, `Font.ITALIC`) και το μέγεθος των χαρακτήρων. Με τη μέθοδο `setFont` ορίζεται η ενεργή γραμματοσειρά.
  - Αντίστοιχα, με τη μέθοδο `setColor` τίθεται το ενεργό χρώμα γραφίδας. Πρώτα ωστόσο πρέπει να κατασκευάσουμε αντικείμενο της κλάσης `Color`. Ένα χρώμα στη Java είναι ένα αντικείμενο με τρεις ιδιότητες - τα βασικά χρώματα κόκκινο, πράσινο, μπλε. Κάθε χρώμα είναι ένας ακέραιος από 0 που σημαίνει καθόλου έως 255 που σημαίνει πλήρως. Για παράδειγμα,

```
Color xroma = new Color(50, 150, 250); //κατασκευάζει ένα αντικείμενο - χρώμα
g.setColor(xroma); //ό,τι γραφτεί από εδώ και κάτω θα έχει αυτό το χρώμα
g.drawString("Γεια", 50, 50);
```

    Επίσης, υπάρχουν «προκατασκευασμένα» χρώματα `Color.red`, `Color.green`, `Color.blue`.

- Οι μέθοδοι `drawLine`, `drawRect`, `drawOval` εμφανίζουν γραμμή, ορθογώνιο και έλλειψη αντίστοιχα πάνω στο αντικείμενο `Graphics` που εφαρμόζονται. Όλες ανήκουν στην κλάση `Graphics` του πακέτου `java.awt`.
  - Οι παραλλαγές τους `fillLine`, `fillRect`, `fillOval` σχεδιάζουν τα ίδια σχήματα «γεμισμένα» με το ισχύον χρώμα του αντικειμένου γραφικών.

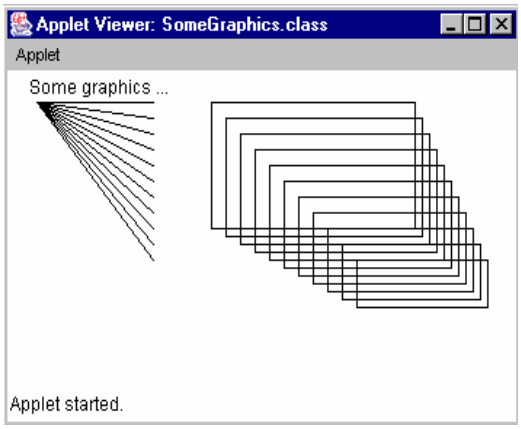
- Μια ακόμα χρήσιμη μέθοδος σχεδίασης είναι η `drawPolygon` (και `fillPolygon`) με επικεφαλίδα

```
public void drawPolygon(int[] xpoints, int[] ypoints, int npoints)
```

    που δέχεται ως παραμέτρους δύο πίνακες ακεραίων για τις συντεταγμένες των σημείων στις κορυφές του πολυγώνου

```
import java.awt.Graphics;    //για γραφικά
import javax.swing.JApplet; //για εφαρμογίδιο

public class SomeGraphics extends JApplet {
 public void paint(Graphics g) {
   //συντεταγμένες στη βάση της αρχής του κειμένου
   g.drawString("Some graphics ...", 15, 15);
   for (int i = 0; i <= 100; i += 10) {
     g.setColor(new Color(255-2*i, 2*i, i));
     //συντεταγμένες σημείων αρχής και τέλους
     g.drawLine (20, 20, 100, 20 + i);
     //συντετ. σημείου αρχής και πλάτος, ύψος
     g.fillRect (140+i, 20+i, 140-i/2, 80-i/2);
   } // end for
   } // end paint method
} // end class
```



Εικόνα 48 Πρόγραμμα που σχεδιάζει γραφικά σε εφαρμογίδιο και το αποτέλεσμα

    και τον αριθμό των πλευρών του.

- Στην Εικόνα 48 χρησιμοποιήσαμε την μέθοδο `paint` που μας δίνει αναφορά στο αντικείμενο γραφικών `g` για να σχεδιάσουμε πάνω του. Διαφορετικά πρέπει να χρησιμοποιήσουμε μια εντολή

```
Container cont = getContentPane();
```

  - Πάνω σε αυτό το αντικείμενο μπορούμε να τοποθετήσουμε ένα πλαίσιο χαρακτήρων, πχ με τον κώδικα

```
JTextArea textArea = new JTextArea (30, 40);  //γραμμές, στήλες
cont.add(textArea);
```

`public void init()`	Καλείται κατά την φόρτωση του εφαρμογίδιου στον περιηγητή. Εδώ αρχικοποιούνται οι μεταβλητές και περνιούνται οι παράμετροι από την ιστοσελίδα.
`public void start()`	Καλείται κάθε φορά που ο περιηγητής προβάλλει την ιστοσελίδα που ενσωματώνει το εφαρμογίδιο.
`public void paint(Graphics g)`	Καλείται κάθε φορά που χρειάζεται να "φρεσκαριστεί" η ιστοσελίδα, πχ από επικάλυψη άλλου παραθύρου. Το αντικείμενο `g` είναι η περιοχή γραφικών της επιφάνειας εργασίας του περιηγητή που ανήκει στο εφαρμογίδιο – οι διαστάσεις του ορίζονται στην ετικέτα `<applet>` της HTML.
`public void stop()`	Καλείται όταν ο περιηγητής εγκαταλείπει την ιστοσελίδα.
`public void destroy()`	Καλείται όταν το εφαρμογίδιο ξεφορτώνεται από τη μνήμη, τυπικά όταν τερματίζει ο περιηγητής.

