JAVA TUTORIAL

Πανεπιστήμιο Μακεδονίας
Τμήμα Εφαρμοσμένης Πληροφορικής
Εργαστήριο Παράλληλης Κατανεμημένης Επεξεργασίας

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΕΦΑΛΑΙΟ 1. Τι κάνει τη Java να ξεχωρίζει;

Η Java προκάλεσε ίσως το μεγαλύτερο ενδιαφέρον σε σύγκριση με οποιαδήποτε άλλη εξέλιξη στον κόσμο του Internet. Όλοι μιλούν γι' αυτήν. Όλοι έχουν ενθουσιαστεί με τη

Java για τις δυνατότητες που προσφέρει. Είναι η πρώτη που κατάφερε να συμπεριλάβει ήχο και κίνηση σε μια ιστοσελίδα. Η Java επιπλέον επιτρέπει στους χρήστες να αλληλεπιδρούν (interact) με την ιστοσελίδα. Εκτός από το να διαβάζει απλά και ίσως να συμπληρώνει μία φόρμα, ο χρήστης μπορεί τώρα να παίξει παιχνίδια, να συνομιλήσει, να λαμβάνει συνεχώς τις πιο πρόσφατες πληροφορίες και πολλά άλλα.

Ακολουθούν μερικές από τις πολλές δυνατότητες της Java:

- Ήχος ο οποίος εκτελείται όποτε ο χρήστης φορτώνει μία σελίδα
- Μουσική που παίζει στο background μιας σελίδας
- Δημιουργία κινουμένων σχεδίων
- Βίντεο
- Παιχνίδια με πολυμέσα

Η Java δεν είναι απλά μια γλώσσα προγραμματισμού του δικτύου με ειδικά χαρακτηριστικά. Παρόλο που η HotJava ήταν η πρώτη γλώσσα που συμπεριέλαβε ήχο και κίνηση, ο Microsoft Internet Explorer 2.0 και ο Netscape Navigator 2.0 υποστηρίζουν αυτά τα χαρακτηριστικά με πολλούς και διαφορετικούς τρόπους. Τι κάνει τη Java να ξεχωρίζει; Η Java είναι μια γλώσσα προγραμματισμού για ποικίλες εφαρμογές. Δεν προσφέρει απλά τη δυνατότητα να προσθέσει ο χρήστης νέο περιεχόμενο στις σελίδες του (όπως συμβαίνει στο Netscape και στον Internet Explorer) αλλά επιτρέπει να προσθέσουμε και τον κώδικα που είναι απαραίτητος. Δεν χρειάζεται πλέον να περιμένετε για να κυκλοφορήσει ο browser που θα υποστηρίξει τον συγκεκριμένο τύπο εικόνας ή το ειδικό πρωτόκολλο παιχνιδιού (special game protocol). Με τη Java εσείς στέλνετε στους browsers το περιεχόμενο που χρειάζεται και το πρόγραμμα για να δείτε αυτό το περιεχόμενο την ίδια στιγμή.

Ας δούμε τι σημαίνει αυτό. Μέχρι τώρα έπρεπε να περιμένετε τους αναγνώστες σας να ενημερώσουν τους browsers τους προτού χρησιμοποιήσετε ένα νέο τύπο περιεχομένου (content type). Η ανταγωνιστικότητα της Java βρίσκεται στο ότι μπορεί να εφαρμοστεί σε οποιονδήποτε browser.

Για παράδειγμα, θέλετε να χρησιμοποιήσετε τα αρχεία EPS στο site σας. Προηγουμένως, έπρεπε να περιμένετε μέχρι ένας τουλάχιστον web browser να εφάρμοζε την υποστήριξη EPS. Τώρα πια δεν περιμένετε. Αντίθετα, μπορείτε να γράψετε τον δικό σας κώδικα για να δείτε τα αρχεία EPS και να το στείλετε σε οποιονδήποτε πελάτη ζητά τη σελίδα σας τον ίδιο χρόνο που ζητά το αρχείο EPS.

Υποθέστε ότι θέλετε άτομα που να μπορούν να ψάχνουν τον ηλεκτρονικό σας κατάλογο (electronic card catalog). Η βάση δεδομένων του καταλόγου όμως υπάρχει σ' ένα μεγάλο σύστημα που δεν αναγνωρίζει την HTTP. Πριν τη Java θα μπορούσατε να ελπίζετε ότι κάποιος browser θα εφάρμοζε το πρωτόκολλο της κάρτας ή θα μπορούσατε να προσπαθήσετε να προγραμματίσετε κάποιο ενδιάμεσο cgi-bin σε ένα UNIX BOX που θα αναγνώριζε HTTP, πράγμα που δεν είναι καθόλου εύκολο. Με τη Java, όταν ένας πελάτης θέλει να μιλήσει στον κατάλογό σας μπορείτε να του στείλετε τον κώδικα που χρειάζεται.

Η Java δεν είναι γλώσσα μόνο για τα web sites. Η Java είναι μια γλώσσα προγραμματισμού που μας επιτρέπει να κάνουμε ό,τι και οι παραδοσιακές γλώσσες, όπως η Fortran και η C++. Είναι σαφώς πιο καθαρή και πιο εύκολη όμως στη χρήση από αυτές. Σαν γλώσσα η Java είναι:

- Απλή (Simple)
- Αντικειμενοστραφής, δηλαδή τα πάντα στη Java είναι είτε κλάση, είτε μέθοδος ή αντικείμενο
- Ανεξάρτητη από το σύστημα, δηλαδή τα προγράμματα σε Java μπορούν να διαβαστούν και να τρέξουν από μεταγλωττιστές σε διάφορες πλατφόρμες όπως Windows 95, Windows NT και Solaris 2.3
- Ασφαλής
- Πολυνηματική, δηλαδή ένα απλό πρόγραμμα σε Java μπορεί να κάνει πολλά, διαφορετικά προγράμματα ανεξάρτητα και αλληλεπιδρώντα.

ΕΓΚΑΘΙΣΤΩΝΤΑΣ ΤΗ JAVA

Εκδόσεις της Java σε διαφορετικά στάδια ολοκλήρωσης διατίθενται από τη Sun for Windows 95 και Windows NT for X86, Unix και MacOS 7.5. Μέχρι στιγμής δεν υπάρχουν εκδόσεις της Java για τα MIPS, Alpha or PowerPC based NT, Windows 3.1, Amiga.

Το βασικό περιβάλλον της Java αποτελείται από έναν web browser, ο οποίος μπορεί να εκτελεί τις μίνι εφαρμογές της Java, έναν compiler που μετατρέπει τον πηγαίο κώδικα της Java σε κώδικα byte, κι έναν μεταφραστή της Java για να εκτελεί τα προγράμματα. Αυτά είναι τα τρία συστατικά-κλειδιά ενός περιβάλλοντος Java. Επίσης απαραίτητος είναι ένας text editor όπως το Brief ή το BBEdit.

Η Sun διαθέτει το Java Developers Kit (JDK). Περιέχει έναν applet viewer όπου θα μπορείτε να δείτε και να ελέγξετε τις εφαρμογές σας. Το JDK περιλαμβάνει επίσης τον javac compiler, τον java interpreter, τον javaprof profiler, τον Java debugger και περιορισμένα κείμενα. Τα περισσότερα από τα κείμενα για το API και τη βιβλιοθήκη κλάσης είναι στο web site της Sun.

Μπορείτε να βρείτε τα προγράμματα στα ακόλουθα sites:

- USA
  - ftp://ftp.javasoft.com/pub/
  - ftp://www.blackdown.org/pub/Java/pub/
  - ftp://ftp.science.wayne.edu/pub/java/
  - ftp://metalab.unc.edu/pub/languages/java/
  - ftp://java.dnx.com/pub/JDK-beta-win32-x86.exe
- Germany: ftp://sunsite.informatik.rwth-aachen.de/pub/mirror/java.sun.com/
- Korea: ftp://ftp.kaist.ac.kr/pub/java/

- China: ftp://math01.math.ac.cn/pub/sunsite/
- Japan: ftp://ftp.glocom.ac.jp/mirror/java.sun.com/
- Sweden: ftp://ftp.luth.se/pub/infosystems/www/hotjava/pub/
- Singapore: ftp://ftp.iss.nus.sg/pub/java/
- United Kingdom: ftp://sunsite.doc.ic.ac.uk/packages/java/

ΟΔΗΓΙΕΣ ΕΓΚΑΤΑΣΤΑΣΗΣ ΣΕ WINDOWS

Θα χρειαστείτε περίπου 6 MB ελεύθερα στο δίσκο για την εγκατάσταση του JDK. Εκτελέστε το αρχείο κάνοντας διπλό κλικ πάνω του στο File Manager ή επιλέγοντας Run... από το Program Manager's File menu και πληκτρολογώντας το μονοπάτι στο αρχείο. Προτείνουμε να το εγκαταστήσετε στο C:drive. Σ' αυτήν την περίπτωση τα αρχεία θα βρίσκονται στο C:\java. Θα πρέπει να προσθέσετε το C:\java\bin directory στο path environment.

Η αρχειοθέτηση περιλαμβάνει δύο κοινά DLL's:

- MSVCRT20.DLL
- MFC30.DLL

Αυτά τα 2 αρχεία θα εγκατασταθούν στο java directory. Αν δεν έχετε ήδη αντίγραφα αυτών στο σύστημά σας, αντιγράψτε τα στο C:\java\bin directory. Αν τα έχετε απλά διαγράψτε τα επιπλέον αντίγραφα.

ΤΡΕΧΟΝΤΑΣ ΤΗΝ ΠΡΩΤΗ ΣΑΣ ΜΙΝΙ ΕΦΑΡΜΟΓΗ (APPLET)

Οδηγίες στο Unix

Ξεκινήστε τον Applet Viewer κάνοντας τα ακόλουθα:

1. Ανοίξτε ένα command line prompt και cd σε ένα από τα directories στο /usr/local/java/demo. Για παράδειγμα

% cd /usr/local/java/demo/TicTacToe

2. Τρέξτε τον appletviewer στο αρχείο html:

% appletviewer example1.html

3. Παίξτε Tic-Tac-Toe!

Οδηγίες στα Windows

Ξεκινήστε τον Applet Viewer κάνοντας τα ακόλουθα:

1. Ανοίξτε ένα παράθυρο DOS και cd σε ένα από τα directories στο C:\JAVA\DEMO. Για παράδειγμα

C:> cd C:\JAVA\DEMO\TicTacToe

2. Τρέξτε τον appletviewer στο αρχείο html:

C:> appletviewer example1.htm

3. Παίξτε Tic-Tac-Toe!

Applets στο Netscape

Ο Netscape 3.0 και οι μεταγενέστεροι θα τρέξουν τις μίνι εφαρμογές της Java σχεδόν παντού εκτός από τα Windows 3.1. Ο Netscape έχει μία σελίδα, την JAVA DEMO PAGE, με συνδέσεις σε διάφορες εφαρμογές, από τις οποίες οι περισσότερες τρέχουν. Παρόλ' αυτά μην ξαφνιαστείτε αν κάποια εφαρμογή δεν δουλέψει κανονικά στον Netscape.

ΚΕΦΑΛΑΙΟ 2. Σύνταξη

Η ΕΦΑΡΜΟΓΗ HELLO WORLD

Από την πρώτη έκδοση των Kernighan και Ritchie «The C Programming Language» είναι πια συνηθισμένο να ξεκινούν τα εγχειρίδια προγραμματισμού με το πρόγραμμα «Hello World», ένα πρόγραμμα που τυπώνει στη οθόνη το αλφαριθμητικό «Hello World». Θα ξεκινήσουμε επηρεασμένοι από τους Kernighan και Ritchie με τον ίδιο τρόπο, χωρίς να διαφοροποιηθούμε από την παράδοση.

Ακολουθεί η εφαρμογή «Hello World» γραμμένη σε Java. Πληκτρολογείστε το σε ένα αρχείο κειμένου ή αντιγράψτε το από το web browser και σώστε το σ' ένα αρχείο με το όνομα *HelloWorld.java.*

```
class HelloWorld {

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

```
    System.out.println("Hello World!");
  }
}
```

Για να μεταγλωτίσετε αυτό το πρόγραμμα, σιγουρευτείτε ότι είστε στο ίδιο directory με το HelloWorld.java και πληκτρολογείστε javac HelloWorld.java στο command prompt. Το *Hello World* είναι ίσως το πιο απλό πρόγραμμα που μπορεί κανείς να φανταστεί. Παρόλο που δε διδάσκει πολλά από προγραμματιστική άποψη, μας δίνει την ευκαιρία να μάθουμε τους μηχανισμούς για να γράφουμε και να μεταγλωτίσουμε κώδικα. Ακολουθούν μερικά από τα πιο συνηθισμένα λάθη :

1. Τοποθετήσατε ; μετά το System.out.println("Hello World");
2. Συμπεριλάβατε την παρένθεση;
3. Πληκτρολογήσατε ό,τι βλέπετε με ακρίβεια; Ειδικότερα τηρήσατε με ακρίβεια τα κεφαλαία και τα μικρά γράμματα; Η Java είναι ευαίσθητη. Για παράδειγμα δεν είναι το ίδιο class και Class.
4. Βρισκόσασταν στο ίδιο directory HelloWorld.java όταν πληκτρολογήσατε το javac HelloWorld.java;

Όταν το πρόγραμμα μεταγλωτιστεί επιτυχώς, ο compiler τοποθετεί την εκτελούμενη έξοδο σε ένα αρχείο που ονομάζεται HelloWorld.class στο ίδιο directory. Έτσι εσείς μπορείτε να τρέχετε το πρόγραμμα πληκτρολογώντας java HelloWorld στο command prompt. Όπως πιθανόν να μαντέψατε το πρόγραμμα ανταποκρίνεται τυπώνοντας «Hello World!» στην οθόνη. Συγχαρητήρια! Μόλις γραψατε το πρώτο σας πρόγραμμα σε Java.

ΕΞΕΤΑΖΟΝΤΑΣ ΤΟ HELLO WORLD

Το *Hello World* είναι ίσως το πιο απλό πρόγραμμα που μπορεί κανείς να φανταστεί. Παρόλ' αυτά μας ενδιαφέρει για πολλούς λόγους. Ας το εξετάσουμε γραμμή προς γραμμή.

Η αρχική δήλωση class μπορεί να θεωρείται ότι προσδιορίζει το όνομα του προγράμματος, στη συγκεκριμένη περίπτωση Hello World. Ο compiler στην πραγματικότητα παίρνει το όνομα από τη δήλωση class HelloWorld στο αρχείο πηγαίου κώδικα. Αν υπάρχουν παραπάνω από μία κλάση σε ένα αρχείο, τότε ο compiler της Java θα αποθηκεύσει το καθένα σ' ένα ξεχωριστό .class αρχείο. Για λόγους που θα δούμε παρακάτω είναι συνετό να δίνουμε στο αρχείο του πηγαίου κώδικα το ίδιο όνομα με τη main class στο αρχείο συν την κατάληξη .java.

Το Hello World class περιλαμβάνει μία μέθοδο, τη main. Όπως και στη C, η μέθοδος main μας δείχνει από που ξεκινά να εκτελείται μία εφαρμογή. Η μέθοδος 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.

Όταν καλείται η μέθοδος main, τυπώνει το «Hello World!» 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. Αλλά για την ώρα θα την θεωρούμε σαν μία μέθοδο.

Μια τελευταία σημείωση. Αντίθετα με την printf στη C, η μέθοδος System.out.println προσθέτει μια καινούρια γραμμή στο τέλος της εξόδου. Έτσι δεν είναι ανάγκη να συμπεριλάβουμε το \n στο τέλος κάθε αλφαριθμητικού.

ΑΣΚΗΣΕΙΣ

1. Τι συμβαίνει αν αλλάξουμε το όνομα στο αρχείο του πηγαίου κώδικα π.χ HelloEarth.java αντί για HelloWorld.java;
2. Τι συμβαίνει αν διατηρήσουμε το όνομα του αρχείου πηγαίου κώδικα (HelloWorld.java), αλλά αλλάξουμε το όνομα της κλάσης π.χ class HelloEarth;

ΑΓΚΙΣΤΡΑ ΚΑΙ ΜΠΛΟΚ (BRACES AND BLOCKS)

Ας εξετάσουμε το πρόγραμμα Hello World λίγο πιο αναλυτικά. Στη Java το αρχείο πηγαίου κώδικα σπάει σε κομμάτια που χωρίζονται μεταξύ τους με παρενθέσεις, αγκύλες και άγκιστρα. Ό,τι υπάρχει μεταξύ { και } είναι ένα μπλοκ και υπάρχει λίγο ή πολύ ανεξάρτητα από οτιδήποτε άλλο έξω από τα άγκιστρα.

Τα blocks έχουν μεγάλη συντακτική και λογική σημασία. Χωρίς τα άγκιστρα ο κώδικας δεν θα μπορούσε να συνταχθεί. Ο compiler θα δυσκολευόταν να ξεχωρίσει το τέλος της μιας μεθόδου και την αρχή της επόμενης. Παράλληλα θα ήταν πολύ δύσκολο για κάποιον που διάβαζε τον κώδικά σας να καταλάβει τι συμβαίνει ή ακόμα θα ήταν δύσκολο και για εσάς τους ίδιους. Τα άγκιστρα χρησιμοποιούνται για να ομαδοποιούν τις σχετιζόμενες δηλώσεις. Γενικότερα, ό,τι βρίσκεται μεταξύ αγκίστρων εκτελείται σαν μία δήλωση. Τα blocks μπορεί να είναι ιεραρχικά. Ένα block μπορεί να περιέχει ένα ή περισσότερα θυγατρικά blocks. Σ' αυτήν την περίπτωση έχουμε ένα εξωτερικό block που προσδιορίζει το HelloWorld class. Ανάμεσα στο HelloWorld block έχουμε ένα block μεθόδου που λέγεται «main».

ΣΧΟΛΙΑ (COMMENTS)

Τα σχόλια μπορούν να εμφανιστούν οπουδήποτε σ' ένα αρχείο. Τα σχόλια δηλώνονται με τον ίδιο τρόπο όπως στη C και στη C++. Ό,τι υπάρχει μεταξύ /*και*/ αγνοείται από τον compiler. Ό,τι υπάρχει σε μία γραμμή μετά από δύο συνεχόμενα slashes επίσης θεωρείται σαν σχόλιο. Γι' αυτό και το ακόλουθο πρόγραμμα είναι, όσο αφορά τον compiler ίδιο με το πρώτο :

```
// This is the Hello World program in Java
class HelloWorld {

    public static void main (String args[]) {
      /* Now let's print the line Hello World */
      System.out.println("Hello World");

  }

}
```

ΔΕΔΟΜΕΝΑ ΚΑΙ ΜΕΤΑΒΛΗΤΕΣ (DATA AND VARIABLES)

Οι μέθοδοι είναι το μισό της Java. Το άλλο μισό είναι τα δεδομένα. Θεωρείστε την παρακάτω γενίκευση του προγράμματος HelloWorld :

```
// This is the Hello Rusty program in Java
class HelloRusty {

    public static void main (String args[]) {

      // You may feel free to replace "Rusty" with your own name
      String name = "Rusty";

      /* Now let's say hello */
      System.out.print("Hello ");
      System.out.println(name);
  }