Εικόνα 49 Μέθοδοι που καλούνται αυτόματα κατά την εκτέλεση εφαρμογίδιου

- Επειδή για την κατασκευή εφαρμογίδιων χρησιμοποιούμε την κλάση `JApplet`, ας δούμε τις μεθόδους που κληρονομούνται από αυτήν. Μπορούμε να τις επαναπροσδιορίσουμε για να υλοποιούν συγκεκριμένη συμπεριφορά στην κλάση μας. Στην Εικόνα 49 δείχνουμε την υπογραφή των κυριότερων μεθόδων και τη σειρά εκτέλεσής τους.

3.4 Εξαιρέσεις

- Ο μηχανισμός των εξαιρέσεων (exceptions) επιτρέπει σε ένα πρόγραμμα να σηματοδοτήσει την πραγματοποίηση ενός συμβάντος. Αυτό ονομάζεται **έγερση** της εξαίρεσης (raise the exception). Τότε διακόπτεται η κανονική ροή εκτέλεσης και ο έλεγχος περνάει στον αντίστοιχο **χειριστή** της εξαίρεσης (exception handler). Το συμβάν που αντιστοιχεί στην εξαίρεση μπορεί να είναι κάποιο σφάλμα ή κάποια ασυνήθιστη ή απροσδόκητη κατάσταση που απαιτεί άμεση αντιμετώπιση. Με το μηχανισμό των εξαιρέσεων: (α) δεν χρειάζεται να ελέγχουμε συνεχώς στο πρόγραμμα για όλα αυτά που μπορούν να συμβούν, και (β) μπορούμε να έχουμε συγκεντρωμένο σε ένα σημείο τον κώδικα που αντιμετωπίζει την εξαίρεση.
  - Παράδειγμα: ένα πρόγραμμα διαβάζει από την είσοδο αριθμούς και υπολογίζει το άθροισμά τους. Κατά την κανονική εκτέλεση του προγράμματος ο χρήστης πληκτρολογεί και όταν πατήσει Enter το πρόγραμμα διαβάζει τη συμβολοσειρά, την μετατρέπει σε αριθμό και την αθροίζει στο τρέχον άθροισμα. Κάποια στιγμή ο χρήστης πληκτρολογεί κάτι που δεν αντιστοιχεί σε νόμιμο αριθμό, πχ 1.8.3 ή a45. Κατά την μετατροπή της συμβολοσειράς σε αριθμό αυτό θα εγείρει μια συγκεκριμένη εξαίρεση. Το πρόγραμμα αντιλαμβάνεται την έγερση εξαίρεσης και, αφού προσδιορίσει τον τύπο της, μεταφέρει τον έλεγχο στον χειριστή της που τυπώνει μήνυμα «Λάθος στον αριθμό που πληκτρολογήσατε!» και τερματίζει το πρόγραμμα τυπώνοντας το τρέχον άθροισμα.
  - Κι άλλο παράδειγμα: Έχουμε δει ότι οι κατασκευαστές αντικειμένων δεν επιτρέπεται να επιστρέφουν κάποια τιμή. Άρα, αν ο κώδικας που καλεί έναν κατασκευαστή του περάσει μη αποδεκτές τιμές, ο κατασκευαστής δεν μπορεί να ειδοποιήσει γι αυτό το συμβάν, επιστρέφοντας πχ ένα συγκεκριμένο κωδικό. Στο παράδειγμα του τραπεζικού λογαριασμού, ας υποθέσουμε ότι έχουμε έναν κατασκευαστή που παίρνει ως παράμετρο μια αρχική κατάθεση και φτιάχνει ένα νέο τραπεζικό λογαριασμό με αυτό το υπόλοιπο. Αν κάποιος καλέσει `BankAccount myBA = new BankAccount(-17.0)`, ο μόνος τρόπος που ο κατασκευαστής μπορεί να ειδοποιήσει αυτόν που τον κάλεσε ότι δεν επιτρέπεται ένας νέος λογαριασμός να έχει αρνητική αρχική κατάθεση είναι να εγείρει μια εξαίρεση (δείτε αργότερα στην Εικόνα 53 για μια υλοποίηση).
- Παρακάτω θα δούμε πώς η Java μας επιτρέπει
  - να συλλαμβάνουμε και χειριζόμαστε εξαιρέσεις που μπορεί να εγερθούν ανά πάσα στιγμή επειδή καλέσαμε μεθόδους από τις βιβλιοθήκες της (εντολή `try`)
  - να εγείρουμε εξαιρέσεις μέσα από δικό μας κώδικα όταν θέλουμε να καταδείξουμε ότι συνέβη κάποια κατάσταση που απαιτεί άμεση αντιμετώπιση από αυτόν που την προκάλεσε (εντολή `throws`)
  - να ορίζουμε νέους τύπους εξαιρέσεων που έχουν τη δική τους συμπεριφορά.
- Στα προγράμματα που έχουμε δει μέχρι τώρα (εκτός από το Είσοδος / Έξοδος στην τυπική ροή, σελ. 15) δεν υπάρχει χειρισμός εξαιρέσεων και έτσι οποιαδήποτε εξαίρεση κι αν εγερθεί προκαλεί τερματισμό του προγράμματος και τυπώνει μήνυμα λάθους, πχ

```
java.lang.ArithmeticException: / by zero
   at SomeClass.main(SomeClass.java:5)
```

  που σημαίνει ότι στην 5η γραμμή του αρχείου `SomeClass.java` και ενώ εκτελούνταν η μέθοδος `main` εγέρθηκε η εξαίρεση `ArithmeticException` εξαιτίας διαίρεσης δια 0.