}
```

Εδώ, επιτρέπουμε στη Java να χαιρετίσει ένα συγκεκριμένο άτομο και όχι γενικά τον κόσμο. Αυτό γίνεται δημιουργώντας μία μεταβλητή αλφαριθμητικού που την ονομάζουμε «name» και αποθηκεύοντας την τιμή «Rusty» σ' αυτήν (Στη θέση της τιμής Rusty μπορούμε να επιλέξουμε οποιοδήποτε όνομα). Μετά τυπώνουμε το «Hello». Παρατηρήστε ότι αλλάξαμε τη μέθοδο System.out.println με την παρόμοια System.out.print. Η System.out.print είναι ακριβώς ίδια με την System.out.println με τη μόνη διαφορά ότι δεν σπάει τη γραμμή αφού τελειώσει. Γι' αυτό, όταν φτάνουμε στην επόμενη γραμμή του κώδικα, ο δρομέας βρίσκεται ακόμα στη γραμμή της λέξης «Hello» και είμαστε έτοιμοι να τυπώσουμε όνομα.

ΟΡΙΣΜΑΤΑ ΓΡΑΜΜΩΝ ΕΝΤΟΛΩΝ (COMMAND LINE ARGUMENTS)

Το πρόγραμμά μας δεν είναι ακόμα πολύ γενικό, δε μπορούμε να αλλάξουμε το όνομα στο οποίο λέμε Hello, χωρίς να επανασυντάξουμε και να μεταγλωτίσουμε τον κώδικα. Αυτό μπορεί να μην ενοχλεί τους προγραμματιστές, αλλά τι θα γίνει αν μια γραμματέας θελήσει ο υπολογιστής να λέει Hello σ' αυτήν;

Αυτό που χρειάζεται είναι ένας τρόπος να αλλάζουμε το όνομα κατά τη διάρκεια που τρέχει το πρόγραμμα (δηλαδή όταν πληκτρολογούμε java HelloRusty). Για να γίνει αυτό θα χρησιμοποιήσουμε τα ορίσματα γραμμών εντολών (command line arguments). Μας επιτρέπουν να πληκτρολογούμε Java Hello Gloria και το πρόγραμμα να ανταποκρίνεται δίνοντας σαν έξοδο «Hello Gloria». Ακολουθεί ο κώδικας :

```
// This is the Hello program in Java
class Hello {

    public static void main (String args[]) {

      /* Now let's say hello */
      System.out.print("Hello ");
      System.out.println(args[0]);
  }

}
```

Μεταγλωτίστε το πρόγραμμα στο java directory όπως συνήθως και μετά πληκτρολογήστε java Hello Gloria.

Στην πραγματικότητα απαλλαχτήκαμε από τη μεταβλητή ονόματος του προγράμματος HelloRusty. Στη θέση της χρησιμοποιούμε args[0]. Args είναι αυτό που γνωρίζατε ως τώρα σαν πίνακα. Ένας πίνακας αποθηκεύει μια σειρά από τιμές. Οι τιμές αυτές μπορεί να είναι αλφαριθμητικά, όπως στο παράδειγμά μας, αριθμοί ή οποιαδήποτε άλλη μορφή δεδομένων της Java.

Args είναι ένας ειδικός πίνακας που κρατάει τα ορίσματα της γραμμής εντολών (command line arguments). Το args[0] κρατάει το πρώτο command line argument. Το args[1] κρατάει το δεύτερο command line argument και ούτω καθεξής. Αυτή τη στιγμή ίσως κάτι να μην σας φαίνεται σωστό. Αν δεν έχετε προγραμματίσει ξανά ή αν έχετε προγραμματίσει μόνο σε Pascal ή σε Fortran, πιθανόν να αναρωτιέστε γιατί το πρώτο στοιχείο του πίνακα είναι στη θέση 0, το δεύτερο στη θέση 1, το τρίτο στη θέση 2, αντί να είναι το πρώτο στοιχείο στη θέση 1 και ούτω καθεξής. Αυτό συμβαίνει και στη C.

Από την άλλη μεριά, αν έχετε συνηθίσει να προγραμματίζετε σε C, πιθανόν να αναρωτιέστε γιατί το args[0] είναι το πρώτο command line argument και όχι το command name. Το πρόβλημα στη Java είναι ότι δεν είναι πάντα φανερό ποιό είναι το command name. Στο παράδειγμά μας παραπάνω είναι το java ή το Hello;

Τώρα θα πρέπει να πειραματιστούμε λίγο με το πρόγραμμα. Τι συμβαίνει αν αντί να πληκτρολογήσουμε java Hello Gloria, πληκτρολογήσουμε java Hello Gloria and Beth; Τι συμβαίνει αν δεν χρησιμοποιήσουμε καθόλου όνομα π.χ java Hello;

Δεν ήταν ενδιαφέρον; Πιθανόν να έχετε δει κάτι παρόμοιο με το: Exception in thread «main» java.lang.ArrayIndexOutOfBoundsException at Hello.main(C:\javahtml\Hello.java:7). Αυτό που συμβαίνει είναι ότι από τη στιγμή που δεν δίνουμε στο Hello κανένα command line argument δεν υπάρχει τίποτα στο args[0].

IF

Ακόμα και τα πιο ασήμαντα προγράμματα πρέπει να παίρνουν αποφάσεις. Πρέπει να ελέγχουν κάποιες συνθήκες και να λειτουργούν διαφορετικά, βασιζόμενα σ' αυτές τις συνθήκες. Αυτό είναι συνηθισμένο στην πραγματική ζωή. Για παράδειγμα, βγάζετε το χέρι σας έξω από το παράθυρο για να διαπιστώσετε αν βρέχει. Αν βρέχει παίρνετε μαζί σας ομπρέλα, διαφορετικά όχι.

Όλες οι γλώσσες προγραμματισμού έχουν μορφές δηλώσεων if που επιτρέπουν να εξετάζουμε συνθήκες. Στον προηγούμενο κώδικα θα έπρεπε να είχαμε ελέγξει αν υπήρχαν command line arguments προτού προσπαθήσουμε να τα χρησιμοποιήσουμε.

Όλοι οι πίνακες έχουν μήκη και γι' αυτό χρησιμοποιούμε τη μεταβλητή arrayname.length. Ελέγχουμε το μήκος του args παρακάτω:

```
// This is the Hello program in Java
class Hello {

    public static void main (String args[]) {

      /* Now let's say hello */
      System.out.print("Hello ");
      if (args.length > 0) {
        System.out.println(args[0]);
      }
  }

}
```

Μεταγλωτίστε και τρέξτε το πρόγραμμα, δίνοντας διαφορετικές εισόδους κάθε φορά. Θα πρέπει να παρατηρήσετε ότι δεν είναι πια ένα ArrayIndexOutOfBoundsException αν δεν δώσετε command line arguments.

Αυτό που κάναμε ήταν να βάλουμε τη δήλωση System.out.println(args[0]) σε έναν έλεγχο υπόθεσης if (args.length>0){ }. Ο κώδικας μεταξύ των αγκίστρων System.out.println(args[0]) τώρα πια εκτελείται αν και μόνο αν το μήκος των args είναι μεγαλύτερο από το 0. Στην Java χρησιμοποιούμε το >,που σημαίνει μεγαλύτερο από, το < που σημαίνει μικρότερο από, το <= και το >=.

Θα περιμένατε ότι ο έλεγχος της ισότητας δύο αριθμών πραγματοποιείται με το σύμβολο =. Εμείς ήδη χρησιμοποιήσαμε το σύμβολο = για να θέσουμε τιμές σε μία μεταβλητή. Γι' αυτό χρειαζόμαστε ένα καινούριο σύμβολο για να ελέγχουμε την ισότητα. Έτσι η Java δανείζεται από τη C τον συμβολισμό ==.

Δεν είναι ασυνήθιστο ακόμα και για τους πιο έμπειρους προγραμματιστές να γράφουν == όταν εννοούν = και το αντίστροφο. Αυτό είναι ένα πολύ συνηθισμένο λάθος στα προγράμματα της C. Ευτυχώς στη Java δεν επιτρέπεται να χρησιμοποιούμε το == και το = στα ίδια σημεία. Έτσι ο compiler μπορεί να αντιληφθεί το λάθος κι εσείς να το διορθώσετε, προτού τρέξετε το πρόγραμμα.

Όλες οι δηλώσεις συνθήκης στη Java ζητούν τιμές boolean κι αυτές επιστρέφουν οι τελεστές ==, >, <,>=,<=. Boolean είναι μια τιμή που είναι true ή false. Αν θέλετε να θέσετε μία μεταβλητή boolean σε ένα πρόγραμμα Java, πρέπει να χρησιμοποιήσετε τις σταθερές true και false. Το false δεν είναι 0 και το true δεν είναι όχι 0, όπως στη C.

Οι έμπειροι προγραμματιστές πιθανόν να επισημάνουν ότι υπάρχει μια εναλλακτική μέθοδος να χειριστούμε το ArrayIndexOutOfBoundsException με τις δηλώσεις try και catch. Θα επιστρέψουμε σ' αυτό σύντομα.

ELSE

Ίσως να παρατηρήσατε ένα μικρό σφάλμα (cosmetic bug) στο προηγούμενο πρόγραμμα. Ένα cosmetic bug δεν σπάει το πρόγραμμα ή το σύστημα, ούτε παράγει λανθασμένα αποτελέσματα αλλά απλά ενοχλεί.

Το cosmetic bug εδώ ήταν ότι αν δεν περιλαμβάναμε κανένα command line argument, το πρόγραμμα δεν θα έσπαζε, αλλά θα τύπωνε το «Hello» και δεν θα άλλαζε γραμμή. Το πρόβλημα ήταν ότι χρησιμοποιήσαμε System.out.print και όχι System.out.println. Δεν υπήρχε χαρακτήρας τέλους γραμμής. Ήταν σαν να πληκτρολογήσαμε αυτό που θέλαμε χωρίς να πατήσουμε το enter.

Αυτό θα μπορούσε να διορθωθεί αν τοποθετούσαμε το System.out.println(«»); στο τέλος της μεθόδου main αλλά έτσι θα είχαμε πολλά end-of-lines αν ο χρήστης πληκτρολογούσε ένα όνομα. Θα μπορούσαμε να προσθέσουμε μία δήλωση if. Έτσι θα είχαμε:

```
// This is the Hello program in Java
class Hello {

    public static void main (String args[]) {

      /* Now let's say hello */
      System.out.print("Hello ");
      if (args.length > 0) {
        System.out.println(args[0]);
      }
      if (args.length <= 0) {
        System.out.println("whoever you are");
      }
  }

}
```

Αυτό διορθώνει το σφάλμα, αλλά είναι δύσκολο να διαβαστεί και να εκτελεστεί ο κώδικας. Είναι εύκολο να χάσουμε μια πιθανή περίπτωση. Για παράδειγμα, μπορεί να ελέγξουμε αν το args.length είναι μικρότερο από το 0 και να αφήσουμε την πιο ενδιαφέρουσα περίπτωση δηλαδή όταν το args.length είναι ίσο με το 0. Αυτό που χρειαζόμαστε είναι μια δήλωση else. Ακολουθεί η σωστή λύση:

```
// This is the Hello program in Java
class Hello {

    public static void main (String args[]) {

      /* Now let's say hello */
      System.out.print("Hello ");
      if (args.length > 0) {
        System.out.println(args[0]);
      }
      else {
        System.out.println("whoever you are");
      }
  }

}
```

Τώρα που το Hello τουλάχιστον αποφεύγει το ArrayIndexOutOfBoundsException, δεν έχουμε τελειώσει ακόμα. Το java Hello και το Java Hello Rusty δουλεύουν, αλλά αν πληκτρολογήσουμε java Hello Elliotte Rusty Harold, η Java τυπώνει μόνο Hello Elliotte. Ας το διορθώσουμε.

Δεν περιοριζόμαστε σε δύο περιπτώσεις. Μπορούμε να συνδυάσουμε ένα else και ένα if δημιουργώντας ένα else if και να το χρησιμοποιήσουμε για να εξετάσουμε μια πληθώρα από αμοιβαία αποκλειόμενες πιθανότητες. Για παράδειγμα, ακολουθεί μία έκδοση του προγράμματος Hello που χειρίζεται τέσσερα ονόματα :

```
// This is the Hello program in Java
class Hello {

    public static void main (String args[]) {

      /* Now let's say hello */
      System.out.print("Hello ");
      if (args.length == 0) {
        System.out.print("whoever you are");
      }
      else if (args.length == 1) {
        System.out.println(args[0]);
      }
      else if (args.length == 2) {
        System.out.print(args[0]);
        System.out.print(" ");
        System.out.print(args[1]);
      }
      else if (args.length == 3) {
        System.out.print(args[0]);
```

```
        System.out.print(" ");
        System.out.print(args[1]);
        System.out.print(" ");
        System.out.print(args[2]);
      }
      else if (args.length == 4) {
        System.out.print(args[0]);
        System.out.print(" ");
        System.out.print(args[1]);
        System.out.print(" ");
        System.out.print(args[2]);
        System.out.print(" ");
        System.out.print(args[3]);
      }
      else {
        System.out.print(args[0]);
        System.out.print(" ");
        System.out.print(args[1]);
        System.out.print(" ");
        System.out.print(args[2]);
        System.out.print(" ");
        System.out.print(args[3]);
        System.out.print(" and all the rest!");
       }
      System.out.println();
  }

}
```

Ένας μη έμπειρος προγραμματιστής της Java θα έγραφε κώδικα σαν τον παραπάνω. Ένας από τους λόγους που κάνουν αυτή τη λύση δύσχρηστη, είναι ότι χρησιμοποιούμε για κάθε μεταβλητή διαφορετική δήλωση που θα την τυπώνει. Η Java παρόλ' αυτά τυπώνει πολλαπλά πράγματα με μία δήλωση. Αντί να περιλαμβάνει μόνο ένα όνομα στο print argument, μπορεί να περιλαμβάνει πολλά που να χωρίζονται μεταξύ τους με +. Αυτά μπορούν να περιλαμβάνουν μεταβλητές όπως args[0] και σταθερά αλφαριθμητικά όπως «and all the rest!». Για παράδειγμα, το τελευταίο block θα μπορούσε να γραφεί :

```
else {
  System.out.print(args[0] + " " + args[1] + " " + args[2] + " " +
args[3] + " and all the rest!");
}
```

Αυτή η σύνταξη είναι πιο εύκολο να διαβαστεί και να γραφτεί αλλά παραμένει δύσχρηστη στην περίπτωση που τα command line arguments αυξηθούν. Στην επόμενη ενότητα θα δούμε πώς να χειριζόμαστε πάνω από δύο δισεκατομμύρια command line arguments με απλό τρόπο.

ΑΣΚΗΣΕΙΣ

1. Ξαναγράψτε ολόκληρο το πρόγραμμα χρησιμοποιώντας μία μόνο μέθοδο τυπώματος σε κάθε block
2. Μία άλλη λύση στο πρόβλημα που δεν την είδαμε ακόμα είναι χρησιμοποιώντας το for. Υπάρχει μία ακόμα πιο αποτελεσματική μέθοδος που δεν χρησιμοποιεί το +, παρα μόνο το if's και ένα else. Δεν χρειάζονται else if's. Ποια είναι αυτή;

ΜΕΤΑΒΛΗΤΕΣ ΚΑΙ ΑΡΙΘΜΗΤΙΚΕΣ ΕΚΦΡΑΣΕΙΣ (VARIABLES AND ARITHMETIC EXPRESSIONS)

Στο κεφάλαιο αυτό θα εισάγουμε την έννοια του βρόγχου (loop). Loop είναι ένα τμήμα κώδικα που εκτελείται συνεχώς μέχρι να συναντήσει μια συνθήκη τερματισμού. Ένα συνηθισμένο loop είναι το ακόλουθο:

```
while there's more data {
  Read a Line of Data
  Do Something with the Data
}
```

Υπάρχουν πολλά, διαφορετικά είδη loops στη Java, συμπεριλαμβανομένων των while, for και do while.

Η for ακολουθεί ένα προκαθορισμένο αριθμό επαναλήψεων και μετά σταματά. Η while επαναλαμβάνεται ασταμάτητα μέχρι να συναντήσει μια συγκεκριμένη συνθήκη. Συνήθως δεν γνωρίζουμε προηγουμένως, πόσες φορές θα επαναληφθεί ένα while loop.

Σ' αυτήν την περίπτωση θέλουμε να γράψουμε ένα βρόγχο που θα τυπώνει καθένα από τα command line arguments ξεκινώντας από το πρώτο. Δεν ξέρουμε προηγουμένως πόσα command line arguments θα είναι, αλλά μπορούμε να το μάθουμε εύκολα προτού ο βρόγχος αρχίσει να χρησιμοποιεί το args.length.Γι' αυτό και θα χρησιμοποιήσουμε τον βρόγχο for. Ακολουθεί ο κώδικας:

```
// This is the Hello program in Java
class Hello {

    public static void main (String args[]) {

      int i;

      /* Now let's say hello */
      System.out.print("Hello ");
      for (i=0; i < args.length; i = i+1) {
        System.out.print(args[i]);
        System.out.print(" ");
      }
      System.out.println();
  }
```

```
}
```

Ξεκινάμε τον κώδικα δηλώνοντας τις μεταβλητές μας. Σ' αυτήν την περίπτωση έχουμε μία μόνο μεταβλητή, την ακέραια i.

Μετά ξεκινάμε το πρόγραμμα γράφοντας «Hello» όπως προηγουμένως. Έπειτα βλέπουμε τον βρόγχο for. Ο βρόγχος ξεκινά αρχικοποιώντας τον μετρητή i να είναι 0. Αυτό συμβαίνει μία φορά στην αρχή του βρόγχου. Η προγραμματιστική παράδοση που βασίζεται στη Fortran επιμένει ότι οι μεταβλητές του βρόγχου πρέπει να ονομάζονται με τη σειρά i, j, k, l, m, n. Αυτό είναι μια σύμβαση και όχι χαρακτηριστικό της γλώσσας Java. Παρόλ' αυτά όμως όποιος διαβάσει τον κώδικά σας θα περιμένει να ακολουθήσετε τη σύμβαση. Αν επιλέξετε να αγνοήσετε τη σύμβαση, προσπαθήστε να δώσετε στις μεταβλητές μνημονικά ονόματα, όπως counter και loop_index.

Ακολουθεί η εξέταση της υπόθεσης. Σ' αυτήν την περίπτωση εξετάζουμε αν το i είναι μικρότερο από τον αριθμό των arguments. Όταν το i γίνει ίσο με τον αριθμό των arguments (args.length), βγαίνουμε από τον βρόγχο και πηγαίνουμε στην πρώτη εντολή μετά το κλείσιμο του βρόγχου. Θυμηθείτε ότι ξεκινήσαμε να μετράμε από το 0, όχι από το 1.

Τέλος, έχουμε το προσαυξητικό βήμα i=i+1. Αυτό εκτελείται στο τέλος κάθε επανάληψης του βρόγχου. Χωρίς αυτό θα παραμέναμε στον βρόγχο για πάντα αφού το i θα ήταν μόνιμα μικρότερο από το args.length (εκτός φυσικά αν το args.length ήταν μικρότερο ή ίσο με το 0).

ΠΟΙΟ ΕΙΝΑΙ ΤΟ ΠΡΟΒΛΗΜΑ ΤΩΝ ΚΑΘΗΓΗΤΩΝ ΤΗΣ ΑΛΓΕΒΡΑΣ ΜΕ ΤΗΝ BASIC ΚΑΙ ΤΗ C?

Η δήλωση i=i+1 βρίσκει τους καθηγητές της άλγεβρας αντίθετους. Αυτό που δηλώνεται είναι άκυρο. Δεν υπάρχει κανένας αριθμός για τον οποίο η δήλωση i=i+1 να είναι αληθής. Στην πραγματικότητα, αν αφαιρέσουμε το i και από τις δύο πλευρές της ισότητας προκύπτει η λανθασμένη δήλωση 0=1. Το μυστικό εδώ είναι ότι το σύμβολο = δεν συμβολίζει ισότητα. Η ισότητα, όπως είπαμε και πιο πάνω συμβολίζεται με το ==. Σε όλες σχεδόν τις γλώσσες προγραμματισμού, συμπεριλαμβανομένης και της Java, το = είναι τελεστής εκχώρησης.

Μία αξιοπρόσεκτη εξαίρεση είναι η Pascal (καθώς επίσης και τα παράγωγά της Modula-2, Modula-3 και Oberon), όπου το = δηλώνει ισότητα ενώ τελεστής απόδοσης είναι το :=. Οι δάσκαλοι των μαθηματικών δεν θέλουν να βλέπουν το = να ταλαιπωρείται. Αυτός είναι ένας από τους λόγους που η Pascal είναι η πιο δημοφιλής γλώσσα για τη διδασκαλία προγραμματισμού. Αξιοσημείωτο είναι ότι οι καθηγητές των μαθηματικών είναι αντίθετοι με γλώσσες όπως η Basic όπου, ανάλογα με το περιεχόμενο, το = δηλώνει είτε απόδοση είτε ισότητα.

ΑΣΚΗΣΕΙΣ

1. Τι γίνεται αν δεν δώσουμε στο πρόγραμμα Hello κανένα command line argument; Εφόσον δεν ελέγχουμε πια τον αριθμό των command line arguments γιατί βγαίνει το μήνυμα ArrayIndexOutOfBoundsException;
2. Για κάποια συγκεκριμένα αριθμητικά συστήματα η δήλωση *i=i+1* έχει μια έγκυρη λύση. Ποια είναι αυτή;

ΚΛΑΣΕΙΣ ΚΑΙ OBJECTS: ΜΙΑ ΠΡΩΤΗ ΜΑΤΙΑ

Οι κλάσεις (classes) είναι τα πιο σημαντικά στοιχεία της Java. Τα πάντα στη Java είναι μια κλάση, ή ένα τμήμα μιας κλάσης ή περιγράφει το πώς συμπεριφέρεται μία κλάση. Παρόλο που οι κλάσεις θα μελετηθούν με λεπτομέρεια στο κεφάλαιο 4, είναι τόσο θεμελιώδεις για την κατανόηση των προγραμμάτων που μια μικρή εισαγωγή σ' αυτό το σημείο είναι απαραίτητη.

Όλο το νόημα στα προγράμματα της Java βρίσκεται στα class blocks, στην περίπτωση μας στο HelloWorld class. Οι μέθοδοι προσδιορίζονται από κλάσεις στις οποίες ανήκουν. Αυτό πιθανόν να προκαλέσει σύγχιση στους προγραμματιστές της C++ που έχουν συνηθίσει να προσδιορίζουν τις μεθόδους έξω από τα class blocks. Στη Java όλα τα συντακτικά και λογικά θέματα πραγματοποιούνται μέσα στην κλάση.

Ακόμα και τα πιο βασικά δεδομένα όπως οι ακέραιοι, συχνά είναι απαραίτητο να ενσωματωθούν σε κλάσεις προτού χρησιμοποιηθούν αποτελεσματικά. Κλάση είναι η θεμελιώδης μονάδα των προγραμμάτων της Java. Για παράδειγμα, παρακολουθήστε το παρακάτω πρόγραμμα :

```
class HelloWorld {

  public static void main (String args[]) {

    System.out.println("Hello World");

  }

}

class GoodbyeWorld {

  public static void main (String args[]) {

    System.out.println("Goodbye Cruel World!");

  }

}
```

Σώστε τον κώδικα σε ένα αρχείο που θα ονομάσετε hellogoodbye.java στο java directory και μεταγλωτίστε το με το javac hellogoodbye.java. Στη συνέχεια καταγράψτε τα περιεχόμενα του directory. Θα παρατηρήσετε ότι ο μεταγλωτιστής δημιούργησε δύο ξεχωριστά αρχεία κλάσης, το HelloWorld.class και το GoodbyeWorld.class.

Η δεύτερη κλάση είναι ένα ανεξάρτητο πρόγραμμα. Πληκτρολογείστε java GoodbyeWorld και μετά java HelloWorld. Αυτά τα προγράμματα τρέχουν και εκτελούνται ανεξάρτητα το ένα από το άλλο παρόλο που υπάρχουν στο ίδιο αρχείο πηγαίου κώδικα. Βέβαια δεν υπάρχει κάποιος λόγος να θέλουμε δύο διαφορετικά προγράμματα στο ίδιο αρχείο, αλλά αν συμβεί έχουμε τη δυνατότητα να το πραγματοποιήσουμε.

Είναι πιο πιθανό να θελήσουμε περισσότερες από μία κλάση στο ίδιο αρχείο. Θα συναντήσουμε αρχείο πηγαίου κώδικα με πολλές κλάσεις και μεθόδους. Στην πραγματικότητα υπάρχουν δηλώσεις που μπορούν, με την πρώτη ματιά, να εμφανιστούν έξω από μια κλάση. Οι δηλώσεις εισόδου εμφανίζονται στην αρχή ενός αρχείου έξω από κάθε class. Ο μεταγλωτιστής όμως τις αντικαθιστά με τα περιεχόμενα του αρχείου που αποτελείται από περισσότερες κλάσεις.

INTERFACES

Όπως είπαμε, οι κλάσεις είναι πολύ βασικό στοιχείο της Java. Παρόλ' αυτά υπάρχει κάτι στον κώδικα της Java που δεν είναι ούτε κλάση, ούτε τμήμα μιας κλάσης. Αυτό λέγεται διεπαφή. Δεν θα πούμε πολλά για τις διεπαφές μια και αφορούν πιο εξειδικευμένο κεφάλαιο. Απλά θα σημειώσουμε ότι μια διεπαφή προσδιορίζει τις μεθόδους που εφαρμόζει μία κλάση. Με άλλα λόγια δηλώνει τί κάνουν συγκεκριμένες κλάσεις. Μια διεπαφή από μόνη της δεν κάνει τίποτα. Όλη η δράση της πραγματοποιείται μέσα στις κλάσεις.

FahrToCelsius

Η Java δεν χρησιμοποιείται μόνο για το World Wide Web. Το πρόγραμμα που ακολουθεί αναφέρεται στην κλασική χρήση των υπολογιστών και μας γυρίζει στην εποχή των υπολογιστών με κάρτες μηχανής. Είναι ένα από τα πιο χρήσιμα προγράμματα που έχουν γραφτεί ποτέ. Σχεδιάστηκε για να υπολογίζει γρήγορα αριθμούς, οι οποίοι θα έπρεπε να υπολογιστούν με το χέρι στα εργαστήρια φυσικής.

```
// Print a Fahrenheit to Celsius table

class FahrToCelsius  {

  public static void main (String args[]) {

  int fahr, celsius;
```

```
  int lower, upper, step;

  lower = 0;      // lower limit of temperature table
  upper = 300;  // upper limit of temperature table
  step  = 20;     // step size

  fahr = lower;
  while (fahr <= upper) {  // while loop begins here
    celsius = 5 * (fahr-32) / 9;
    System.out.print(fahr);
    System.out.print(" ");
    System.out.println(celsius);
    fahr = fahr + step;
  } // while loop ends here
} // main ends here

} //FahrToCelsius ends here
```

Το πρόγραμμα αυτό υπολογίζει έναν πίνακα θερμοκρασιών Fahrenheit με τις ισοδύναμες θερμοκρασίες Celsius. Οι δύο πρώτες γραμμές της κύριας μεθόδου δηλώνουν τις μεταβλητές που θα χρησιμοποιήσουμε. Συγκεκριμενοποιούνται τα ονόματα και οι τύποι. Για την ώρα χρησιμοποιούμε μόνο ακέραιους. Στην Java ένας int μπορεί να πάρει τιμές από -2,147,483,648 μέχρι 2,147,483,647.

Στη συνέχεια αρχικοποιούμε τις μεταβλητές χρησιμοποιώντας δηλώσεις όπως lower=0. Αυτό θέτει την αρχική τιμή της lower ίση με 0.

Αφού θέσουμε αρχικές τιμές σε όλες μας τις μεταβλητές, πηγαίνουμε στον βρόγχο που κάνει τη βασική δουλειά στο πρόγραμμά μας. Στην αρχή κάθε επανάληψης του βρόγχου (fahr <= upper) ελέγχουμε να δούμε αν η τιμή του fahr είναι πράγματι μικρότερη ή ίση με την τρέχουσα τιμή του upper. Αν είναι, τότε ο υπολογιστής εκτελεί τις εντολές του βρόγχου (ό,τι υπάρχει δηλαδή μεταξύ του «while loop begins here» και «while loop ends here»). Οι βρόγχοι στη Java κλείνονται μέσα σε ζευγάρια αγκίστρων και μπορούν να είναι φωλιασμένοι.

celsius=5*(fahr-32)/9**;** Υπολογίζει τους βαθμούς Celsius έχοντας τους βαθμούς Fahrenheit. Οι αριθμητικοί τελεστές κάνουν ακριβώς αυτό που γνωρίζετε. Το * συμβολίζει τον πολλαπλασιασμό, το - την αφαίρεση, το / την διαίρεση και το + παρόλο που δεν το συναντήσαμε εδώ, συμβολίζει την πρόσθεση. Η προτεραιότητα των πράξεων ακολουθεί τους κανονικούς αλγεβρικούς κανόνες.

Η Java περιέχει ένα ολοκληρωμένο σύνολο αριθμητικών τελεστών. Όπως και στη C λείπει ένας εκθετικός τελεστής. Για τις εκθετικές πράξεις θα πρέπει να χρησιμοποιήσετε τις μεθόδους pow στο πακέτο java.lang.Math.

Η εκτύπωση της εξόδου είναι πολύ απλή. Χρησιμοποιούμε το System.out.print(fahr) για να τυπώσουμε τις τιμές Fahrenheit, μετά το System.out.print(" ") για να τυπώσουμε το κενό και τελικά το System.out.println(Celsius) για να τυπώσουμε τις τιμές Celsius.

ΜΕΤΑΒΛΗΤΕΣ ΚΙΝΗΤΗΣ ΥΠΟΔΙΑΣΤΟΛΗΣ (FLOATING POINT VARIABLES)

Θα παρατηρήσατε κάτι ανορθόδοξο στην παραπάνω έξοδο. Οι αριθμοί δεν ήταν ακριβώς σωστοί. Το 0 σε βαθμούς Fahrenheit είναι στην πραγματικότητα -17.778 βαθμούς Celsius και όχι -18°C όπως μας δίνει το πρόγραμμα. Το πρόβλημα είναι ότι χρησιμοποιήσαμε μόνο ακέραιους, όχι δεκαδικούς αριθμούς. Στη γλώσσα των υπολογιστών οι δεκαδικοί αριθμοί ονομάζονται «αριθμοί κινητής υποδιαστολής».

Οι αριθμοί κινητής υποδιαστολής μπορούν να αναπαραστήσουν μεγαλύτερο πλήθος τιμών απ' ότι οι ακέραιοι. Για παράδειγμα, μπορείτε να γράψετε πολύ μεγάλα νούμερα, όπως η ταχύτητα του φωτός (2.998E8 μέτρα ανά δευτερόλεπτο) αλλά και πολύ μικρά, όπως η σταθερά του Plank, χρησιμοποιώντας τον ίδιο αριθμό ψηφίων. Από την άλλη μεριά χάνεται μέρος από την ακρίβεια, που πιθανόν όμως να μην ήταν απαραίτητη για τόσο μεγάλα ή τόσο μικρά νούμερα.

Μερικές γλώσσες έχουν και ένα τρίτο είδος αριθμών που ονομάζονται αριθμοί σταθερής υποδιαστολής. Αυτοί οι αριθμοί έχουν μια δεδομένη ακρίβεια, για παράδειγμα δύο δεκαδικές θέσεις και είναι συχνά χρήσιμοι στους νομισματικούς υπολογισμούς. Η Java δεν έχει τέτοιο τύπο δεδομένων.

Η χρήση αριθμών κινητής υποδιαστολής δεν είναι δυσκολότερη από τη χρήση ακεραίων. Μπορούμε να κάνουμε το πρόγραμμά μας πιο ακριβές μετατρέποντας όλες τις int μεταβλητές σε double.

```
// Print a more accurate Fahrenheit to Celsius table

class FahrToCelsius  {

  public static void main (String args[]) {

    double fahr, celsius;
    double lower, upper, step;

    lower = 0.0;      // lower limit of temperature table
    upper = 300.0;  // upper limit of temperature table
    step  = 20.0;     // step size

    fahr = lower;
    while (fahr <= upper) {  // while loop begins here
      celsius = 5.0 * (fahr-32.0) / 9.0;
      System.out.print(fahr);
      System.out.print(" ");
      System.out.println(celsius);
      fahr = fahr + step;
    } // while loop ends here
  } // main ends here

} //FahrToCelsius ends here
```

Προσέξτε την αλλαγή στο πρόγραμμα. Όλες οι ακέραιες σταθερές όπως το 5 και το 9 έγιναν 5.0, 9.0 κτλ. Αν ένας σταθερός αριθμός περιέχει την υποδιαστολή, τότε ο μεταγλωτιστής θεωρεί ότι είναι ένας αριθμός κινητής υποδιαστολής. Αν όχι, τότε ο μεταγλωτιστής θεωρεί ότι είναι ακέραιος. Όταν όμως δύο αριθμοί διαφορετικού τύπου, για παράδειγμα ένας ακέραιος και ένας κινητής υποδιαστολής, βρίσκονται στη δεξιά πλευρά μιας ισότητας τότε ο μεταγλωτιστής προάγει τον αριθμό με τον πιο αδύναμο τύπο στον πιο δυνατό προτού κάνει τον υπολογισμό.

Τι κάνει τον τύπο ενός αριθμού δυνατότερο από κάποιον άλλο; Η δυνατότητα να αναπαραστήσει μια πιο διευρυμένη κλίμακα αριθμών. Εφόσον ένα byte μπορεί να αναπαραστήσει μόνο 256 αριθμούς είναι πιο αδύναμο από έναν short που μπορεί να αναπαραστήσει 65.537 διαφορετικούς αριθμούς, συμπεριλαμβανομένων κι εκείνων που αναπαριστά ένα byte. Όμοια, ένας int είναι ισχυρότερος από έναν short. Οι αριθμοί κινητής υποδιαστολής είναι δυνατότεροι από οποιονδήποτε ακέραιο τύπο και οι doubles είναι ο πιο ισχυρός τύπος απ' όλους.

Γι' αυτό θα μπορούσαμε να αφήσουμε όλες τις μικρές σταθερές σαν ακέραιους και η έξοδος του προγράμματος να μην άλλαζε καθόλου. Είναι καλύτερα όμως να βάζουμε την υποδιαστολή προκειμένου να θυμίζουμε σε όλους τι συμβαίνει.

Αυτά εφαρμόζονται σε υπολογισμούς που συμβαίνουν στη δεξιά πλευρά του =. Με την αριστερή πλευρά συμβαίνει κάτι άλλο. Οι προγραμματιστές έχουν δώσει ειδικές ονομασίες στην κάθε πλευρά. Η αριστερή ονομάζεται lvalue ενώ η δεξιά πλευρά ονομάζεται rvalue.

Η rvalue είναι ένα αριθμητικό αποτέλεσμα και όπως αναφέρθηκε πιο πάνω, λαμβάνει τον ισχυρότερο τύπο των αριθμών που συμμετέχουν στον υπολογισμό. Από την άλλη μεριά η lvalue έχει έναν τύπο που πρέπει να προσδιοριστεί προτού χρησιμοποιηθεί. Αυτό κάνουν οι δηλώσεις τύπου float fahr, celsius; Από τη στιγμή που ο τύπος της lvalue δηλωθεί, δεν αλλάζει ποτέ. Γι' αυτό αν δηλώσουμε το fahr σαν int, τότε στην αριστερή πλευρά της ισότητας το fahr θα είναι πάντα int, ποτέ float ή double ή long.

Προκύπτει το εξής ερώτημα : Τί γίνεται όταν ο τύπος στα αριστερά δεν συμπίπτει με τον τύπο στα δεξιά; Για παράδειγμα, έχουμε τον παρακάτω κώδικα :

```
class FloatToInt  {

  public static void main (String args[]) {

    int myInteger;

    myInteger = 9.7;

  } // main ends here

} //FloatToInt ends here
```

Δύο πράγματα μπορεί να συμβούν. Αν όπως πιο πάνω προσπαθούμε να μετατρέψουμε έναν αριθμό σε πιο αδύναμο τύπο μεταβλητής τότε ο compiler αναγνωρίζει το λάθος. Αντίθετα, αν προσπαθούμε να μετατρέψουμε έναν αδύναμο τύπο σε πιο ισχυρό τότε ο compiler κάνει τη μετατροπή. Για παράδειγμα, έχουμε τον παρακάτω μόνιμο κώδικα:

```
class IntToFloat  {

  public static void main (String args[]) {

    float myFloat;
    int   myInteger;

    myInteger = 9;
    myFloat = myInteger;
    System.out.println(myFloat);

  } // main ends here

} //IntToFloat ends here
```

ΑΣΚΗΣΕΙΣ

1. Προσθέστε στο FahrToCelsius μια επικεφαλίδα πριν τον πίνακα
2. Γράψτε ένα παρόμοιο πρόγραμμα που να μετατρέπει τους Celsius σε Fahrenheit

Η ΔΗΛΩΣΗ FOR

Στην Java μπορούμε σχεδόν πάντα να γράψουμε ένα πρόγραμμα με περισσότερους από έναν τρόπους. Το παρακάτω πρόγραμμα παράγει την ίδια έξοδο με το προηγούμενο. Η κύρια διαφορά είναι ο βρόγχος for στη θέση του βρόγχου while.