Εξαίρεση	Πακέτο	Εξήγηση
ArithmeticException	java.lang	Επιχειρήθηκε παράνομη αριθμητική πράξη, όπως διαίρεση δια μηδέν.
ArrayStoreException	java.lang	Επιχειρήθηκε να ανατεθεί ακατάλληλη τιμή σε ένα στοιχείο πίνακα.
IOException	java.io	Λειτουργία εισόδου / εξόδου δεν μπόρεσε να ολοκληρωθεί επιτυχώς.
IllegalArgumentException	java.lang	Μέθοδος κλήθηκε με ακατάλληλη παράμετρο.
IndexOutOfBoundsException	java.lang	Υποδείκτης ενός πίνακα, συμβολοσειράς ή διανύσματος είναι εκτός ορίων.
NoSuchMethodException	java.lang	Η συγκεκριμένη μέθοδος δεν βρέθηκε.
NumberFormatException	java.lang	Επιχειρήθηκε λειτουργία με αριθμό σε παράνομη μορφή.
ParseException	java.text	Μια συμβολοσειρά δεν μπόρεσε να αναγνωριστεί σύμφωνα με την προδιαγεγραμμένη μορφή.
RuntimeException	java.lang	Η υπέρ-κλάση όλων των μη ελεγμένων εξαιρέσεων

Εικόνα 50 Μερικοί από τους τύπους εξαιρέσεων που ορίζει η Java

- Η Java έχει ένα προκαθορισμένο σύνολο τύπων εξαιρέσεων που μπορούν να συμβούν κατά την εκτέλεση ενός προγράμματος. Στην Εικόνα 50 παρουσιάζονται αυτές που συναντάμε πιο συχνά.

```
try {
 εντολές-που-μπορεί-να-προκαλέσουν-εξαίρεση
}
catch (ΤύποςΕξαίρεσης1 var1) {
  εντολές-χειρισμού-εξαίρεσης1
}
catch (ΤύποςΕξαίρεσης2 var2) {
  εντολές-χειρισμού-εξαίρεσης2
}
//εδώ μπορούν να υπάρχουν και άλλες catch
finally { //προαιρετικό
  εντολές-που-εκτελούνται-πάντα
  //είτε εγερθεί κάποια εξαίρεση είτε όχι
}
```

Εικόνα 51 Συντακτικό εντολής για χειρισμό εξαιρέσεων

- **Χειρισμός εξαιρέσεων.** Για να ελέγχουμε για τυχόν έγερση εξαιρέσεων σε κάποιο κομμάτι κώδικα, το εγκλείουμε σε μια εντολή **try**. Αμέσως μετά τοποθετούμε ενότητες **catch** που χειρίζονται συγκεκριμένες εξαιρέσεις. Κατά την εκτέλεση, αν οι εντολές στο εσωτερικό της **try** δεν προκαλέσουν κάποια εξαίρεση, το πρόγραμμα συνεχίζει μετά τις **catch**. Αν εγερθεί μια εξαίρεση, η ροή εκτέλεσης διακόπτεται και ο έλεγχος περνάει στις ενότητες **catch**. Εκεί αναγνωρίζεται ποια εξαίρεση έχει συμβεί και εκτελείται ο αντίστοιχος κώδικας χειρισμού της. Αν συμβεί μια εξαίρεση για την οποία δεν υπάρχει αντίστοιχη **catch**, η ροή ελέγχου μεταβιβάζεται στην μέθοδο που κάλεσε την τρέχουσα μέθοδο, κοκ. Αυτό ονομάζεται **διάδοση εξαίρεσης** (exception propagation) και συνεχίζεται προς τα έξω μέχρι να βρεθεί κατάλληλος χειριστής ή να φτάσουμε στο επίπεδο της `main`, οπότε το πρόγραμμα τερματίζεται.

```
import javax.swing.*;
public class ExceptionHandlingExample {
  public static void main(String args[]) throws Exception{
    String str; int iEtos;
    try {
      str = JOptionPane.showInputDialog(null, "Δώσε έτος:");
      iEtos = Integer.parseInt(str);
    } //end try
    catch (Exception e) {
      System.out.print("αγνόησε την εξαίρεση");
      iEtos = 2003;
    } //end catch
    System.out.println("Πέρσι ήταν " + (iEtos - 1));
  } //end main
} //end class
```

Εικόνα 52 Χειρισμός εξαιρέσεων

  - Ένα σημείο που στα προγράμματά μας χρειάζεται να χειριστούμε εξαιρέσεις που εγείρει το σύστημα είναι όταν χρησιμοποιούμε τις μεθόδους `Integer.parseInt`, `Float.parseFloat`, κλπ, για να μετατρέψουμε μια συμβολοσειρά που εισήχθηκε από το χρήστη σε κάποιο τύπο δεδομένων. Αν η είσοδος δεν αναγνωρίζεται ως νόμιμη, οι `parseΚάτι` εγείρουν μια εξαίρεση και το πρόγραμμα τερματίζει. Για να μην διακοπεί η εκτέλεση του προγράμματος, πρέπει να έχουμε χειριστή της εξαίρεσης (δείτε Εικόνα 52).
  - Η ενότητα **finally** της εντολής **try**, αν υπάρχει, τοποθετείται μετά από όλες τις **catch**. Αν δεν εγερθεί κάποια