```
// Print a Fahrenheit to Celsius table

class FahrToCelsius  {

  public static void main (String args[]) {

    int fahr, celsius;
    int lower, upper, step;

    lower = 0;      // lower limit of temperature table
    upper = 300;  // upper limit of temperature table
    step  = 20;     // step size

    for (fahr=lower; fahr <= upper; fahr = fahr + step) {
      celsius = 5 * (fahr-32) / 9;
      System.out.println(fahr + " " + celsius);
    } // for loop ends here

  } // main ends here
```

```
}
```

Ο βρόγχος for έχει την ίδια σύνταξη με το for στη γλώσσα C. Για παράδειγμα, for (initialization; test; increment). Η αρχικοποίηση, που σ' αυτή την περίπτωση, θέτει τη μεταβλητή fahr ίση με το κατώτερο όριο, συμβαίνει μόνο την πρώτη φορά. Για κάθε άλλη φορά, όταν ο έλεγχος φτάνει στην αρχή του βρόγχου γίνεται ένας έλεγχος. Στο παράδειγμά μας ο έλεγχος είναι αν η μεταβλητή fahr είναι μικρότερη ή ίση με το ανώτερο όριο. Εάν είναι, τότε εκτελούμε τον κώδικα του βρόγχου άλλη μία φορά. Εάν όχι τότε εκτελούμε τον κώδικα που ακολουθεί τον βρόγχο. Όταν φτάνουμε στο τέλος του βρόγχου, γίνεται κάθε φορά προσαύξηση του βήματος. Σ' αυτή την περίπτωση αυξάνουμε το fahr κατά step.

Αν δεν είναι κατανοητό ας δούμε το παρακάτω απλό παράδειγμα:

```
//Count to ten

class CountToTen  {

  public static void main (String args[]) {

    int i;
    for (i=1; i <=10; i = i + 1) {
      System.out.println(i);
    }
    System.out.println("All done!");

 }

}
```

Αυτό το πρόγραμμα τυπώνει τους αριθμούς από το 1 μέχρι το 10. Ξεκινά θέτοντας τη μεταβλητή i ίση με 1. Έπειτα ελέγχει αν το 1 είναι πράγματι μικρότερο ή ίσο με το 10. Εφόσον το 1 είναι μικρότερο του 10, το πρόγραμμα το τυπώνει. Στο τέλος προσθέτει μία μονάδα στο i και αρχίζει από την αρχή. Το i τώρα είναι 2. Το πρόγραμμα ελέγχει αν το 2 είναι μικρότερο ή ίσο του 10. Επειδή είναι μικρότερο, τυπώνει το «2» και προσθέτει άλλη μια μονάδα στο i. Το i τώρα είναι 3. Για άλλη μια φορά ο κώδικας ελέγχει αν το 3 είναι μικρότερο ή ίσο του 10 κτλ μέχρι το i να γίνει 10. Ο κώδικας ελέγχει αν το 10 είναι μικρότερο ή ίσο του 10. Είναι ίσο, άρα τυπώνεται το «10» και το i, αφού προσθέσουμε άλλη μία μονάδα γίνεται 11. Το 11 όμως δεν είναι μικρότερο ή ίσο του 10 άρα το πρόγραμμα δεν το τυπώνει. Αντί γι' αυτό κινείται στην επόμενη εντολή που ακολουθεί τον βρόγχο for.

System.out.println("All done!");. Ο υπολογιστής τυπώνει «All done!» και το πρόγραμμα τελείωνει.

Οι βρόγχοι for δεν δουλεύουν πάντα ομαλά. Για παράδειγμα, θεωρείστε το παρακάτω πρόγραμμα:

```
//Count to ten??
```

```
class BuggyCountToTen  {

  public static void main (String args[]) {

    int i;
    for (i=1; i <=10; i = i - 1) {
      System.out.println(i);
    }
    System.out.println("All done!");

 }

}
```

Αυτό το πρόγραμμα μετράει ανάποδα. Δεν υπάρχει τίποτα λάθος με ένα πρόγραμμα που μετράει ανάποδα, απλά εξετάζουμε αν το i είναι μεγαλύτερο από το 10. Εφόσον το i δεν πρόκειται ποτέ να γίνει μεγαλύτερο από το 10 σ' αυτό το πρόγραμμα, το πρόγραμμα δεν σταματά ποτέ. (Αυτό δεν αληθεύει πάντα. Αν έχουμε πολύ γρήγορο μηχάνημα ή περιμένουμε πολύ, κάπου στο -2.000.000.000, το i θα γίνει ξαφνικά ένας πολύ μεγάλος θετικός αριθμός και το πρόγραμμα θα κρεμάσει.

ΤΕΛΕΣΤΕΣ ΕΚΧΩΡΗΣΗΣ, ΑΥΞΗΣΗΣ, ΜΕΙΩΣΗΣ (ASSIGNMENT,INCREMENT AND DECREMENT OPERATORS)

Στην πραγματικότητα, κανένας δεν γράφει βρόγχους for, όπως κάναμε εμείς στις προηγούμενες ενότητες. Στη θέση τους χρησιμοποιούν τους τελεστές προσαύξησης ή μείωσης. Δύο από αυτούς, το ++ και -- λειτουργούν όπως στο ακόλουθο παράδειγμα.

```
//Count to ten

class CountToTen  {

  public static void main (String args[]) {
    int i;
    for (i=1; i <=10; i++) {
      System.out.println(i);
    }
    System.out.println("All done!");

 }

}
//Count to ten??

class BuggyCountToTen  {

  public static void main (String args[]) {
    int i;
    for (i=1; i <=10; i--) {
      System.out.println(i);
```

```
    }
    System.out.println("All done!");

  }

}
```

Το i++ είναι συντομευμένη η έκφραση i=i+1. Αντίστοιχα, όταν γράφουμε i--, είναι σαν να γράφουμε με συντομία το i=i-1. Ο λόγος που προστέθηκαν αυτοί οι τελεστές στις γλώσσες προγραμματισμού είναι ότι πολύ συχνά προσθέτουμε ή αφαιρούμε τη μονάδα σ' ένα αριθμό. Έτσι ο κώδικας γίνεται πιο απλός.

Τι γίνεται όμως όταν θέλουμε ν' αυξήσουμε ένα μέγεθος όχι κατά 1 αλλά κατά 2, 3 ή 15; Θα μπορούσαμε φυσικά να γράψουμε i=i+15, αλλά αυτό συμβαίνει τόσο συχνά, που υπάρχει ένας άλλος τελεστής πρόσθεσης και εκχώρησης, το +=. Έτσι θα γράψουμε i+=15.Αν θέλαμε να μετρήσουμε από το 0 ως το 20 θα γράφαμε:

```
class CountToTwentyByTwos  {

  public static void main (String args[]) {
    int i;
    for (i=0; i <=20; i += 2) {
      System.out.println(i);
    }
    System.out.println("All done!");

 } //main ends here

}
```

Όπως πιθανόν να μαντέψατε υπάρχει και ο αντίστοιχος -= τελεστής. Αν θέλαμε να μετρήσουμε από το 20 ως το 0 ανά δύο θα γράφαμε :

```
class CountToZeroByTwos  {

  public static void main (String args[]) {
    int i;
    for (i=20; i >= 0; i -= 2) {
      System.out.println(i);
    }
    System.out.println("All done!");

  }

}
```

Σημειώστε ότι αν θέλαμε να μετρήσουμε από πάνω προς τα κάτω όπως στο προηγούμενο παράδειγμα, πρέπει ν' αλλάξουμε την αρχικοποίηση και τα συστατικά του ελέγχου στον βρόγχο for.

Υπάρχουν επίσης και οι τελεστές *= και /= που πολλαπλασιάζουν και διαιρούν από τη δεξιά πλευρά πριν την απόδοση. Παρακάτω έχουμε ένα παράδειγμα.

Πριν πολλά χρόνια, κάποιος άντρας στην Ινδία εφεύρε το παιχνίδι του σκακιού. Αυτό το παιχνίδι είχε ενθουσιάσει τόσο πολύ το Βασιλιά που ήταν διατεθιμένος να χαρίσει στον εφευρέτη του το μισό του βασίλειο και το χέρι της κόρης του.

Ο εφευρέτης που ήταν πολύ έξυπνος και καθόλου πλεονέκτης δεν ικανοποιήθηκε με το μισό βασίλειο και είπε στο Βασιλιά: « Βασιλιά μου, είμαι ένας ταπεινός άνθρωπος και δεν ξέρω τι να κάνω το μισό σου βασίλειο. Έλα να υπολογίσουμε το δώρο μου αλλιώς. Βάλε στο πρώτο τετράγωνο του σκακιού ένα σπόρο σιτάρι. Μετά, στο δεύτερο τετράγωνο δύο σπόρους σιτάρι, στο τρίτο τετράγωνο δύο φορές από δύο σπόρους κ.τ.λ μέχρι να καλυφθεί ολόκληρο το σκάκι από σπόρους.

Όταν το άκουσε αυτό ο Βασιλιάς χάρηκε πολύ που θα γλίτωνε με τόσο λίγα έξοδα και συμφώνησε αμέσως. Ζήτησε να του φέρουν μια σακούλα σιτάρι και όταν του την έφεραν άρχισε να το τοποθετεί. Σύντομα το σιτάρι τελείωσε και δεν είχε καλύψει ούτε το μισό σκάκι. Ζήτησε και δεύτερη και τρίτη και άλλες πολλές σακούλες, μέχρι που αναγκάστηκε να παραδεχτεί την ήττα του.

Πόσο σιτάρι χρειαζόταν ο βασιλιάς; Ας προσπαθήσουμε να υπολογίσουμε. Θυμηθείτε ότι το σκάκι έχει 64 τετράγωνα

```
class CountWheat  {

  public static void main (String args[]) {

    int i, j, k;

    j = 1;
    k = 0;

    for (i=1; i <= 64; i++) {
      k += j;
      System.out.println(k);
      j *= 2;
    }
    System.out.println("All done!");

  }

}
```

Μπορούμε να βελτιώσουμε ελαφρώς τα αποτελέσματα αλλάζοντας τους ints με longs κι έτσι έχουμε:

```
class CountWheat {

  public static void main (String args[]) {

    long i, j, k;

    j = 1;
    k = 0;
```

```
    for (i=1; i <= 64; i++) {
      k += j;
      System.out.println(k);
      j *= 2;
    }
    System.out.println("All done!");

  }

}
```

Ένας long είναι μια μεταβλητή ακέραιου τύπου που μπορεί να εκφράσει 9,223,372,036,854,775,807 αριθμούς. Ακόμα κι ένας long όμως δεν μπορεί να μετρήσει το σιτάρι που χρειαζόταν ο βασιλιάς. Ας προσπαθήσουμε να χρησιμοποιήσουμε τώρα έναν double.

```
class CountWheat {

  public static void main (String args[]) {

  int i;
  double j, k;

  j = 1.0;
  k = 0.0;

  for (i=1; i <= 64; i++) {
      k += j;
      System.out.println(k);
      j *= 2;
  }
  System.out.println("All done!");

 }

}
```

Ένας double μπορεί να κρατήσει έναν αριθμό μεγέθους 1.79769313486231570e+308. Έτσι είναι ικανός να μετρήσει το χρέος του Βασιλιά που ανέρχεται σε 1.84467e+019 σπόρους από σιτάρι.

ΑΣΚΗΣΕΙΣ

1. Είναι ικανός ένας float να μετρήσει πόσους σπόρους από σιτάρι χρωστάει ο βασιλιάς;
2. Γιατί δεν υπάρχει ένας τελεστής ** ή //;

ΜΕΘΟΔΟΙ (METHODS)

Όλα τα παραπάνω προγράμματα που έχουμε γράψει μέχρι τώρα είναι πολύ απλά κι έχουν λιγότερο από 50 γραμμές κώδικα το καθένα. Καθώς τα προγράμματα μεγαλώνουν σε έκταση είναι λογικό να τα χωρίζουμε σε τμήματα. Κάθε τμήμα μπορεί να πραγματοποιήσει έναν υπολογισμό και πιθανότατα να επιστρέψει μία τιμή. Αυτό είναι πολύ χρήσιμο, ειδικά όταν ο υπολογισμός θα επαναληφθεί σε διάφορα άλλα μέρη του προγράμματος. Επίσης βοηθάει στο να προσδιοριστεί με πιο σαφή τρόπο η ροή ενός προγράμματος, όπως η περίληψη προσδιορίζει τη ροή ενός βιβλίου.

Κάθε υπολογιστικό τμήμα ενός προγράμματος ονομάζεται μέθοδος. Οι μέθοδοι είναι αντίστοιχες με τις λειτουργίες (functions) της C, τις διαδικασίες και λειτουργίες (procedures and functions) της Pascal και τις λειτουργίες και υπορουτίνες (functions and subroutines) της Fortran.

Τα παραπάνω προγράμματα χρησιμοποίησαν έναν αριθμό μεθόδων, παρόλο που οι μέθοδοι αυτές παρέχονται από το σύστημα. Όταν γράψαμε System.out.println(«Hello World!»);. Στο πρώτο πρόγραμμα χρησιμοποιήσαμε τη μέθοδο System.out.println(). (Για να είμαστε πιο συγκεκριμένοι, χρησιμοποιήσαμε τη μέθοδο println() του τμήματος out στην κλάση System. Θα μιλήσουμε γι' αυτό στο κεφάλαιο 4). Η μέθοδος System.out.println() απαιτεί αρκετό κώδικα ο οποίος υπάρχει στις βιβλιοθήκες του συστήματος. Έτσι, αντί να συμπεριλαμβάνουμε τον κώδικα κάθε φορά που θέλουμε να τυπώσουμε, απλά καλούμε τη μέθοδο System.out.println().

Μπορείτε να γράψετε και να καλέσετε τις δικές σας μεθόδους. Ας δούμε ένα απλό παράδειγμα. Το παρακάτω είναι ένα απλό πρόγραμμα που ζητά έναν αριθμό από τον χρήστη και μετά υπολογίζει την παράγωγο του αριθμού.

Θα χρησιμοποιήσουμε δύο μεθόδους στο πρόγραμμα, μία που ελέγχει αν ο χρήστης εισήγαγε έναν έγκυρο, θετικό ακέραιο και μία άλλη που υπολογίζει την παράγωγο. Θα ξεκινήσουμε γράφοντας την κύρια μέθοδο του προγράμματος:

```
class Factorial {

  public static void main(String args[]) {

    int n;

    while ((n = getNextInteger()) >= 0)  {
      System.out.println(factorial(n));
    }

  } // main ends here

}
```

Εκτός των άλλων, ο κώδικας αυτός αποδεικνύει ότι οι μέθοδοι καθιστούν δυνατό να σχεδιαστεί η ροή του προγράμματος χωρίς περιττές λεπτομέρειες. Απλά ονομάσαμε δύο μεθόδους getNextInteger() και factorial(), χωρίς να ανησυχούμε για την εφαρμογή τους. Μπορούμε να προσθέσουμε τον υπόλοιπο κώδικα σε μικρότερα, ευκολονόητα τμήματα. Ας γράψουμε πρώτα τη μέθοδο factorial:

```
long factorial (long n) {

  int i;
  long result=1;

  for (i=1; i <= n; i++) {
   result *= i;
  }

  return result;

} // factorial ends here
```

Θα μπορούσαμε να συμπεριλάβουμε αυτό τον κώδικα στη μέθοδο main αλλά είναι πιο εύκολο να κατανοήσουμε τον αλγόριθμο σπάζοντας τον κώδικα σε μικρότερα τμήματα. Είναι επίσης πιο εύκολο να ελέγξουμε και να διορθώσουμε. Μπορούμε να γράψουμε ένα απλό πρόγραμμα που θα μας επιτρέπει να ελέγχουμε τη μέθοδο factorial προτού μας απασχολήσει το αν είναι αποδεκτή και έγκυρη η είσοδος του χρήστη. Ακολουθεί το πρόγραμμα ελέγχου:

```
class FactorialTest {

  public static void main(String args[]) {

    int n;
    int i;
    long result;

    for (i=1; i <=10; i++)  {
      result = factorial(i);
      System.out.println(result);
    }

  } // main ends here


  static long factorial (int n) {

  int i;
  long result=1;

  for (i=1; i <= n; i++) {
   result *= i;
  }

  return result;

} // factorial ends here


}
```

Οι προγραμματιστές της C θα πρέπει να προσέξουν ότι και οι δύο μέθοδοι προσδιορίζονται μέσα σε ορισμό κλάσης (class definition). Παρατηρούμε για άλλη μια φορά ότι τα πάντα στη Java ανήκουν σε κλάσεις.

Ας δούμε με μια πιο προσεκτική ματιά τη σύνταξη της μεθόδου:

```
 static long factorial (int n) {

  int i;
  long result=1;

  for (i=1; i <= n; i++) {
   result *= i;
  }

  return result;

}
```

Οι μέθοδοι ξεκινούν με μία δήλωση. Αυτό μπορεί να περιλαμβάνει από 3 έως 5 τμήματα. Πρώτα είναι ένας προαιρετικός καθοριστής πρόσβασης (optional access specifier), που μπορεί να είναι γενικός (public), ιδιωτικός (private) ή προστατευόμενος (protected). Μία γενική μέθοδο μπορούμε να την καλέσουμε από παντού. Μία ιδιωτική μέθοδος μπορεί να χρησιμοποιηθεί μόνο από την κλάση που προσδιορίζεται. Μία προστατευμένη μέθοδος μπορεί να χρησιμοποιηθεί οπουδήποτε εντός του πακέτου που προσδιορίζεται. Οι μέθοδοι που δεν έχουν δηλωθεί σαν γενικές ή ιδιωτικές θεωρούνται προστατευόμενες.Στη συνέχεια αποφασίζουμε αν μια μέθοδος είναι στατική ή όχι. Οι στατικές μέθοδοι έχουν ένα στιγμιότυπο ανά κλάση παρά ένα παράδειγμα ανά αντικείμενο. Όλα τα αντικείμενα μιας κλάσης μοιράζονται ένα αντίγραφο της στατικής μεθόδου. Εξ' ορισμού οι μέθοδοι δεν είναι στατικές.

Στη συνέχεια εξειδικεύουμε τον τύπο επιστροφής. Αυτό είναι η τιμή που θα επιστραφεί στην καλούμενη μέθοδο όταν τελειώσουν όλοι οι υπολογισμοί μέσα στη μέθοδο. Για παράδειγμα, αν ο τύπος επιστροφής είναι int, μπορούμε να χρησιμοποιήσουμε τη μέθοδο όπου χρησιμοποιούμε έναν σταθερό ακέραιο. Αν ο τύπος επιστροφής είναι void, τότε δεν επιστρέφεται καμία τιμή.

Έπειτα ακολουθεί το όνομα της μεθόδου.

Στη συνέχεια έχουμε τις παρενθέσεις. Μέσα στις παρενθέσεις δίνουμε ονόματα και τύπους στα ορίσματα της μεθόδου. Μία μέθοδος μπορεί να έχει κανένα, ένα ή πολλά ορίσματα. Τα ορίσματα αυτά μπορούν να χρησιμοποιηθούν μέσα στη μέθοδο όπως οι τοπικές μεταβλητές.

Τέλος το υπόλοιπο της μεθόδου περικλείεται μέσα σε άγκιστρα τα οποία το κάνουν ένα απλό μπλοκ. Το τμήμα μέσα στα άγκιστρα είναι όπως οι κύριες μέθοδοι. Υπάρχουν δηλώσεις μεταβλητών, κώδικας και τέλος κάτι νέο, μία δήλωση επιστροφής, η οποία στέλνει μια τιμή πίσω στην καλούμενη μέθοδο. Ο τύπος της τιμής αυτής πρέπει να συμφωνεί με τον τύπο που δηλώθηκε στη μέθοδο.

Γενικά κάθε γραμμή που μοιάζει με το Text(arg1, arg 2) ή με το text(arg 1) ή text() είναι ένα κάλεσμα μεθόδου. Ο compiler είναι υπεύθυνος για να αναγνωρίσει μεταξύ των παρενθέσεων, ποιες σημαίνουν κάλεσμα μεθόδου και ποιες χρησιμοποιούνται σε μαθηματικές εκφράσεις, όπως (3+7)*2. Ο compiler κάνει καλή δουλειά σ' αυτό και μπορείτε με ασφάλεια να συμπεράνετε ότι ό,τι δεν είναι αριθμητική έκφραση είναι κάλεσμα μεθόδου.

Οι μέθοδοι χωρίζουν ένα πρόγραμμα σε αλγόριθμους και υπολογισμούς. Σε μεγάλα προγράμματα είναι απαραίτητο να χωρίζουμε τα δεδομένα.

ΑΝΑΔΡΟΜΙΚΕΣ ΜΕΘΟΔΟΙ (RECURSIVE METHODS)

Η Java υποστηρίζει τις αναδρομικές μεθόδους. Για παράδειγμα, αν βρίσκεστε ήδη στη μέθοδοΑ() μπορείτε να καλέσετε τη μέθοδοΑ(). Ο πιο απλός τρόπος για να εξηγήσουμε την αναδρομή είναι να δούμε το απλό ακρονύμιο GNU. Το GNU, εκτός των άλλων, προσπαθεί να παράγει ελεύθερες εκδόσεις του UNIX και πολλά εργαλεία στο UNIX. Ένα βασικό πρόβλημα είναι ότι το όνομα UNIX είναι σήμα κατατεθέν κι έτσι το GNU δε μπορεί να το χρησιμοποιήσει. Έτσι, αντί για το UNIX έχουμε το GNU, όπου GNU σημαίνει «Gnu's Not Unix». Τι είναι GNU; Ένα επίπεδο είναι «(Gnu's Not Unix)'s Not Unix». Ένα άλλο είναι «((Gnu's Not Unix)'s Not Unix)'s Not Unix». Είναι σα να καθόμαστε μπροστά σε καθρέφτες. Στους υπολογιστές η αναδρομή κατορθώνεται με το να επιτρέπουμε σε μια μέθοδο να καλεί τον εαυτό της.

Πιθανόν να εντοπίσατε ένα πρόβλημα. Πού σταματούν όλα αυτά; Όταν γράφετε αναδρομικές μεθόδους πρέπει να είστε προσεκτικοί και να συμπεριλαμβάνετε συνθήκες τερματισμού. Παρόλο που η Java δεν θέτει όρια για το βάθος μιας επανάληψης είναι πιθανόν κάτι τέτοιο να χρησιμοποιήσει όλη τη μνήμη του υπολογιστή σας.

Ας δούμε ένα παράδειγμα. n! (που το ονομάζουμε «n παραγοντικό») προσδιορίζεται σαν n φορές (n-1) φορές (n-2) φορές ....*2 φορές*1, όπου το n είναι ένας θετικός ακέραιος. 0! προσδιορίζεται σαν 1. Όπως θα δούμε n!=n φορές (n-1)!. Αυτό οδηγεί σ' έναν αναδρομικό υπολογισμό, όπως στην παρακάτω μέθοδο:

```
public static long factorial (int n) {

  if (n == 0) {
    return 1;
  }
  else {
    return n*factorial(n-1);
  }

}
```

Τι γίνεται όμως όταν ένας αρνητικός ακέραιος μπει στη μέθοδο factorial; Για παράδειγμα, ας υποθέσουμε ότι ζητάμε το παραγοντικό (-1). Τότε έχουμε σαν αποτέλεσμα το -1*-2*-3*-4*... Αν είστε τυχεροί, το

πρόγραμμα μπορεί απροσδόκητα να φτάσει στους θετικούς αριθμούς και να μετρήσει μετά το 0. Αν όχι, τότε το πρόγραμμα πέφτει πάνω σε ένα StackOutOfMemoryError. Οι συνθήκες τερματισμού είναι πολύ σημαντικές. Σ' αυτή την περίπτωση θα πρέπει να ελέγξετε αν περάσατε έναν αρνητικό ακέραιο. Αν ναι, επιστρέφει άπειρο. (Η factorial είναι μια ειδική περίπτωση της λειτουργίας gamma για τους μη αρνητικούς ακέραιους. Παρόλο που η συνάρτηση factorial ορίζεται μόνο για τους θετικούς ακέραιους, η gamma ορίζεται για όλους τους πραγματικούς αριθμούς. Είναι δυνατόν να δείξουμε ότι η gamma είναι άπειρη για τους αρνητικούς ακέραιους). Η Java δεν υποστηρίζει τις άπειρες τιμές κι έτσι επιστρέφει την προειδοποιητική τιμή -1. Ακολουθεί μια καλύτερη αναδρομική παραγοντική μέθοδος:

```
public static long factorial (int n) {

  if (n < 0) {
    return -1;
  }
  else if (n == 0) {
    return 1;
  }
  else {
    return n*factorial(n-1);
  }

}
```

Μπορεί να αποδειχθεί μαθηματικά ότι όλοι οι αναδρομικοί αλγόριθμοι έχουν μη αναδρομικά αντίτυπα. Για παράδειγμα η αναδρομική μέθοδος μπορεί να γραφτεί μη αναδρομικά όπως παρακάτω:

```
public static long factorial (int n) {

  long result = 1;

  for (int i = 1; i <= n; i++) {
    result *= i;
  }

  return result;

}
```

Η μη αναδρομική μέθοδος στο πρόβλημα είναι ολοφάνερη αλλά υπάρχουν και περιπτώσεις που δεν είναι.

Ακολουθεί ένα παράδειγμα ενός αναδρομικού προγράμματος για το οποίο δεν μπορέσαμε να βρούμε μία απλή μη αναδρομική αντίστοιχη μέθοδο. Είναι ένα επαναληπτικό πρόγραμμα που υπολογίζει και τυπώνει όλα τα πιθανά RAM configurations:

```
import java.util.Hashtable;
import java.util.Enumeration;
```

```
public class RamConfig {

   static int[] sizes = {0, 8, 16, 32, 64};
   static Hashtable configs = new Hashtable(64);
   static int slots[] = new int[4];

   public static void main (String args[]) {

     System.out.println("Available DIMM sizes are: ");
     for (int i=0; i < sizes.length; i++) System.out.println(sizes[i]);

     fillSlots(slots.length - 1);
     System.out.println("Ram configs are: ");

     for (Enumeration e = configs.elements(); e.hasMoreElements(); ) {
       System.out.println(e.nextElement());
     }

   }


  private static void fillSlots(int n) {

    int total;

    for (int i=0; i < sizes.length; i++) {
       slots[n] = sizes[i];
       if (n == 0) {
          total = 0;
          for (int j = 0; j < slots.length; j++) {
            total += slots[j];
          }
          configs.put(Integer.toString(total), new Integer(total));
       }
       else {
         fillSlots(n - 1);
       }
    }

  }

}
```

Η αναδρομή δοκιμάζει την ταχύτητα με την οποία μια γλώσσα καλεί τις μεθόδους της. Αυτό είναι σημαντικό γιατί οι μοντέρνες εφαρμογές έχουν μια τάση να καταναλώνουν το χρόνο τους κάνοντας διάφορες API λειτουργίες. Η PCWeek χρησιμοποιεί το ονομαζόμενο Tak που πραγματοποιεί 63,609 καλέσματα επαναληπτικών μεθόδων ανά πέρασμα. Ο αλγόριθμος είναι απλός. Αν το y είναι μεγαλύτερο ή ίσο με το x τότε Tak(x,y,z) είναι το z. Αυτή είναι η μη αναδρομική συνθήκη τερματισμού. Διαφορετικά αν το y είναι μικρότερο από το x, το Tak(x,y,z) είναι το Tak(Tak(x-1, y, z), Tak(y-1, z, x), Tak(z-1, x, y)). Το Tak υπολογίζει το Tak(18,12,6) μεταξύ 100 και 10.000 φορές και δίνει τον αριθμό των περασμάτων ανά δευτερόλεπτο. Για περισσότερες πληροφορίες για το Tak δείτε το άρθρο του Peter Coffee «Tak test stands the test of time» στη σελ 91 στο PCWeek της 9/30/1996.

Παρακάτω ακολουθεί μια παραλλαγή. Υπάρχουν εκδόσεις πολυμορφικών ακεραίων και κινητής υποδιαστολής στη μέθοδο Tak. Εξ ορισμού χρησιμοποιείται ακέραιος. Αν η σημαία f δίνεται στο command line, χρησιμοποιείται η μέθοδος της κινητής υποδιαστολής. Στο command line μπορεί να εισάγεται και ο αριθμός των περασμάτων. Αν όχι, τότε πραγματοποιούνται 1000 περάσματα. Ακολουθεί η κλάση Date της Java:

```
import java.util.Date;


public class Tak {

  public static void main(String[] args) {

   boolean useFloat = false;
   int numpasses;

   for (int i = 0; i < args.length; i++) {
     if (args[i].startsWith("-f")) useFloat = true;
   }
   try {
     numpasses = Integer.parseInt(args[args.length-1]);
   }
   catch (Exception e) {
     numpasses = 1000;
   }

   Date d1 = new Date();
   for (int i = 0; i < numpasses; i++) {
      Tak(18.0f, 12.0f, 6.0f);
   }
   Date d2 = new Date();
   long TimeRequired = d2.getTime() - d1.getTime();
   double numseconds = TimeRequired/1000.0;

   System.out.println("Completed " + numpasses + " passes in "
    + numseconds + " seconds" );
   System.out.println(numpasses/numseconds + " calls per second");

  }

  public static int Tak (int x, int y, int z) {
    if (y >= x) return z;
    else return Tak(Tak(x-1, y, z), Tak(y-1, z, x), Tak(z-1, x, y));

  }

  public static float Tak (float x, float y, float z) {
    if (y >= x) return z;
    else return Tak(Tak(x-1.0f, y, z), Tak(y-1.0f, z, x), Tak(z-1.0f,
x, y));

  }

}
```

ΑΣΚΗΣΗ

1. Βρείτε τον μη επαναληπτικό ισότιμο αλγόριθμο Ram Config.

ΠΙΝΑΚΕΣ (ARRAYS)

Στα σημαντικά προβλήματα που αφορούν τους υπολογιστές συχνά χρειάζεται να αποθηκεύουμε λίστες αντικειμένων. Συχνά αυτά τα αντικείμενα καθορίζονται σειριακά και αναφέρονται στη θέση τους στη λίστα. Πολλές φορές η σειρά αυτή είναι φυσική, όπως η σειρά των δέκα πρώτων ανθρώπων που φτάνουν στην τράπεζα. Το πρώτο άτομο θα είναι το αντικείμενο 1 στη λίστα, το δεύτερο άτομο που έφτασε θα είναι το αντικείμενο 2 στη λίστα κτλ. Σε άλλες περιπτώσεις η σειρά αυτή δεν έχει καμία σημασία, όπως στο RAM configuration problem όπου το να έχουμε 4 MB SIMM στη slot A και 8 MB SIMM στη slot B είναι ακριβώς το ίδιο με το να έχουμε 8 MB SIMM στη slot A και 4 MB SIMM στην slot B.

Υπάρχουν πολλοί τρόποι για να αποθηκεύσουμε λίστες, όπως συνδεδεμένες λίστες, σύνολα, δέντρα και πίνακες. Ποιο θα προτιμήσετε εξαρτάται από τις απαιτήσεις της εφαρμογής σας και τη φύση των δεδομένων σας. Η Java παρέχει κλάσεις για πολλούς απ' αυτούς τους τρόπους αποθήκευσης δεδομένων.

Οι πίνακες (Arrays) είναι πιθανότατα ο αρχαιότερος και πιο αποτελεσματικός τρόπος αποθήκευσης συνόλων μεταβλητών. Ένας πίνακας είναι ένα σύνολο μεταβλητών που μοιράζονται το ίδιο όνομα και συντάσσονται με τη σειρά από το 0 μέχρι το πλήθος των μεταβλητών μειωμένο κατά 1. Ο αριθμός των μεταβλητών που μπορεί να αποθηκευτεί ονομάζεται διάσταση του πίνακα. Κάθε μεταβλητή ονομάζεται στοιχείο του πίνακα.

Ένας πίνακας μπορεί να παρασταθεί σαν στήλη δεδομένων όπως πιο κάτω:

```
I[0]




4
I[1]    2
I[2]    76
I[3]    -90
I[4]
6
```

Σ' αυτή την περίπτωση βλέπουμε έναν ακέραιο πίνακα που ονομάζεται I με 5 στοιχεία. Ο τύπος του πίνακα είναι int και έχει διάσταση 5.

ΔΗΜΙΟΥΡΓΙΑ ΠΙΝΑΚΩΝ

Υπάρχουν τρία στάδια για τη δημιουργία πινάκων:

1. **Να δηλώσουμε τον πίνακα**
2. **Να ορίσουμε τον πίνακα**
3. **Να αρχικοποιήσουμε τον πίνακα**

Δήλωση Πινάκων. Όπως και οι άλλες μεταβλητές στη Java, ένας πίνακας πρέπει να έχει ένα συγκεκριμένο τύπο όπως byte, int, string ή double. Μόνο μεταβλητές του αντίστοιχου τύπου μπορούν να αποθηκευτούν σε έναν πίνακα. Για παράδειγμα δεν μπορούμε να έχουμε έναν πίνακα που να αποθηκεύει και ints και strings.

Όπως και οι άλλες μεταβλητές στη Java, έτσι και οι πίνακες πρέπει να δηλώνονται. Όταν δηλώνουμε έναν πίνακα βάζουμε στον τύπο και το [] που δείχνει ότι η μεταβλητή είναι ένας πίνακας. Ακολουθούν μερικά παραδείγματα:

```
int[] k;

float[] yt;

String[] names;
```

Με άλλα λόγια δηλώνουμε έναν πίνακα όπως δηλώνουμε οποιαδήποτε άλλη μεταβλητή μόνο που βάζουμε και αγκύλες στο τέλος του τύπου της μεταβλητής.

Ορισμός Πινάκων. Για να δημιουργήσουμε έναν πίνακα χρησιμοποιούμε τον τελεστή new. Πρέπει να πούμε στον compiler πόσα στοιχεία θα αποθηκευτούν στον πίνακα. Πιο κάτω βλέπουμε πώς δημιουργούμε τις μεταβλητές:

```
k = new int[3];

yt = new float[7];

names = new String[50];
```

Τα νούμερα στις αγκύλες καθορίζουν τη διάσταση του πίνακα, δηλαδή πόσες θέσεις έχει. Ο k μπορεί να κρατήσει τρία ints, ο yt μπορεί να κρατήσει επτά floats και ο names μπορεί να κρατήσει πενήντα strings. Συνήθως αυτό ονομάζεται ορισμός του πίνακα αφού καθορίζει το τμήμα της RAM που χρειάζεται ο πίνακας.

Εδώ θα δούμε για πρώτη φορά και τον τελεστή new. Ο new είναι δεσμευμένη λέξη στη Java και χρησιμοποιείται όχι μόνο για να ορίσει πίνακες αλλά και άλλα είδη αντικειμένων.

Αρχικοποίηση Πινάκων. Κάθε στοιχείο του πίνακα προσδιορίζεται από το όνομα του πίνακα και από έναν ακέραιο που φανερώνει τη θέση του στον πίνακα. Οι αριθμοί που

χρησιμοποιούνται ονομάζονται δείκτες του πίνακα. Οι δείκτες είναι συνεχόμενοι αριθμοί που ξεκινούν από το 0. Γι' αυτό και ο πίνακας k έχει τα στοιχεία k[0], k[1] και k[2]. Εφόσον ξεκινάμε από το 0 δεν υπάρχει k[3] και αν προσπαθήσουμε να έχουμε πρόσβαση σ' αυτό θα δημιουργήσουμε ένα ArrayIndexOutOfBoundsException.

Παρακάτω θα δούμε πώς αποθηκεύουμε τιμές στους πίνακες:

```
k[0] = 2;

k[1] = 5;

k[2] = -2;

yt[6] = 7.5f;

names[4] = "Fred";
```

Αυτό το βήμα ονομάζεται αρχικοποίηση του πίνακα ή καλύτερα αρχικοποίηση των στοιχείων του πίνακα.

Ακόμα και για τους πίνακες μεσαίου μεγέθους είναι σπάνιο να αρχικοποιούμε κάθε στοιχείο ξεχωριστά. Συχνά είναι πιο εύκολο να χρησιμοποιούμε τον βρόγχο for. Για παράδειγμα ακολουθεί ένα loop που γεμίζει έναν πίνακα με τα τετράγωνα των αριθμών από το 0 ως το 100.

```
float[] squares = new float[101];

for (int i=0, i <= 100; i++) {

squares[i] = i*i;

}
```

Θα πρέπει να προσέξετε δύο σημεία του κώδικα:

- Εφόσον οι δείκτες ξεκινούν από το 0, χρειαζόμαστε 101 στοιχεία αν θέλουμε να συμπεριλάβουμε και το τετράγωνο του 100.
- Παρόλο που το i είναι int, γίνεται float όταν αποθηκεύει τετράγωνο αφού δηλώσαμε τα τετράγωνα σαν πίνακα float.

Ο τρόπος για να αποφύγουμε το πρώτο σημείο είναι το παράδειγμα που ακολουθεί:

```
float[] squares = new float[101];

for (int i=0, i < squares.length; i++) {

squares[i] = i*i;
```

```
}
```

Σημειώστε ότι το <= γίνεται <.

Συντομεύσεις (Shortcuts)

Πιθανόν να φαίνεται δύσκολη η δημιουργία ενός πίνακα, ειδικά σε κάποιον που έχει μάθει να χειρίζεται γλώσσες όπως η Fortran. Ευτυχώς η Java παρέχει πολλές ευκολίες για τη δήλωση, τον ορισμό και την αρχικοποίηση πινάκων.

Μπορούμε να δηλώσουμε και να ορίσουμε έναν πίνακα την ίδια στιγμή:

```
int[] k = new int[3];

float[] yt = new float[7];

String[] names = new String[50];
```

Μπορούμε ακόμα να δηλώσουμε, να ορίσουμε και να αρχικοποιήσουμε έναν πίνακα την ίδια στιγμή δημιουργώντας μια λίστα με τις αρχικές τιμές μέσα σε αγκύλες όπως πιο κάτω:

```
int[] k = {1, 2, 3};

float[] yt = {0.0f, 1.2f, 3.4f, -9.87f, 65.4f, 0.0f, 567.9f};
```

ΜΕΤΡΗΣΗ ΨΗΦΙΩΝ (COUNTING DIGITS)

Οι κύριες μέθοδοι μιας εφαρμογής αποθηκεύουν τα command line arguments σ'έναν πίνακα από αλφαριθμητικά που ονομάζεται args.

Ας θεωρήσουμε μια κλάση που μετρά πόσες φορές επαναλαμβάνονται τα ψηφία 0-9 στο δεκαδικό μέρος του αριθμού pi για παράδειγμα. Τα ψηφία θα πρέπει να επαναλαμβάνονται με την ίδια συχνότητα μετά από ένα μεγάλο χρονικό διάστημα.

Αυτό θα το δούμε δημιουργώντας έναν πίνακα από 10 longs που ονομάζεται ndigit. Το μηδενικό στοιχείο του ndigit θα ανιχνεύει τον αριθμό των μηδενικών, το πρώτο στοιχείο του ndigit θα ανιχνεύει τον αριθμό των άσσων και ούτω καθεξής. Ελέγχουμε τον random number generator της Java για να δούμε αν παράγει τυχαίους αριθμούς.

```
import java.util.*;

class RandomTest {
```

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

    int[] ndigits = new int[10];
    double x;
    int n;

    Random myRandom = new Random();

    // Initialize the array
    for (int i = 0; i < 10; i++) {
      ndigits[i] = 0;
    }

    // Test the random number generator a whole lot
    for (long i=0; i < 100000; i++) {
      // generate a new random number between 0 and 9
      x = myRandom.nextDouble() * 10.0;
      n = (int) x;
      //count the digits in the random number
      ndigits[n]++;
    }

    // Print the results
    for (int i = 0; i <= 9; i++) {
      System.out.println(i+": " + ndigits[i]);
    }
  }

}
```

Έχουμε τρεις βρόγχους for στον παρακάτω κώδικα, έναν για να αρχικοποιήσουμε τον πίνακα, έναν για να πραγματοποιήσουμε τον επιθυμητό υπολογισμό και έναν τρίτο για να τυπώνει τα αποτελέσματα. Αυτό είναι αρκετά συνηθισμένο σε κώδικες που χρησιμοποιούν πίνακες.