εξαίρεση, οι εντολές μέσα στη **finally** εκτελούνται μετά τις εντολές του σώματος της **try**. Αν εγερθεί κάποια εξαίρεση, η εκτέλεση της εντολής μέσα στην **try** διακόπτεται, ερευνάται αν υπάρχει κατάλληλη **catch** για να χειριστεί την εξαίρεση και, αν υπάρχει, εκτελούνται οι εντολές χειρισμού της εξαίρεσης. Κατόπιν εκτελούνται και οι εντολές της **finally**.

```
public class BankAccount {
  private double balance;

  public BankAccount() { balance = 0.0d; }

  public BankAccount(double initialAmount) {
    if (initialAmount >= 0.0d) balance = initialAmount;
    else throw new IllegalArgumentException ("Opening amount cannot be negative.");
  }

  public void deposit(double amount) {
    if (amount > 0.0d) balance += amount; //καταθεση
    else throw new IllegalArgumentException ("Amount must be positive."); }

  public void withdraw(double amount) {
    if (amount>0.0d)
      if (amount <= balance) balance -= amount; //αναληψη
      else throw new IllegalArgumentException ("Balance not enough.");
    else throw new IllegalArgumentException ("Amount must be positive."); }

  public double getBalance() { return balance; }
}
```

```
public class BankAccountDemo {
  public static void main(String[] args) {
    BankAccount myAccount = new BankAccount();
    boolean ok;
    try {
      myAccount.deposit(100.0); myAccount.withdraw(50.0);
      myAccount.withdraw(70.0); myAccount.deposit(-20.0);
    }
    catch (Throwable e) {
      System.out.println(e);
    }
    System.out.println(myAccount.getBalance());
  }
}
```

Εικόνα 53 Έλεγχος παραμέτρων με εξαιρέσεις

- **Έγερση εξαιρέσεων.** Όταν γράφουμε μεθόδους, οι παράμετροί τους πρέπει να υπακούουν σε συγκεκριμένους περιορισμούς, οι οποίοι περιγράφονται στην τεκμηρίωση της μεθόδου. Ο μηχανισμός των εξαιρέσεων μπορεί να χρησιμοποιηθεί για να ελέγχει την τήρηση των περιορισμών, όπως φαίνεται στην Εικόνα 53. Στην κλάση `BankAccount` εγείρονται εξαιρέσεις όταν δεν μπορούν να εκτελεστούν οι δοσοληψίες `deposit` και `withdraw`. Στην κλάση `BankAccountDemo` η **try** ελέγχει το επίμαχο κομμάτι του κώδικα για να χειριστεί τις εξαιρέσεις που τυχόν θα εγερθούν.
  - Για να εγείρουμε την εξαίρεση, χρησιμοποιούμε την εντολή **throw** `εξαίρεση`. Κατά τα γνωστά, η εξαίρεσή μας θα διακόψει την εκτέλεση του προγράμματος, εκτός κι αν έχουμε φτιάξει χειριστή για τη συγκεκριμένη εξαίρεση.
  - Αν σε μια μέθοδο υπάρχει κώδικας (όπως μια `readLine`) που μπορεί να προκαλέσει την έγερση μιας εξαίρεσης και δεν υπάρχει χειριστής γι αυτή την εξαίρεση, στην επικεφαλίδα της μεθόδου πρέπει να δηλωθεί ότι μέσα από τη συγκεκριμένη μέθοδο μπορεί να εγερθεί μια εξαίρεση. Αυτή η δήλωση γίνεται στο τέλος της επικεφαλίδας της μεθόδου με την ενότητα **throws** `ΤύποςΕξαίρεσης`. Δείτε το παράδειγμα στην ενότητα 1.14 Είσοδος / Έξοδος στην τυπική ροή.
  - Για να επιτραπεί σε μια μέθοδο να εγείρει έναν συγκεκριμένο τύπο εξαίρεσης πρέπει να το δηλώσουμε στην κεφαλίδα της με την εντολή **throws**.
  - Οι εντολές **throw** και **throws** δεν πρέπει να συγχέονται. Η **throws** δηλώνει ότι μια μέθοδος *έχει τη δυνατότητα* να εγείρει μια εξαίρεση, ενώ η **throw** εγείρει την εξαίρεση σε ένα συγκεκριμένο σημείο του κώδικα της μεθόδου.