ΔΙΔΙΑΣΤΑΤΟΙ ΠΙΝΑΚΕΣ (TWO DIMENSIONAL ARRAYS)

Ο πιο συνηθισμένος τύπος πολυδιάστατου πίνακα είναι ο διδιάστατος. Ένας διδιάστατος πίνακας σαν πίνακας τιμών είναι όπως ο παρακάτω:

c0	c1	c2	c3	
r0	0	1	2	3
r1	1	2	3	4
r2	2	3	4	5
r3	3	4	5	6
r4	4	5	6	7

Εδώ έχουμε έναν πίνακα με 5 γραμμές και 4 στήλες. Έχει συνολικά 20 στοιχεία. Έχουμε πει ότι έχει διαστάσεις 5x4 , όχι 20. Αυτός ο πίνακας δεν είναι ίδιος με τον 4x5 πίνακα που ακολουθεί πιο κάτω:

c0	c1	c2	c3	c4	
r0	0	1	2	3	4
r1	1	2	3	4	5
r2	2	3	4	5	6
r3	3	4	5	6	7

Πρέπει να χρησιμοποιούμε 2 αριθμούς για να αναγνωρίσουμε τη θέση σε ένα διδιάστατο πίνακα. Αυτές είναι οι θέσεις των γραμμών και των στηλών που βρίσκονται τα στοιχεία. Για παράδειγμα αν ο παραπάνω πίνακας λέγεται J τότε J[0][0] είναι 0, J[0][1] είναι 1, J[0][2] είναι 2, J[0][3] είναι 3, J[1][0] είναι 1 κτλ.

Εδώ βλέπουμε πώς αναφέρονται τα στοιχεία ενός 4x5 πίνακα που ονομάζεται M.

M[0][0]

M[0][1]	M[0][2]	M[0][3]	M[0][4]	
M[1][0]	M[1][1]	M[1][2]	M[1][3]	M[1][4]
M[2][0]	M[2][1]	M[2][2]	M[2][3]	M[2][4]
M[3][0]	M[3][1]	M[3][2]	M[3][3]	M[3][4]

Οι διδιάστατοι πίνακες δηλώνονται, ορίζονται και αρχικοποιούνται όπως και οι μονοδιάστατοι πίνακες. Γι' αυτό πρέπει να καθορίζουμε δύο διαστάσεις αντί για μία και γι' αυτό χρησιμοποιούμε δύο φωλιασμένα for για να γεμίσουμε τον πίνακα.

Στα παραπάνω παραδείγματα οι πίνακες γεμίζονται με το άθροισμα των δεικτών των γραμμών και των στηλών. Βλέπετε τον κώδικα που δημιουργεί και γεμίζει έναν τέτοιο πίνακα:

```
class FillArray {

  public static void main (String args[]) {

    int[][] M;
    M = new int[4][5];

    for (int row=0; row < 4; row++) {
      for (int col=0; col < 5; col++) {
        M[row][col] = row+col;
```

```
      }
    }

  }

}
```

Ο αλγόριθμος που θα χρησιμοποιήσετε για να γεμίσετε έναν πίνακα εξαρτάται από τη χρήση για την οποία θέλετε τον πίνακα. Ακολουθεί ένα πρόγραμμα που υπολογίζει την ταυτοτική μήτρα για μια δεδομένη διάσταση.

```
class IDMatrix {

  public static void main (String args[]) {

    double[][] ID;
    ID = new double[4][4];

    for (int row=0; row < 4; row++) {
      for (int col=0; col < 4; col++) {
        if (row != col) {
          ID[row][col]=0.0;
        }
        else {
          ID[row][col] = 1.0;
        }
      }
    }

  }

}
```

Στους διδιάστατους πίνακες το λάθος ArrayIndexOutOfBounds συμβαίνει όταν υπερβαίνουμε το μέγιστο δείκτη γραμμής ή στήλης.

ΑΣΚΗΣΗ

1. Γράψτε ένα πρόγραμμα που να παράγει την HTML για τους πιο πάνω πίνακες.

ΠΟΛΥΔΙΑΣΤΑΤΟΙ ΠΙΝΑΚΕΣ (MULTIDIMENSIONAL ARRAYS)

Η Java μας επιτρέπει να έχουμε πίνακες τριών, τεσσάρων ή και περισσότερων διαστάσεων. Είναι πολύ πιθανό αν χρειάζεστε έναν πίνακα με περισσότερες από 3 διαστάσεις να χρησιμοποιήσετε λανθασμένη δομή δεδομένων. Ακόμα και οι τρισδιάστατοι πίνακες είναι σπάνιοι έξω από τις επιστημονικές και μηχανικές εφαρμογές.

Η σύνταξη για τους τρισδιάστατους πίνακες είναι μια επέκταση αυτής που χρησιμοποιήσαμε στους διδιάστατους. Ακολουθεί ένα πρόγραμμα που δηλώνει, ορίζει και αρχικοποιεί έναν τρισδιάστατο πίνακα:

```
class Fill3DArray {

  public static void main (String args[]) {

    int[][][] M;
    M = new int[4][5][3];

    for (int row=0; row < 4; row++) {
      for (int col=0; col < 5; col++) {
        for (int ver=0; ver < 3; ver++) {
          M[row][col][ver] = row+col+ver;
        }
      }
    }

  }

}
```

Χρειαζόμαστε τρία φωλιασμένα for για να χειριστούμε την επιπλέον διάσταση. Η σύνταξη για ακόμα μεγαλύτερες διαστάσεις είναι παρόμοια. Απλά προσθέτουμε άλλο ένα ζευγάρι αγκίστρων και μια ακόμα διάσταση.

ΜΗ ΙΣΟΖΥΓΙΣΜΕΝΟΙ ΠΙΝΑΚΕΣ (UNBALANCED ARRAYS)

Όπως και στη C, έτσι και στη Java δεν έχουμε πραγματικούς πολυδιάστατους πίνακες. Στη θέση τους η Java χρησιμοποιεί πίνακες πινάκων. Αυτό σημαίνει ότι είναι πολύ πιθανό να έχουμε μη ισοζυγισμένους πίνακες. Ένας μη ισοζυγισμένος πίνακας είναι ένας πολυδιάστατος πίνακας που οι διαστάσεις δεν είναι ίδιες για όλες τις γραμμές. Στις περισσότερες εφαρμογές αυτό πρέπει να το αποφεύγουμε.

ΑΝΑΖΗΤΗΣΗ (SEARCHING)

Είναι πολύ συνηθισμένο να ψάχνουμε σ' έναν πίνακα μια συγκεκριμένη τιμή. Μερικές φορές αυτή η τιμή είναι γνωστή εκ των προτέρων. Άλλες φορές αναζητάμε το μικρότερο ή το μεγαλύτερο στοιχείο.

Εκτός από την περίπτωση που ξέρουμε κάτι για το περιεχόμενο ενός πίνακα, σε όλες τις άλλες περιπτώσεις ο πιο γρήγορος αλγόριθμος αναζήτησης ενός πίνακα είναι η γραμμική αναζήτηση. Χρησιμοποιούμε έναν βρόγχο for και βλέπουμε κάθε στοιχείο του πίνακα μέχρι να βρούμε το στοιχείο που θέλουμε. Ακολουθεί μια απλή μέθοδος που τυπώνει το μεγαλύτερο και το μικρότερο στοιχείο ενός πίνακα.

```
   static void printLargestAndSmallestElements (int[] n) {

    int max = n[0];
    int min = n[0];
```

```
    for (int i=1; i < n.length; i++) {
      if (max < n[i]) {
        max = n[i];
      }
      if (min > n[i]) {
        min = n[i];
      }
    }

    System.out.println("Maximum: " + max);
    System.out.println("Minimum: " + min);

    return;

  }
```

Αν πρόκειται να ψάξετε έναν πίνακα πολλές φορές, πιθανόν να θελήσετε να τον ταξινομήσετε πρώτα. Θα δούμε τους αλγόριθμους ταξινόμησης στο επόμενο κεφάλαιο.

ΤΑΞΙΝΟΜΗΣΗ (SORTING)

Όλοι οι αλγόριθμοι ταξινόμησης βασίζονται σε δύο θεμελιώδεις λειτουργίες: τη σύγκριση και τη μετακίνηση (swapping). Η σύγκριση είναι απλή. Το swapping είναι λίγο πιο πολύπλοκο. Θεωρείστε το ακόλουθο πρόβλημα. Θέλουμε να μεταφέρουμε την τιμή από το a στο b. Οι πιο πολλοί θα πρότειναν μια λύση σαν την ακόλουθη:

```
class Swap1 {

  public static void main(String args[]) {

   int a = 1;
   int b = 2;

   System.out.println("a = "+a);
   System.out.println("b = "+b);

   // swap a and b

   a = b;
   b = a;

   System.out.println("a = "+a);
   System.out.println("b = "+b);

  }

}
```

Αυτή θα δημιουργούσε την παρακάτω έξοδο:

a = 1

b = 2

a = 2

b = 2

Δεν περιμέναμε όμως αυτό! Το πρόβλημα είναι ότι χάσαμε την τιμή 1 καθώς βάλαμε την τιμή του b στο a. Για να διορθωθεί αυτό πρέπει να εισάγουμε μια τρίτη μεταβλητή, την temp, η οποία θα κρατάει την αρχική τιμή του a.

```
class Swap2 {

  public static void main(String args[]) {

   int a = 1;
   int b = 2;
   int temp;

   System.out.println("a = "+a);
   System.out.println("b = "+b);

   // swap a and b

   temp = a;
   a = b;
   b = temp;

   System.out.println("a = "+a);
   System.out.println("b = "+b);

  }

}
```

Αυτός ο κώδικας παράγει το αποτέλεσμα που περιμέναμε.

a = 1

b = 2

a = 2

b = 1

Bubble Sort

Τώρα που είδαμε πώς να μετακινούμε τις τιμές 2 μεταβλητών, ας προχωρήσουμε στην ταξινόμηση. Υπάρχουν πολλοί διαφορετικοί αλγόριθμοι ταξινόμησης. Ένας από τους πιο απλούς και δημοφιλής αλγόριθμους είναι ο bubble sort. Η έννοια του bubble sort είναι να ξεκινήσουμε από την κορυφή του πίνακα. Συγκρίνουμε κάθε στοιχείο με το επόμενο

στοιχείο. Αν είναι μεγαλύτερο από αυτό τότε τα μεταθέτουμε. Αυτή τη διαδικασία την κάνουμε όσες φορές χρειαστεί. Η μικρότερη τιμή εμφανίζεται στην κορυφή του πίνακα ενώ η μεγαλύτερη στο τέλος. Ακολουθεί ο κώδικας:

```
import java.util.*;

class BubbleSort {

  public static void main(String args[]) {

   int[] n;
   n = new int[10];
   Random myRand = new Random();

   // initialize the array
   for (int i = 0; i < 10; i++) {
     n[i] = myRand.nextInt();
   }

   // print the array's initial order
   System.out.println("Before sorting:");
   for (int i = 0; i < 10; i++) {
     System.out.println("n["+i+"] = " + n[i]);
   }

   boolean sorted = false;
   // sort the array
   while (!sorted) {
     sorted = true;
     for (int i=0; i < 9; i++) {
       if (n[i] > n[i+1]) {
         int temp = n[i];
         n[i] = n[i+1];
         n[i+1] = temp;
         sorted = false;
       }
     }
   }

  // print the sorted array
  System.out.println();
  System.out.println("After sorting:");
   for (int i = 0; i < 10; i++) {
     System.out.println("n["+i+"] = " + n[i]);
   }


  }

}
```

Σ’ αυτήν την περίπτωση ταξινομήσαμε έναν πίνακα με αύξουσα σειρά. Το μικρότερο στοιχείο μπαίνει πρώτο. Θα ήταν εύκολο να το αλλάξουμε σε ταξινόμηση με φθίνουσα σειρά.

ΕΞΑΙΡΕΣΕΙΣ (CATCHING EXCEPTIONS)

Θυμάστε τον κώδικα:

```
// This is the Hello program in Java
class Hello {

    public static void main (String args[]) {

      /* Now let's say hello */
      System.out.print("Hello ");
      System.out.println(args[0]);
  }

}
```

Θυμάστε τί γινόταν όταν τρέχαμε το πρόγραμμα χωρίς να δίνουμε command line arguments; Το σύστημα εκτέλεσης (runtime system) δημιουργούσε μία εξαίρεση.

Exception in thread "main" java.lang.ArrayIndexOutOfBounds exception at Hello.mainC:\javahtml\Hello.java:7)

Αυτό που συνέβαινε ήταν ότι από τη στιγμή που δεν δίναμε στο Hello κανένα command line argument, δεν υπήρχε τίποτα στο args[0].Γι' αυτό και η Java έβγαζε το όχι και τόσο φιλικό μήνυμα λάθους.

Προηγουμένως είχαμε διορθώσει αυτό το πρόβλημα ελέγχοντας το μήκος του πίνακα προτού προσπαθήσουμε να προσπελάσουμε το πρώτο του στοιχείο. Η λύση ήταν καλή για μια απλή περίπτωση. Αν όμως έπρεπε να ελέγχουμε για κάθε πιθανή λανθασμένη συνθήκη σε κάθε γραμμή του κώδικα, τότε ο κώδικας θα είχε περισσότερους ελέγχους λαθών παρά καθαυτώ κώδικα. Επιπλέον θα έπρεπε να αρχίσετε να ελέγχετε για λάθη στις συνθήκες λαθών.

Στόχος του χειρισμού των εξαιρέσεων είναι να προσδιορίσουμε την κανονική ροή του προγράμματος σε τμήμα του κώδικα, χωρίς να ανησυχούμε για όλες τις ειδικές περιπτώσεις. Έπειτα, σ' ένα ξεχωριστό μπλοκ του κώδικα, θα καλύψουμε τις περιπτώσεις των εξαιρέσεων. Έτσι παράγουμε περισσότερο ευανάγνωστο κώδικα αφού δεν χρειάζεται να διακόπτουμε τη ροή του αλγόριθμου για να ελέγξουμε και να ανταποκριθούμε σε κάθε περίεργη κατάσταση. Το περιβάλλον εκτέλεσης είναι υπεύθυνο για τη μετακίνηση από την κανονική ροή του προγράμματος στους χειριστές εξαιρέσεων, όταν προκύπτει μια τέτοια συνθήκη.

Στην πράξη αυτό που κάνετε είναι να γράφετε blocks από κώδικα που μπορούν να παράγουν εξαιρέσεις. Εσείς δοκιμάζετε τις δηλώσεις που παράγουν τις εξαιρέσεις. Μέσα στα try blocks είστε ελεύθεροι να ενεργείτε σα να μη συμβαίνει κανένα λάθος. Έπειτα, μέσα σε ένα ή περισσότερα catch blocks γράφετε τη λογική του προγράμματος που χειρίζεται όλες τις ειδικές περιπτώσεις.

Ακολουθεί ένα παράδειγμα χειρισμού εξαιρέσεων σε Java που εφαρμόζεται στο πρόγραμμα Hello World:

```
// This is the Hello program in Java
class ExceptionalHello {

    public static void main (String args[]) {

      /* Now let's say hello */
      try {
        System.out.println("Hello " + args[0]);
      }
      catch (Exception e) {
        System.out.println("Hello whoever you are");
      }
  }

}
```

Κάποιες εξαιρέσεις είναι δυνατόν να τις αντιληφθούμε και να τις χειριστούμε, ενώ κάποιες άλλες είναι τόσο δύσκολες που το σύστημα εκτέλεσης παραιτείται. Ο compiler θα «χτυπήσει» αν γράφετε κώδικα και δεν συμπεριλάβετε τις όχι και τόσο επικίνδυνες εξαιρέσεις, αλλά εσείς πρέπει να προσέχετε για τις επικίνδυνες (όπως το ArrayIndexOutOfBoundsExceptions).

Πολλές φορές το να αντιληφθούμε την εξαίρεση δεν αρκεί γιατί δεν μπορούμε να την αντιμετωπίσουμε. Οι «κακές» εξαιρέσεις σταματούν το πρόγραμμα. Πολλές φορές το θέλετε κι εσείς οι ίδιοι. Σ' αυτή την περίπτωση καλείται η System.exit(int) μέθοδος και το εκτελούμενο πρόγραμμα κρεμάει.

Άλλες φορές μπορείτε να βγείτε από έναν βρόγχο και να συνεχίσετε με το υπόλοιπο πρόγραμμα. Αυτό είναι συνηθισμένο όταν μια εξαίρεση είναι αναμενόμενη ή όταν δεν επηρεάζει αρνητικά τη λογική του προγράμματος.

Μπορείτε να τυπώνετε ένα μήνυμα λάθους ή όχι. Αν γράφετε ένα χειριστή εξαιρέσεων και δεν περιμένετε να τον καλείτε τότε βάλτε ένα:

```
System.out.println("Error: " + e);
```

Μ' αυτόν τον τρόπο, αν κάτι πάει λάθος (που πάντα συμβαίνει) τουλάχιστον θα γνωρίζετε πού συμβαίνει. Μη βάλετε μήνυμα λάθους στους χειριστές του κανονικού προγράμματος. Θυμηθείτε ότι είναι εξαιρέσεις και όχι λάθη.

ΑΡΧΕΙΑ I/O ΚΑΙ STREAMS

Πολλές φορές γράφετε δεδομένα σ' ένα αρχείο αντί για την οθόνη του υπολογιστή. Κάτω από UNIX ή DOS μπορείτε να το κάνετε αυτό χρησιμοποιώντας τους τελεστές <και>.

Κάποιες φορές χρειάζεστε μια άλλη προσέγγιση που να γράφει συγκεκριμένα δεδομένα σε ένα αρχείο, ενώ όλα τα άλλα στην οθόνη. Ή χρειάζεστε πρόσβαση σε διάφορα αρχεία ταυτοχρόνως. Ή ίσως θέλετε να διαβάσετε τα δεδομένα ενός αρχείου που έχουν μια συγκεκριμένη μορφοποίηση. Στη Java όλες αυτές οι μέθοδοι συμβαίνουν σαν streams.

Τα streams είναι σημαντικό θέμα της Java και αργότερα θα αφιερώσουμε ολόκληρο κεφάλαιο σ' αυτό. Τώρα θέλουμε απλά να καλύψουμε τα πιο βασικά που θα σας επιτρέψουν να γράψετε, να διαβάσετε αρχεία και να επικοινωνήσετε με το χρήστη. Στην πραγματικότητα το System.out.println() που χρησιμοποιούμε συνεχώς είναι μια εφαρμογή των streams.

ΑΜΕΣΗ ΕΠΙΚΟΙΝΩΝΙΑ ΜΕ ΤΟΝ ΧΡΗΣΤΗ

Θα ξεκινήσουμε με ένα πολύ απλό πρόγραμμα το οποίο ζητά τα ονόματα των χρηστών και έπειτα τυπώνει έναν προσωπικό χαιρετισμό.

```
import java.io.*;

class PersonalHello {

  public static void main (String args[])
    {

      byte name[] = new byte[100];
      int nr_read = 0;

      System.out.println("What is your name?");
      try {
        nr_read = System.in.read(name);
        System.out.print("Hello ");
        System.out.write(name,0,nr_read);
      }
      catch (IOException e) {
        System.out.print("I'm Sorry.  I didn't catch your name.");
      }

    }

}
```

Το import.java.io.*; είναι το αντίστοιχο #include <studio.h> της C. Τα περισσότερα απ' αυτά που διαβάζετε και γράφετε στη Java είναι σε bytes. (Αργότερα θα δούμε ότι γίνεται να διαβάζουμε αρχεία κειμένου σε strings. Εδώ ξεκινήσαμε με έναν πίνακα από bytes που θα κρατάει το όνομα του χρήστη.