- **Ορισμός νέων τύπων εξαιρέσεων.** Εκτός από τους τύπους εξαιρέσεων που μας παρέχει η Java, μπορούμε να ορίσουμε νέους τύπους εξαιρέσεων. Ένας νέος τύπος εξαίρεσης πρέπει να επεκτείνει την κλάση `Exception` ή κάποια υποκλάση της[4]. Συνήθως παρέχουμε έναν κατασκευαστή που τυπώνει ένα μήνυμα, όπως φαίνεται στην Εικόνα 54.

```
public class MyException extends Exception {
//ορίζεται ο τύπος της νέας εξαίρεσης
  public MyException() { //κατασκευαστής
    super("ΟΥΠΣ! Συνέβη μια NewException!");
  }
} //end class MyException
```

```
public class MyProgram { //κάποιο πρόγραμμα
  public void someMethod() throws MyException {
  //δηλώνεται ότι μπορεί να εγείρει τη συγκεκριμένη εξαίρεση
    ...
    if (somethingHappened) throw new MyException(); //έγερση
    ...
  }
} //end class MyProgram
```

Εικόνα 54 Ορισμός και έγερση μιας νέας εξαίρεσης

- Η χρήση του μηχανισμού των εξαιρέσεων είναι απαραίτητη για την σύλληψη και τον χειρισμό των εξαιρέσεων που εγείρει η ίδια η Java. Αλλά είναι και επιθυμητή γιατί διαχωρίζει τον κώδικα που αντιμετωπίζει τις ακραίες ή σπάνιες περιπτώσεις από τον κύριο κώδικα του προγράμματος τον οποίο ο προγραμματιστής μπορεί να βελτιστοποιήσει. Επίσης βελτιώνει την αξιοπιστία του προγράμματος, αφού βοηθάει στην αποφυγή των ανώμαλων τερματισμών (program crashes).

3.5 Ισχυρισμοί (assertions)

- Για να πειστούμε για την ορθότητα του προγράμματος μπορούμε να υποβάλλουμε το πρόγραμμα σε δοκιμές. Σπάνια μπορούμε να ελέγξουμε την ορθότητα των εξόδων του προγράμματος για όλες τις πιθανές εισόδους, οπότε επιλέγουμε ένα σύνολο αντιπροσωπευτικών εισόδων και ελέγχουμε τις εξόδους τους. Έτσι όμως *πιθανολογούμε* για την ορθότητα. Εάν επιθυμούμε να ελέγξουμε το πρόγραμμα πλήρως και δεν είναι δυνατόν να εξαντλήσουμε το σύνολο των πιθανών εισόδων του, μπορούμε να χρησιμοποιήσουμε ισχυρισμούς.
  - Σε επιλεγμένα σημεία του προγράμματος τοποθετούμε προτάσεις που, από τις προδιαγραφές, επιβάλλεται να ισχύουν, όποτε η ροή εκτέλεσης φθάνει εκεί. Ο ισχυρισμός που τίθεται στην αρχή του προγράμματος απεικονίζει τις απαιτήσεις ορθότητας των δεδομένων εισόδου. Τότε πρέπει να αποδείξουμε, ξεκινώντας από τον αρχικό ισχυρισμό, ότι κατά την εκτέλεση του προγράμματος ισχύουν όλοι οι ισχυρισμοί που θέσαμε. Ο τελικός ισχυρισμός θα ισχύει για την έξοδο του προγράμματος και θα πιστοποιεί ότι η έξοδος πληροί τις προδιαγραφές που θέσαμε κατά την ανάπτυξη του προγράμματος.

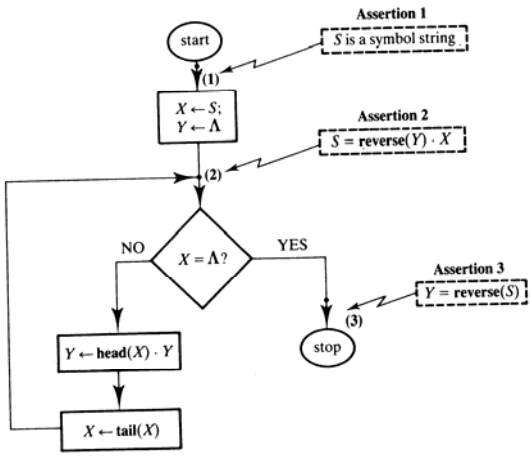
- Η Java υλοποιεί το χαρακτηριστικό των ισχυρισμών από την έκδοση 1.4 του JDK με την εντολή: **`assert`** δυαδικήΠαράσταση`;`
  - Κατά την εκτέλεση του προγράμματος, εκτιμάται η παράσταση και, αν είναι αληθής, το πρόγραμμα συνεχίζει την εκτέλεσή του κανονικά. Αν είναι ψευδής, εγείρεται μια εξαίρεση τύπου `AssertionError`.
  - Οι ισχυρισμοί στη Java λειτουργούν ως πρόσθετος μηχανισμός αποσφαλμάτωσης που ελέγχει αν συγκεκριμένες συνθήκες ισχύουν σε συγκεκριμένα σημεία του προγράμματος.
  - Για λόγους ταχύτητας ο έλεγχος των ισχυρισμών είναι εξ ορισμού ***απ***ενεργοποιημένος κατά τη διάρκεια της εκτέλεσης των προγραμμάτων. Η πρακτική που ακολουθείται συνήθως είναι να διατηρούμε το μηχανισμό ελέγχου των ισχυρισμών ενεργό κατά τη διάρκεια της ανάπτυξης του προγράμματος τουλάχιστον. Πάντως, ασφαλέστερο είναι να θυσιάζεται ένα μικρό ποσοστό επιδόσεων προκειμένου να διατηρείται πάντοτε ενεργός ο μηχανισμός ελέγχου ισχυρισμών, ακόμη και όταν το πρόγραμμα έχει περάσει τις διαδικασίες ελέγχου και πιστοποίησης και εκτελείται σε περιβάλλον παραγωγής.
  - Η υλοποίηση των ισχυρισμών στη Java παρουσιάζει κενά και μειονεκτήματα σε σχέση με άλλες γλώσσες προγραμματισμού, όπως η Eiffel. Μόνον μερικώς υποστηρίζει το στιλ προγραμματισμού **σχεδίαση-κατά-συμβόλαιο** (design-by-contract). Κατά τη μεθοδολογία αυτή οι προδιαγραφές του λογισμικού τίθενται εκ των προτέρων και δεσμευτικά (σαν να ήταν συμβόλαιο) και η σχεδίαση γίνεται με τέτοιο τρόπο ώστε τα αρθρώματα που απαρτίζουν το πρόγραμμα υποχρεωτικά να πληρούν τις προδιαγραφές μέσω ισχυρισμών επί των δεδομένων εισόδου και εξόδου. Οι ισχυρισμοί που ορίζονται αντικατοπτρίζουν πιστά τις προδιαγραφές.

- Ορισμοί μαθηματικής ορθότητας προγράμματος
  - **Μερική ορθότητα** προγράμματος: εάν φθάσει στο τέλος, τότε το αποτέλεσμα θα είναι σωστό.
  - **Ολική ορθότητα**: πράγματι, θα φθάσει στο τέλος σε πεπερασμένο χρόνο, και το αποτέλεσμα θα είναι σωστό.

- Η **εξακρίβωση βρόχων** (loop verification) είναι μια μέθοδος για να αποδείξουμε την ορθότητα σε επαναληπτικές δομές.

[4] Στην ιεραρχία κλάσεων της Java κάτω από την κλάση Exception υπάρχουν πιο ειδικές υποκλάσεις: `RuntimeException`, `ClassNotFoundException`, κά. Αρκετοί από τους τύπους εξαιρέσεων στην Εικόνα 50 είναι υποκλάσεις της `RuntimeException`.

- Ας πάρουμε για παράδειγμα ένα πρόγραμμα που αντιστρέφει μια συμβολοσειρά. Η λειτουργία του προγράμματος απεικονίζεται στο διάγραμμα ροής στην Εικόνα 55 που παίρνει σε κάθε επανάληψη βρόχου ένα χαρακτήρα από τη συμβολοσειρά εισόδου και τον συσσωρεύει σε μια μεταβλητή εξόδου. Εκτελέστε το πρόγραμμα για είσοδο S="abc" και βεβαιωθείτε ότι μετά τρεις επαναλήψεις Y="cba". Όταν αποδείξουμε τους ισχυρισμούς και στις 3 μεταβάσεις (1→2, 2→3, 2→2) είμαστε βέβαιοι για τη μερική ορθότητα του προγράμματος.
- Ο πρώτος ισχυρισμός προκύπτει από τα δεδομένα εισόδου – γνωρίζουμε ότι το S είναι μια συμβολοσειρά, ακόμη κι αν είναι η κενή συμβολοσειρά (συμβολίζεται με Λ). Ο ισχυρισμός 2 αποδεικνύεται με τη μέθοδο της μαθηματικής αναδρομής: στην πρώτη επανάληψη είναι S=reverse(Y).X=reverse(Λ).X=Λ.X=X, αφού το αντίστροφο της κενής συμβολοσειράς είναι μια κενή συμβολοσειρά. Υποθέτουμε ότι σε κάποια επανάληψη του βρόχου ισχύει S=reverse(Y_0).X_0. Κατόπιν εκτελούνται οι δύο εντολές του βρόχου Y_1=head(X_0).Y_0 και X_1=tail(X_0). Πρέπει να ελέγξουμε ότι ισχύει ότι S

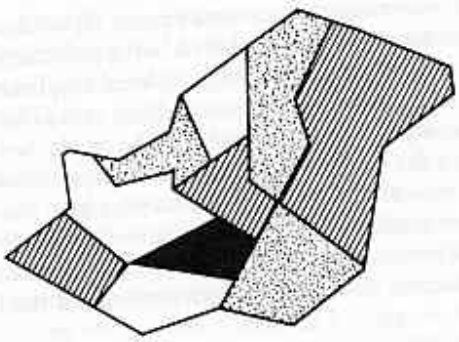


Εικόνα 55 Αλγόριθμος με ισχυρισμούς

S = reverse(Y_1)	.X_1	με αντικατάσταση προκύπτει
= reverse(head(X_0).Y_0)	.tail(X_0)	από ιδιότητα της reverse ισχύει
= (reverse(Y_0).reverse(head(X_0)).tail(X_0)		ο αντίστροφος ενός χαρακτήρα είναι ο χαρακτήρας
= reverse(Y_0) .head(X_0)	.tail(X_0)	συνένωση κεφαλής και ουράς κάνουν τη συμβολοσειρά
= reverse(Y_0) .X_0=S		που ισχύει.

- Τέλος χρησιμοποιώντας μαθηματική αναδρομή αποδεικνύουμε ότι ο αλγόριθμος ολοκληρώνεται μετά από τόσα βήματα, όσα και οι χαρακτήρες της συμβολοσειράς εισόδου. Άρα το πρόγραμμα τερματίζει σε πεπερασμένο αριθμό βημάτων, συνεπώς δείξαμε την ολική ορθότητά του.