Πρώτα τυπώνουμε ένα ερώτημα ζητώντας το όνομα του χρήστη. Έπειτα διαβάζουμε το όνομα χρησιμοποιώντας τη μέθοδο System.in.read(). Αυτή η μέθοδος παίρνει έναν πίνακα byte σαν όρισμα και τοποθετεί αυτό που πληκτρολογεί ο χρήστης. Έπειτα όπως και πριν τυπώνουμε «Hello». Τέλος τυπώνουμε το όνομα του χρήστη.

Το πρόγραμμα στην πραγματικότητα δεν βλέπει τι πληκτρολογεί ο χρήστης μέχρι αυτός να πατήσει το enter. Έτσι δίνεται η ευκαιρία στον χρήστη να διορθώσει τυχόν λάθη. Όταν όμως πατήσει το πλήκτρο enter, ό,τι υπάρχει στη γραμμή τοποθετείται στον πίνακα.

Τι γίνεται αν ο χρήστης πληκτρολογήσει περισσότερους από 100 χαρακτήρες προτού πατήσει το enter; Σε πολλές προγραμματιστικές γλώσσες αυτό θα οδηγούσε στο σπάσιμο του προγράμματος. Η Java είναι πιο ασφαλής. Η μέθοδος System.in.read() ελέγχει το μήκος του πίνακα χρησιμοποιώντας την ιδιότητα name.length.

ΔΙΑΒΑΖΟΝΤΑΣ ΑΡΙΘΜΟΥΣ

Συχνά τα αλφαριθμητικά δεν αρκούν. Πολλές φορές ζητάμε από το χρήστη έναν αριθμό σαν είσοδο. Όλες οι είσοδοι του χρήστη έρχονται σαν αλφαριθμητικά, τα οποία θέλουμε να τα μετατρέψουμε σε αριθμούς.

Τώρα γράφουμε τη μέθοδο getNextInteger() που δέχεται έναν ακέραιο από το χρήστη.

```
static int getNextInteger() {

  String line;

  DataInputStream in = new DataInputStream(System.in);
  try {
    line = in.readLine();
    int i = Integer.valueOf(line).intValue();
    return i;
  }
  catch (Exception e) {
    return -1;
  }

} // getNextInteger ends here
```

ΔΙΑΒΑΖΟΝΤΑΣ ΜΟΡΦΟΠΟΙΗΜΕΝΑ ΔΕΔΟΜΕΝΑ

Είναι συχνή η περίπτωση να ζητάμε όχι έναν αριθμό αλλά πολλούς. Άλλες φορές θέλουμε να διαβάσουμε κείμενο και αριθμούς στην ίδια γραμμή. Γι' αυτό το σκοπό η Java παρέχει την κλάση StreamTokenizer.

ΓΡΑΦΟΝΤΑΣ ΕΝΑ ΑΡΧΕΙΟ ΚΕΙΜΕΝΟΥ

Κάποιες φορές θέλουμε να σώσουμε μία έξοδο σέρνοντάς την (scrolling) απλώς στην οθόνη. Για να το κάνουμε αυτό πρέπει να μάθουμε πώς να γράφουμε δεδομένα σ' ένα

αρχείο. Αντί να δημιουργήσουμε ένα καινούριο πρόγραμμα θα τροποποιήσουμε το Fahrenheit to Celsius.

```
// Write the Fahrenheit to Celsius table in a file

import java.io.*;

class FahrToCelsius  {

  public static void main (String args[]) {

    double fahr, celsius;
    double lower, upper, step;

    lower = 0.0;    // lower limit of temperature table
    upper = 300.0;  // upper limit of temperature table
    step  = 20.0;   // step size

    fahr = lower;

    try {

      FileOutputStream fout =  new FileOutputStream("test.out");

      // now to the FileOutputStream into a PrintStream
      PrintStream myOutput = new PrintStream(fout);

      while (fahr <= upper) {  // while loop begins here
        celsius = 5.0 * (fahr-32.0) / 9.0;
        myOutput.println(fahr + " " + celsius);
        fahr = fahr + step;
      } // while loop ends here

    }  // try ends here
    catch (IOException e) {
      System.out.println("Error: " + e);
      System.exit(1);
    }

  } // main ends here

}
```

Υπάρχουν 3 απαραίτητες συνθήκες για να γράψουμε μορφοποιημένη έξοδο σε ένα αρχείο:

1. Ανοίξτε ένα FileOutputStream χρησιμοποιώντας μία γραμμή όπως παρακάτω:

```
FileOutputStream fout = new FileOutputStream("test.out");
```

Αυτή η γραμμή αρχικοποιεί το FileOutputStream με το όνομα του αρχείου που θέλετε να γράψετε μέσα.

2. Μετατρέψτε το FileOutputStream σε PrintStream χρησιμοποιώντας μία δήλωση όπως:

PrintStream myOutput = new PrintStream(fout);

3. Αντί να χρησιμοποιήσετε το System.out.println(), χρησιμοποιείστε το myOutput.println().System.out και myOutput. Είναι διαφορετικά παραδείγματα της κλάσης PrintStream. Για να τυπώσουμε σε ένα διαφορετικό PrintStream κρατάμε ίδια τη σύνταξη αλλά αλλάζουμε το όνομα του PrintStream.

ΔΙΑΒΑΖΟΝΤΑΣ ΕΝΑ ΑΡΧΕΙΟ ΚΕΙΜΕΝΟΥ

Τώρα που ξέρουμε πώς να γράψουμε ένα αρχείο κειμένου ας προσπαθήσουμε να διαβάσουμε ένα. Ο παρακάτω κώδικας εφαρμόζει τη χρησιμότητα του Unix στη Java. Δέχεται μια σειρά από ονόματα αρχείων στη γραμμή εντολών και μετά τυπώνει αυτά τα ονόματα στη συγκεκριμένη έξοδο με τη σειρά που καταγράφηκαν.

```
// Imitate the Unix cat utility

import java.io.*;

class cat  {

  public static void main (String args[]) {

  String thisLine;

  //Loop across the arguments
  for (int i=0; i < args.length; i++) {

  //Open the file for reading
  try {
    FileInputStream fin =  new FileInputStream(args[i]);

    // now turn the FileInputStream into a DataInputStream
    try {
      DataInputStream myInput = new DataInputStream(fin);

      try {
        while ((thisLine = myInput.readLine()) != null) {  // while
loop begins here
          System.out.println(thisLine);
        } // while loop ends here
      }
      catch (Exception e) {
       System.out.println("Error: " + e);
      }
    } // end try
    catch (Exception e) {
      System.out.println("Error: " + e);
    }

   } // end try
   catch (Exception e) {
    System.out.println("failed to open file " + args[i]);
```

```
    System.out.println("Error: " + e);
  }
  } // for end here

  } // main ends here

}
```

ΠΕΡΙΛΗΨΗ

Αν έχετε φτάσει μέχρι εδώ είστε πια ικανοί να κάνετε αρκετή δουλειά σε Java. Μπορεί να μην το πιστεύετε επειδή δεν έχουμε μιλήσει ακόμα για τα ειδικά χαρακτηριστικά της Java, όπως είναι οι μίνι εφαρμογές.Η Java που γνωρίζετε είναι ικανή να χειριστεί οποιοδήποτε πρόβλημα που παλιότερα χειρίζονταν οι Basic και η Fortran 77.

Το FahrToCelsius είναι μια πολύ βασική εφαρμογή που μπορεί να γραφτεί σε όλες σχεδόν τις προγραμματιστικές γλώσσες, από τον πιο αρχαίο κώδικα μέχρι την πιο ανεπτυγμένη μηχανή LISP. Είναι σημαντικό να σημειώσουμε ότι η Java λύνει αυτό το πρόβλημα τόσο εύκολα όσο και μια γλώσσα που συνδυάζει αριθμητικό και επιστημονικό προγραμματισμό, όπως η Fortran, η C. Ο κώδικας αυτός μεταφράστηκε σχεδόν κατά λέξη από τους Kernighan και Ritchie. Απαιτήθηκαν μόνο κάποιες μικρές σημασιολογικές αλλαγές για να γίνει ένα έγκυρο και αποτελεσματικό πρόγραμμα σε Java. Παρόλο που η Java έχει πολλά χαρακτηριστικά που την κάνουν κατάλληλη για πολύπλοκες εφαρμογές, είναι κατάλληλη και για κλασσικά αριθμητικά προγράμματα, κάτι που δεν ισχύει για τους ανταγωνιστές όπως SmallTalk και LISP.

Στην πραγματικότητα η Java μπορεί να ξεπεράσει τη Fortran και τη C στις αριθμητικές εφαρμογές, όταν η ακρίβεια, η αξιοπιστία και η ευελιξία είναι πιο σημαντικές από την ταχύτητα.

ΚΕΦΑΛΑΙΟ 3: Μίνι Εφαρμογές (Applets)

Συγκεκριμένοι προγραμματιστές λέγεται ότι «γράφουν Fortran σε όλες τις γλώσσες». Τώρα πια έχετε τη γνώση να συνοδεύετε τη Java με ό,τι μπορεί να γίνει μέσα στα όρια του ANSI-standard Fortran 77.

Πρόκειται να δουλέψουμε με προγραμματισμό που καθοδηγείται από γεγονότα. Αυτός ο τρόπος προγραμματισμού πρέπει να είναι γνωστός στους προγραμματιστές Macintosh και Windows. Σ’ αυτά τα περιβάλλοντα η λογική του προγράμματος δεν ρέει από την κορυφή στη βάση του προγράμματος όπως στους πιο διαδικαστικούς κώδικες. Το λειτουργικό σύστημα συλλέγει τα γεγονότα και το πρόγραμμα ανταποκρίνεται σ’ αυτό.

Κάθε πρόγραμμα έχει ένα event loop. Αυτός είναι ένας βρόγχος while που δουλεύει ασταμάτητα. Σε κάθε πέρασμα μέσα από το loop, η εφαρμογή επανακτά το επόμενο γεγονός από τη σειρά γεγονότων και ανταποκρίνεται ανάλογα.

Οι μίνι εφαρμογές (applets) συμπεριφέρονται παρόμοια. Το περιβάλλον εκτέλεσης (π.χ. ο browser) φροντίζει για τον βρόγχο της εφαρμογής.

HELLO WORLD : Η ΜΙΝΙ ΕΦΑΡΜΟΓΗ

Ο λόγος για τον οποίο όλοι είναι ενθουσιασμένοι με τη Java είναι ότι επιτρέπει να γράφουμε αλληλεπιδραστικές εφαρμογές στο δίκτυο. Το Hello World δεν είναι ένα αλληλεπιδραστικό πρόγραμμα αλλά ας δούμε την παρακάτω έκδοση:

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

public class HelloWorldApplet extends Applet {

  public void paint(Graphics g) {
    g.drawString("Hello world!", 50, 25);
  }

}
```

Αυτή η έκδοση είναι λίγο πιο πολύπλοκη από την προηγούμενη αλλά θέλει κι άλλη προσπάθεια για να τρέξει καλύτερα.

Πρώτα πληκτρολογήστε τον κώδικα και σώστε το σ' ένα αρχείο που να λέγεται HelloWorldApplet.java στο javahtml/κλάσειςidirectory. Μεταγλωτίστε το πρόγραμμα πληκτολογώντας javac HelloWorldApplet.java στο command line prompt.

Αν όλα πάνε καλά θα δημιουργηθεί ένα αρχείο με το όνομα HelloWorldApplet.class. Αυτό το αρχείο πρέπει να βρίσκεται στο directory των κλάσεων. Τώρα πρέπει να δημιουργήσετε ένα αρχείο HTML που θα συμπεριλαμβάνει την εφαρμογή σας. Ακολουθεί το απλό αρχείο HTML.

```
<HTML>
<HEAD>
<TITLE> Hello World </TITLE>
</HEAD>

<BODY>
This is the applet:<P>
<APPLET codebase="classes" code="HelloWorldApplet.class" width=200
height=200 ></APPLET>
</BODY>
</HTML>
```

Σώστε αυτό το αρχείο σαν «HelloWorldApplet.html» στο javahtml directory. Όταν το κάνετε φορτώστε το αρχείο HTML σ' έναν Java browser, όπως HotJava ή Netscape 2.0. Θα πρέπει να δείτε το ακόλουθο:

This is the applet:

Hello World!

Αν η μίνι εφαρμογή μεταγλωτιστεί χωρίς λάθος και παράγει το αρχείο HelloWorldApplet.class και ακόμα δεν βλέπετε το αλφαριθμητικό «Hello World» στο browser, τότε πιθανότατα το αρχείο κλάσης είναι σε λάθος μέρος. Ελέγξτε αν το αρχείο .html είναι στο directory javahtml και το αρχείο .class είναι στο java/κλάσειςιdirectory.

ΕΞΕΤΑΖΟΝΤΑΣ ΤΗ ΜΙΝΙ ΕΦΑΡΜΟΓΗ HELLO WORLD

Το Hello World Applet πρόσθεσε κάποια πράγματα στο Hello World Application. Κινούμενοι από πάνω προς τα κάτω, το πρώτο πράγμα που προσέχουμε είναι οι δύο γραμμές.

```
import java.applet.Applet;

import java.awt.Graphics;
```

Η δήλωση import στη Java είναι παρόμοια με τη δήλωση #include στην C ή στην C++. Εισάγει τις κλάσεις που περιέχονται σ' ένα πακέτο κάπου αλλού. Ένα πακέτο είναι ένα σύνολο από σχετιζόμενες κλάσεις. Σ' αυτήν την περίπτωση ζητάμε πρόσβαση στις κλάσεις που συμπεριλαμβάνονται στο java.applet.Applet και java.awt.Graphics.

Η επόμενη διαφορά που παρατηρούμε είναι ο ορισμός της κλάσης:

```
public class HelloWorldApplet extends Applet
```

Η λέξη κλειδί extends υπονοεί ότι αυτή η κλάση είναι μια υποκλάση της κλάσης Applet. Ή αλλιώς η Applet είναι η υπερκλάση του HelloWorldApplet. Η κλάση Applet προσδιορίζεται στο πακέτο java.applet.Applet. Εφόσον το HelloWorldApplet είναι υποκλάση του Applet, τότε το HelloWorldApplet κληρονομεί όλη τη λειτουργικότητα του αρχικού Applet. Ό,τι μπορεί να κάνει η Applet, μπορεί να το κάνει και το HelloWorldApplet επίσης.

Η επόμενη διαφορά είναι λιγότερο ορατή (εκτός κι αν είστε έμπειρος προγραμματιστής της C). Δεν υπάρχει κύρια μέθοδος. Οι Applets δεν τις χρειάζονται. Η κύρια μέθοδος είναι στον browser ή στον AppletViewer, όχι στην ίδια την Applet. Οι Applets προσφέρουν επιπλέον λειτουργικότητα αλλά δεν μπορούν να τρέξουν χωρίς να τις φιλοξενήσει ένα κύριο πρόγραμμα.

Αντί να αρχίσουν από ένα συγκεκριμένο σημείο του κώδικα, οι μίνι εφαρμογές οδηγούνται από γεγονότα. Μία εφαρμογή περιμένει για ένα γεγονός όπως το πάτημα

ενός πλήκτρου ή το κλικ του ποντικιού και έπειτα εκτελεί το κατάλληλο event handler. Εφόσον αυτό είναι το πρώτο μας πρόγραμμα είχαμε μόνο έναν event handler, το paint.

Οι περισσότερες εφαρμογές πρέπει να χειρίζονται το paint. Αυτό το γεγονός συμβαίνει όταν κάποιο τμήμα της ορατής περιοχής της εφαρμογής δεν είναι καλυμμένο και πρέπει να βαφτεί.

Η μέθοδος paint χρησιμοποιεί ένα αντικείμενο Graphics που επιλέγουμε να το λέμε g. Η κλάση Graphics ορίζεται στο πακέτο java.awt.Graphics. Μέσα στη μέθοδο paint καλούμε την μέθοδο g's drawString να γράψει το αλφαριθμητικό «Hello World» στις συντεταγμένες (50,25). Αυτό είναι 50 pixels αριστερά και 25 pixels κάτω από την πάνω αριστερή γωνία της εφαρμογής. Θα μιλήσουμε περισσότερο για το σύστημα συντεταγμένων αργότερα. Η ζωγραφική λαμβάνει χώρα όταν τμήμα της οθόνης που συμπεριλαμβάνει την εφαρμογή είναι αρχικά καλυμμένη αλλά μετά χρειάζεται φρεσκάρισμα.

Η ΕΤΙΚΕΤΑ (TAG) APPLET ΤΗΣ HTML

Οι Applets συμπεριλαμβάνονται στις σελίδες του δικτύου χρησιμοποιώντας την ετικέτα <APPLET>. Η ετικέτα <APPLET> είναι σχεδόν όμοια με την ετικέτα <IMG>. Όπως το <IMG> έτσι και το <APPLET> πρέπει να αναφέρεται σ' ένα πηγαίο αρχείο που δεν είναι τμήμα της σελίδας HTML στην οποία βρίσκεται. Τα IMG το κάνουν αυτό με το SRC=parameter. Τα APPLET το κάνουν αυτό με το CODE=parameter. Η παράμετρος CODE λέει στον browser πού να κοιτάξει για το μεταγλωτισμένο αρχείο .class. Αυτό σχετίζεται με το πού βρίσκεται το πηγαίο κείμενο. Έτσι αν έχουμε CODE=animation.class, τότε το αρχείο animation.class πρέπει να βρίσκεται στο http://metalab.unc.edu/javafaq /animation.class.

Για λόγους που οι συγγραφείς της HTML δεν γνωρίζουν, αλλά που πιθανόν να έχουν σχέση με τα πακέτα και τα classpaths, αν η εφαρμογή δεν βρίσκεται στο ίδιο directory με τη σελίδα τότε δεν δίνετε απλά ένα URL στην τοποθεσία του. Δείχνετε το directory όπου το αρχείο .class χρησιμοποιεί την παράμετρο CODEBASE. Πρέπει ακόμα να χρησιμοποιείτε το CODE για να δώσετε όνομα στο αρχείο .class.

Το APPLET, όπως και το IMG, έχει αρκετές παραμέτρους που προσδιορίζουν το πώς τοποθετείται στη σελίδα. Οι παράμετροι HEIGHT και WIDTH, που λειτουργούν όπως ακριβώς και στο IMG, καθορίζουν το πόσο μεγάλο θα είναι το ορθογώνιο που πρέπει να αφήσει ο browser για την εφαρμογή. Αυτοί οι αριθμοί εκφράζονται σε pixels. Το ALIGN (στους browsers που το υποστηρίζουν, καθορίζει πώς τοποθετείται στη σελίδα το ορθογώνιο της εφαρμογής σε σχέση με τα άλλα στοιχεία. Οι πιθανές τιμές συμπεριλαμβάνουν τις LEFT, RIGHT, TOP, TEXTTOP, MIDDLE, ABSMIDDLE, BASELINE, BOTTOM και ABSBOTTOM. Τέλος, όπως και στα IMG μπορούμε να καθορίσουμε το HSPACE και το VSPACE για να εκφράσουμε το κενό διάστημα ανάμεσα στην εφαρμογή και το κείμενο.

Το APPLET έχει επίσης και την ετικέτα ALT. Το ALT δεν εφαρμόζεται σε κανέναν browser μέχρι τώρα. Μία ετικέτα ALT χρησιμοποιείται από έναν browser που αναγνωρίζει την ετικέτα APPLET αλλά για κάποιο λόγο δεν μπορεί να παίξει την εφαρμογή. Για παράδειγμα, αν μια εφαρμογή πρέπει να γράψει ένα αρχείο στο σκληρό δίσκο αλλά δεν του επιτρέπεται, τότε ο browser θα πρέπει να δείξει το κείμενο ALT.

Περνώντας παραμέτρους στα applets(μίνι εφαρμογές).

Η περιοχή ανάμεσα στο άνοιγμα και το κλείσιμο της ετικέτας του applet χρησιμοποιείται, επίσης, για να περαστούν παράμετροι στις μίνι εφαρμογές. Αυτό επιτυγχάνεται με τη χρήση της ετικέτας PARAM.HTML και της μεθόδου getParameter του java.applet.Applet.class.

Για να το δείξουμε αυτό θα χρησιμοποιήσουμε το HelloWorldApplet σε μια γενικά σχεδιασμένη εφαρμογή. Για να το κάνουμε αυτό θα περάσουμε στο applet τους παραμέτρους εκείνους που ορίζουν τα αλφαριθμητικά.

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

public class DrawStringApplet extends Applet {

  String input_from_page;

  public void init() {
    input_from_page = getParameter("String");
  }

  public void paint(Graphics g) {
    g.drawString(input_from_page, 50, 25);
  }

}
```

Τώρα χρειάζεται να δημιουργήσεις ένα html αρχείο που θα περιλαμβάνει το applet σου. Αυτό θα το κάνει το ακόλουθο html αρχείο:

```
<HTML>
<HEAD>
<TITLE> Draw String </TITLE>
</HEAD>

<BODY>
This is the applet:<P>
<APPLET codebase="classes" code="DrawStringApplet.class" width=200
height=200><PARAM name="String" value="Howdy, there!"></APPLET>
</BODY>
</HTML>
```

Φυσικά είσαι ελεύθερος να αλλάξεις το ‘Howdy, there’ σε ένα string της επιλογής σου. Σημείωσε ότι αυτό σου επιτρέπει να αλλάξεις το output του applet χωρίς να αλλάξεις ή να ξαναδιορθώσεις τον κώδικα.