- Το πρόβλημα του χρωματισμού χάρτη: Έχουμε ένα χάρτη που απεικονίζει τα σύνορα χωρών. Θέλουμε να χρωματίσουμε τις χώρες, έτσι ώστε όμορες χώρες να μην έχουν το ίδιο χρώμα. Ποιος είναι ο μικρότερος αριθμός χρωμάτων που πρέπει να χρησιμοποιήσουμε; Το θεώρημα λέει ότι αρκούν 4 χρώματα για να μην συνορεύουν χώρες όμοιου χρώματος. Το πρόβλημα τέθηκε το 1852, η πρώτη απόδειξη έγινε το 1976 με υπολογιστή που εξέτασε 1700 αντιπροσωπευτικές περιπτώσεις χαρτών.



Εικόνα 56 Χρωματισμένος χάρτης

3.6 Ασκήσεις

Α42. Το παρκάρισμα σε ένα χώρο στάθμευσης κοστίζει €2.00 για τις τρεις πρώτες ώρες. Από εκεί και πέρα κοστίζει €0.50 για κάθε ώρα ή τμήμα της. Η μέγιστη ημερήσια χρέωση είναι €10.00. Υποθέστε ότι κανένα αυτοκίνητο δεν μένει για περισσότερο από 24 ώρες. Φτιάξτε εφαρμογίδιο που θα υπολογίζει το κόστος με βάση τον χρόνο στάθμευσης. Δείτε την ενότητα 3.8 Ώρα και πάρκινγκ για μια υλοποίηση.

Α43. Φτιάξτε πρόγραμμα που να εμφανίζει τη γραφική παράσταση μιας συνάρτησης y=f(x) για συγκεκριμένο εύρος τιμών x. Θα δίνεται ως είσοδος το x_{min} και x_{max}, καθώς και το πλήθος των γραμμών που θα σχεδιαστούν. Πάρτε δειγματοληπτικές τιμές της συνάρτησης για να φτιάξετε την κλίμακα του άξονα y των τεταγμένων.

Α44. Ένας αριθμός ονομάζεται τέλειος εάν το άθροισμα των παραγόντων του, εκτός από τον ίδιο, κάνει τον αριθμό αυτόν. Για παράδειγμα, ο 6 είναι τέλειος, επειδή 1+2+3=6. Φτιάξτε μέθοδο που να ελέγχει αν ένας αριθμός είναι τέλειος και έπειτα χρησιμοποιήστε την για να βρείτε και να τυπώσετε όλους τους τέλειους αριθμούς έως το 1000 πάνω σε μια `JTextArea`. Για καθέναν από αυτούς θα τυπώνονται και οι παράγοντές του.

Α45. Προσθέστε στην άσκηση Α41 (σελίδα 31) εξαιρέσεις για τις περιπτώσεις που προσπαθεί κάποιος να κατασκευάσει «παράνομα» νομίσματα ή να προσθέσει ανόμοια νομίσματα, πχ EUR με USD. Για εξάσκηση κατασκευάστε μεθόδους που συγκρίνουν δύο νομίσματα για να βρουν αν είναι ίσης αξίας, ποιο είναι το μεγαλύτερο, και ποιο το μικρότερο σε αξία.

3.7 Εφαρμογίδια και γραφικά

```
<html>
<head>   <title> Ιστοσελίδα με εφαρμογίδιο </title>   </head>
<body>
<h2> Γεια ! </h2>
<p> Ακολουθεί το παράθυρο του εφαρμογίδιου</p>
<applet code = "SomeGraphics.class" name = "TestApplet"
  width = "500" height = "500" hspace = "0" vspace = "0">
<param name="grammes" value="11">
</applet>
<p align="right"> Καλή συνέχεια! </p>
</body>
```

```
</html>
```

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

public class SomeGraphics extends JApplet {

  final static int YPSOS = 500;  //διαστάσεις παραθύρου
  final static int PLATOS = 500;

  int iGrammes; //μεταβλητή για την παράμετρο που έρχεται από την ιστοσελίδα

  public void init() {
    iGrammes = Integer.parseInt(getParameter("grammes"));
  }

  public void paint(Graphics g) {
    Font f = getFont(); //πάρε την ενεργή γραμματοσειρά
    f = new Font (f.getFontName(), f.BOLD, 20); // με αυτή σαν βάση άλλαξέ της στιλ και μέγεθος
    g.setFont(f);
    g.drawString("Σχέδια", 100, 100); //γράψε με την ενεργή γραμματοσειρά και χρώμα στο (100,100)

    int iApostasi = YPSOS / iGrammes;
    for (int i = 1; i <= iGrammes; i++) {
      g.setColor(new Color(randomColor(),randomColor(),randomColor()));
      g.drawLine(0, 250, 500, i * iApostasi);
    }

    for (int i = 1; i <= 100; i++) {
      g.setColor(new Color(randomColor(),randomColor(),randomColor()));
      g.fillOval(randomPoint(),randomPoint(),30,10);
    }
  }

  private static int randomColor(){ return (int) (256 * Math.random()); }
  private static int randomPoint(){ return (int) (YPSOS * Math.random()); }
```

3.8 Ώρα και πάρκινγκ

```
package parking;
public class Ora { //διαχειρίζεται αντικείμενα τύπου 'ώρα'

  // μεταβλητές στιγμιοτύπου ************************************************
  private byte ora;     // ώρες           0..23
  private byte lepta;   // λεπτά της ώρας 0..59
  private byte deftera; // δευτερόλεπτα   0..59

  // κατασκευαστές **********************************************************
  public Ora() {
    // δημιουργεί ένα αντικείμενο που δείχνει μεσάνυκτα, 00:00:00
  } // end Ora1

  public Ora(int ora, int lepta, int deftera) {
    /* Ελέγχει αν η ώρα, τα λεπτά και τα δεύτερα είναι νόμιμα
     * Αν είναι εκτός ορίων, τα θέτει στο 0. */

    this.ora     = (byte) (((ora>=0)     && (ora<=23))     ?ora    :0);
    this.lepta   = (byte) (((lepta>=0)   && (lepta<=59))   ?lepta  :0);
    this.deftera = (byte) (((deftera>=0) && (deftera<=59))?deftera:0);
  } // end Ora2

  public Ora(String plirisOra) {
  /* η παράμετρος πρέπεινα είναι της μορφής hh:mm:ss
   * κομματιάζει τη συμβολοσειρά και καλεί τον προηγούμενο κατασκευαστή */
    this(Integer.parseInt(plirisOra.substring(0, 2)),
         Integer.parseInt(plirisOra.substring(3, 5)),
         Integer.parseInt(plirisOra.substring(6, 8)));
  } // end Ora3

  // μέθοδοι ανάγνωσης και εγγραφής *****************************************
  public int getPart(char part) {
    // ανάλογα με την τιμή της παραμέτρου h, m, s, επιστρέφει το τμήμα της ώρας
    byte byTemp;
```

```
    switch (part) {
      case 'h':
        byTemp = ora;
        break;
      case 'm':
        byTemp = lepta;
        break;
      case 's':
        byTemp = deftera;
        break;
      default:
        byTemp = 0;
    } // end switch
    return (int) byTemp;
  } // getPart

  public void setPart(char part, int value) {
  /* ανάλογα με την τιμή της παραμέτρου part (h, m, s), θέτει ώρα, λεπτά ή
   * δευτερόλεπτα στην τιμή value, αφού πρώτα ελέγξει τη νομιμότητά της */
    switch (part) {
      case 'h':
        ora = (byte) (((value>=0) && (value<=23))?value:0);
        break;
      case 'm':
        lepta = (byte) (((value>=0) && (value<=59))?value:0);
        break;
      case 's':
        deftera = (byte) (((value>=0) && (value<=59))?value:0);
        break;
      default:
        // μην κάνεις τίποτα
    } // end switch
  } // setPart

  // άλλες μέθοδοι ************************************************************
  public String showOra() {
    return ora + ":" + lepta + ":" + deftera;  } // showOra

  public String showOraB() {
    return (ora>12?ora-12:ora) + ":" + lepta + ":" + deftera +
      (ora>12?"μμ":"πμ");
  } // showOraB

  public double convertOra() {
    // μετατρέπει την ώρα σε πραγματικό αριθμό
    return ora + lepta / 60.0 + deftera / 3600.0;
  } //  convertOra

} //end class
```

```
package parking;
import java.awt.Graphics;
import javax.swing.*;

public class Parking extends JApplet {

  // σταθερές κλάσης **********************************************************
  final static double MINIMUM_TIMH  = 2.0;
  final static double MINIMUM_ORA   = 3.0;
  final static double TIMH_ANA_ORA  = 0.5;
  final static double MAXIMUM_TIMH  = 6.0;

  // μεταβλητές στιγμιοτύπου **************************************************
  Ora eisodos;
  Ora exodos;

  // εισαγωγή δεδομένων στο πρόγραμμα *****************************************
  public void init() {
    String sEisodos = JOptionPane.showInputDialog("Ωρα εισόδου (ωω:λλ:δδ): ");
    String sExodos  = JOptionPane.showInputDialog("Ωρα εξόδου  (ωω:λλ:δδ): ");
    eisodos = new Ora(sEisodos);
    exodos  = new Ora(sExodos);
  } //end init
```

```
  // εμφάνιση αποτελεσμάτων ************************************************
  public void paint(Graphics g) {
    g.drawString("Είσοδος αυτοκινήτου:     " + eisodos.showOraB(), 15, 20);
    g.drawString("Εξοδος αυτοκινήτου:      " + exodos.showOraB(),  15, 40) ;
    double diarkeia = computeDiafora(eisodos, exodos);
    g.drawString("Διάρκεια σε ώρες:        " + diarkeia,           15, 60) ;
    double timh = computeTimh(diarkeia);
    g.drawString("Χρέωση σε ευρώ:          " + timh,               15, 80) ;
  } //end paint

  // υπολογισμοί ***********************************************************
  private static double computeDiafora(Ora ora1, Ora ora2) {
    // υπολογίζει την διαφορά δύο ωρών
    return ora2.convertOra() - ora1.convertOra();
  } // computeDiafora

  private static double computeTimh(double diarkeia) {
    // υπολογίζει την τιμή που χρεώνει το πάρκινγκ
    double dTimh;
    dTimh = MINIMUM_TIMH; //με την είσοδο στο πάρκινγκ, χρεώνεσαι το μίνιμουμ
    if (diarkeia > MINIMUM_ORA) {
      // μείωσε τη διάρκεια γιατί οι πρώτες ώρες έχουν χρεωθεί
      diarkeia -= MINIMUM_ORA;
      // αύξησε το κόστος για τις υπόλοιπες ώρες
      dTimh += TIMH_ANA_ORA * Math.ceil(diarkeia); //υπολόγισε κάθε τμήμα ώρας ως ολόκληρη
    } // end if
    if (dTimh > MAXIMUM_TIMH) dTimh = MAXIMUM_TIMH;
    return dTimh;
  } // computeTimh
} //end class
```