Δεν είσαι περιορισμένος σε μια μόνο παράμετρο. Μπορείς να περάσεις πολλές ονομαζόμενες παραμέτρους σε ένα applet.

Η μέθοδος getParameter είναι απλή. Της δίνεις ένα string το οποίο είναι το όνομα της παραμέτρου που θέλεις. Σου επιστρέφει ένα string, που έιναι η τιμή της παραμέτρου. Όλοι οι παράμετροι χρησιμοποιούνται σαν string. Εαν θέλεις να πάρεις κάτι άλλο, όπως έναν ακέραιο, θα πρέπει να το χρησιμοποιήσεις σαν string και να το μετατρέψεις, έπειτα, στον τύπο που πραγματικά θέλεις.

Η ετικέτα PARAM HTML είναι επίσης απλή. Τοποθετείται ανάμεσα στο <APPLET> και στο </APPLET>. Έχει δυο παραμέτρους δικές του, το NAME(όνομα) και το VALUE(τιμή). Το NAME καθορίζει ποιοι παράμετροι είναι για τη μέθοδο getParameter και VALUE είναι η τιμή της παραμέτρου σαν string. Και τα δυο πρέπει να εσωκλειστούν σε διπλά εισαγωγικά, όπως όλοι οι άλλες παράμετροι ετικέτας html.

EVENTS(ΓΕΓΟΝΟΤΑ) ΚΑΙ APPLETS(ΜΙΝΙ ΕΦΑΡΜΟΓΕΣ)

EVENT TUTOR APPLET

Το ακόλουθο applet είναι σχεδιασμένο για να σου δώσει κάποια αίσθηση για το τι είναι προγραμματισμός καθοδηγούμενος από γεγονότα και ποια είναι τα διάφορα γεγονότα που πιθανόν θα συναντήσεις. Όταν συμβαίνει ένα γεγονός το applet ανταποκρίνεται τυπώνοντας το όνομα του γεγονότος στο command line.

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

public class EventTutor extends Applet {

  public void init() {
    System.out.println("init event");
  }

  public void paint(Graphics g) {
    System.out.println("paint event");
  }

  public void start() {
    System.out.println("start event");
  }

  public void destroy() {
    System.out.println("destroy event");
  }

  public void update(Graphics g) {
    System.out.println("update event");
```

```
  }

  public boolean mouseUp(Event e, int x, int y) {
    System.out.println("mouseUp event");
    return false;
  }

  public boolean mouseDown(Event e, int x, int y) {
   System.out.println("mouseDown");
   return false;
  }

  public boolean mouseDrag(Event e, int x, int y) {
    System.out.println("mouseDrag event");
        return false;
  }

  public boolean mouseMove(Event e, int x, int y) {
    System.out.println("mouseMove event");
    return false;
  }

  public boolean mouseEnter(Event e, int x, int y) {
    System.out.println("mouseEnter event");
    return false;
  }

 public boolean mouseExit(Event e, int x, int y) {
    System.out.println("mouseExit event");
    return false;
  }

  public void getFocus() {
    System.out.println("getFocus event");
  }

  public void gotFocus() {
    System.out.println("gotFocus event");
  }

  public void lostFocus() {
    System.out.println("lostFocus event");
  }

  public boolean keyDown(Event e, int x) {
    System.out.println("keyDown event");
    return true;
  }

}
```

Καθώς θα έχεις διορθώσει και φορτώσει αυτό το applet παίξε μαζί του. Κάνε κλικ στο παράθυρο του applet. Κάνε διπλό κλικ με το ποντίκι. Κάνε κλικ και σύρε το ποντίκι. Τύπωσε κάποιο κείμενο. Ξαναμέτρησε το παράθυρο του browser. Κάλυψέ το και μετά

άνοιξέ το. Καθώς θα το κάνεις κράτα το βλέμμα σου στο standard output (Java console στο δίκτυο).

Παρακάτω υπάρχουν μερικές ερωτήσεις;

1. Μπορείς να έχεις ένα mouseDown γεγονός, που δεν ακολουθείται από ένα mouseUp γεγονός;
2. Μπορείς να έχεις ένα mouseDown γεγονός, που δεν ακολουθείται από ένα mouseDrag γεγονός;
3. Μπορείς να έχεις ένα mouseUp γεγονός, που δεν προηγείται ενός mouseDown γεγονότος;
4. Τι πρέπει να συμβεί για να πραγματοποιηθεί ένα paint γεγονός;
5. Ποια είναι τα πιο κοινά γεγονότα; Γιατί;
6. Υπάρχουν γεγονότα που δεν βλέπεις;
7. Πόσες φορές μπορείς να καλέσεις το start γεγονός και πόσες το stop γεγονός;
8. Από αυτά τα γεγονότα μπορείς να καταλάβεις πώς θα το κάνεις; Με πόσους διαφορετικούς τρόπους μπορείς να το κάνεις;

Υπάρχουν αρκετά νέα πράγματα σ' αυτόν τον κώδικα, αλλά δεν είναι ιδιαίτερα δύσκολα. Πρώτον, είναι η δήλωση import java.awt.*. Αυτήν τη φορά χρειαζόμαστε παραπάνω από μια κλάση από το πακέτο awt. Για να μην ανησυχούμε ποιο να εισάγουμε τα παίρνουμε όλα με το *. Ο compiler είναι αρκετά 'έξυπνος' για να συνδέσει μόνο αυτά που πραγματικά χρειάζεται.

Τελικά, υπάρχει ένα σύνολο από νέες μεθόδους γεγονότων. Θα τις καλύψουμε στο επόμενο κεφάλαιο. Προς το παρόν δες κάτω από ποιες συνθήκες μπορούν όλα αυτά να συμβούν.

Δημιουργώντας μία λίστα

Είναι ανάρμοστο να χρησιμοποιούμε τον τύπο System.out.println() σε μια μίνι εφαρμογή. Σε μερικά συστήματα αυτό μπορεί να μην δουλεύει καθόλου. Όμως, έχει το πλεονέκτημα να είναι οικείο και εύκολο. Για πιο σοβαρές εφαρμογές θα πρέπει να ζωγραφίσεις το κείμενο σου στο παράθυρο της μίνι εφαρμογής. Υπάρχουν τουλάχιστον τρεις διαφορετικοί τρόποι για να το κάνεις αυτό. Για τους δικούς μας σκοπούς αυτό που κάνει μεγαλύτερη αίσθηση είναι η χρήση λίστας.

Μια λίστα είναι μια κυλιόμενη λίστα από συμβολοσειρές που ορίζονται στο java.awt.List. Θα δημιουργήσουμε μια καινούργια λίστα, ακριβώς όπως δημιουργούμε ένα αντικείμενο. Ο συγκεκριμένος κατασκευαστής constructor) που χρησιμοποιούμε, ψάχνει για έναν int (έτσι είναι το όνομα των ορατών γραμμών) και έναν boolean, το οποίο λέει αν επιτρέπονται ή όχι πολλαπλές επιλογές. Θα ζητήσουμε 25 γραμμές και καθόλου πολλαπλές επιλογές.

List theList;

theList = new List(25, false);

Θα προσθέσουμε strings στην λίστα χρησιμοποιώντας την μέθοδο addItem από το List.

theList.addItem("This is a list item");

Τέλος χρειάζεται να προσθέσουμε αυτήν την λίστα στο applet μας(πιο συγκεκριμένα στο applet container).Αυτό το κάνουμε με τη γραμμή: add(theList); στην init μέθοδο. Αυτό είναι όλο. Μπορούμε να χρησιμοποιήσουμε το ίδιο applet με αυτό που χρησιμοποιήσαμε πριν, με αυτές τις μικρές αλλαγές.

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

public class EventList extends Applet {

  List theList;

  public void init() {
    theList = new List(25, false);
    add(theList);
    theList.addItem("init event");
  }

  public void paint(Graphics g) {
    theList.addItem("paint event");
  }

  public void start() {
    theList.addItem("start event");
  }

  public void destroy() {
    theList.addItem("destroy event");
  }

  public void update(Graphics g) {
    theList.addItem("update event");
  }

  public boolean mouseUp(Event e, int x, int y) {
    theList.addItem("mouseUp event");
    return false;
  }

  public boolean mouseDown(Event e, int x, int y) {
   theList.addItem("mouseDown");
   return false;
  }

  public boolean mouseDrag(Event e, int x, int y) {
    theList.addItem("mouseDrag event");
```

```
        return false;
  }

  public boolean mouseMove(Event e, int x, int y) {
    theList.addItem("mouseMove event");
    return false;
  }

  public boolean mouseEnter(Event e, int x, int y) {
    theList.addItem("mouseEnter event");
    return false;
  }

 public boolean mouseExit(Event e, int x, int y) {
    theList.addItem("mouseExit event");
    return false;
  }

  public void getFocus() {
    theList.addItem("getFocus event");
  }

  public void gotFocus() {
    theList.addItem("gotFocus event");
  }

  public void lostFocus() {
    theList.addItem("lostFocus event");
  }

  public boolean keyDown(Event e, int x) {
    theList.addItem("keyDown event");
    return true;
  }

}
```

Τα γεγονότα(events)

Σ' αυτήν την παράγραφο θα προσπαθήσουμε να κατηγοριοποιήσουμε τα γεγονότα στα οποία πρέπει να ανταποκριθεί το applet σου. Δεν θα χρειάζεται κάθε applet να ανταποκρίνεται σε όλα τα γεγονότα.

Για μερικά events θα εσωκλείσουμε νέες μεθόδους για το EventTutor applet, που δίνει μια ακόμα πληροφορία. Θα χρειαστεί να αντικαταστήσουμε την παλιά μέθοδο με καινούργια.

init

Η init() μέθοδος καλείται όταν το applet σου αρχίζει να εκτελείται. Το Netscape λέγεται, επίσης ότι καλεί αυτή τη μέθοδο σε άλλες στιγμές, όπως όταν ένα applet ξαναφορτώνεται ή όταν επιστρέφεις σε σελίδα που περιλαμβάνει ένα applet. Γενικά, χρησιμοποιείς αυτήν τη μέθοδο για να καθορίσεις κάθε δομή δεδομένων και να παρουσιάσεις οποιαδήποτε

λειτουργία χρειάζεσαι για να είσαι έτοιμος να τρέξεις το applet. Από τότε που καλείται μια φορά είναι εύκολο να χάσεις την init() μέθοδο στο EventTutor applet. Αν είναι απαραίτητο ανακαθόρισε το standard output σε ένα αρχείο και κοίτα στην πρώτη γραμμή του αρχείου να το δεις.

```
public void init() {

System.out.println("init event");

}
```

paint

Έχουμε ήδη δει τη μέθοδο paint(). Με αυτήν θα κάνεις όλη τη ζωγραφική. Μπορείς να γράψεις στην οθόνη του applet με τη μέθοδο paint. Θα υπάρξουν, όμως, στιγμές που θα θες να γράψεις σε ένα offscreen image με μια άλλη μέθοδο και μετά σχεδόν αμέσως να αντιγράψεις αυτό το image στην οθόνη στην μέθοδο paint().

```
public void paint(Graphics g) {

theList.addItem("paint event");

}
```

stop

Ένα μήνυμα stop() λέει ότι ο χρήστης δεν κοιτάζει πλέον την σελίδα που περιέχει το applet. Αυτό συμβαίνει όχι επειδή ο χρήστης αφήνει τη σελίδα, αλλά ελαχιστοποιεί το παράθυρο. Από αυτήν την στιγμή πρέπει να σταματήσει κάθε CPU eating ενέργεια. Καθώς ο χρήστης επιστρέφει στη σελίδα καλείται η μέθοδος start().

```
public void stop()

theList.addItem("stop event");

}
```

start

Η μέθοδος start() καλείται όταν ένας χρήστης επικεντρώνει την προσοχή του σε ένα applet π.χ. μετά από μεγιστοποίηση ενός παραθύρου ή επιστροφή στην σελίδα του applet. Καλείται μετά από την init() μέθοδο. Ο κώδικας αρχικοποίησης, που παρουσιάζεται κάθε φορά που ένα applet ξαναρχίζει , πρέπει να τοποθετηθεί εδώ.

```
public void start() {

theList.addItem("start event");

}
```

destroy

Η μέθοδος destroy() καλείται πριν το applet αποφορτωθεί εντελώς. Καλείται μετά τη μέθοδο stop(). Οι χρήστες ίσως ξαναφορτώσουν το applet αργότερα, αλλά αν το κάνουν θα ήταν σαν να μην το είχαν δει ποτέ πριν. Όλες οι μεταβλητές, στατικές, μέλη, τοπικές ή άλλες θα αρχικοποιηθούν. Αν έχεις κανένα τελικό ξεκαθάρισμα να κάνεις (π.χ. να στείλεις output πίσω στον http server) καν' το εδώ.

```
public void destroy() {

theList.addItem("destroy event");

}
```

update

Η μέθοδος update() καλείται αυτόματα από το σύστημα.

```
public void update(Graphics g) {

theList.addItem("update event");

}
```

mouseUp

Η μέθοδος mouseUp() καλείται όταν το κουμπί του ποντικιού απελευθερώνεται στο applet. Στις περισσότερες περιπτώσεις αυτό είναι το γεγονός που θα θέλεις να παρακολουθήσεις και όχι το mouseDown. Ένα κουμπί είναι τυπικά φωτισμένο όταν το κουμπί του ποντικιού πατιέται πάνω σ' αυτό, αλλά δεν είναι ενεργοποιημένο μέχρι ο χρήστης να το απελευθερώσει. Αυτό δίνει στο χρήστη την ευκαιρία να αλλάξει τη γνώμη του μετακινώντας τον δρομέα από το αντικείμενο χωρίς να το απελευθερώσει.

Η εξαίρεση θα είναι όταν θα θες να συνεχιστεί μια πράξη, καθώς το κουμπί του ποντικιού είναι κρατημένο.

Η μέθοδος mouseUp() λαμβάνει επίσης τις συντεταγμένες του σημείου στο οποίο το ποντίκι ήταν απελευθερωμένο.

```
public boolean mouseUp(Event e, int x, int y) {

theList.addItem("mouseUp event at ("+x+","+y+ ")");

return false;

}
```

mouseDown

Η μέθοδος mouseDown καλείται όταν το κουμπί του ποντικιού πατιέται στο applet σου. Στις περισσότερες περιπτώσεις θέλεις να περιμένεις για ένα mouseUp γεγονός πριν λάβεις οποιαδήποτε δράση. Η μέθοδος mouseDown λαμβάνει επίσης τις συντεταγμένες του σημείου όπου το ποντίκι απελευθερώθηκε.

```
public boolean mouseDown(Event e, int x, int y) {

theList.addItem("mouseDown event at (" + x + "," + y + ")");

return false;

}
```

mouseDrag

Η μέθοδος mouseDrag συμβαίνει όταν ένας χρήστης μετακινεί το ποντίκι καθώς διατηρεί πατημένο το κουμπί του ποντικιού. Η mouseDrag() μέθοδος λαμβάνει τις συντεταγμένες του σημείου που βρίσκεται το ποντίκι όταν συμβαίνει το γεγονός.

```
public boolean mouseDrag(Event e, int x, int y) {

theList.addItem("mouseDrag event at (" + x + "," + y + ")");

return false;

}
```

mouseMove

Η μέθοδος mouseMove συμβαίνει όταν ένας χρήστης μετακινεί το ποντίκι χωρίς να διατηρεί πατημένο το κουμπί του ποντικιού. Η μέθοδος mouseMove() λαμβάνει τις συντεταγμένες του σημείου στο οποίο βρίσκεται το ποντίκι όταν συμβεί το γεγονός.

```
public boolean mouseMove(Event e, int x, int y) {

theList.addItem("mouseMove event at (" + x + "," + y + ")");

return false;

}
```

mouseEnter

Το applet σου λαμβάνει ένα mouseEnter γεγονός, όταν ο δρομέας εισάγει το applet από κάπου αλλού. Θα λάβεις, επίσης, τις συντεταγμένες του σημείου στο οποίο ο δρομέας εισήγαγε το applet. Μετά που θα συμβεί αυτό, ακολουθείται τυπικά ένα Stream από τα mouseMoved γεγονότα, καθώς ο δρομέας συνεχίζει μέσα στο applet.

```
public boolean mouseEnter(Event e, int x, int y) {

theList.addItem("mouseEnter event at " + x + "," + y + ")");

return false;

}
```

mouseExit

Το applet σου λαμβάνει ένα mouseExit γεγονός, όταν ο δρομέας αφήνει το applet σου. Θα λάβεις επίσης τις συντεταγμένες του σημείου στο οποίο ο δρομέας αφήνει το applet σου.

```
public boolean mouseExit(Event e, int x, int y) {

theList.addItem("mouseExit event at (" + x + "," + y + ")");

return false;

}
```

getFocus

```
public void getFocus() {

theList.addItem("getFocus event");

}
```

gotFocus

```
public void gotFocus() {theList.addItem('gotFocus event');}
```

lostFocus

```
public void lostFocus() {

theList.addItem("lostFocus event");

}
```

keyDown

Ένα keyDown γεγονός πραγματοποιείται όταν ο χρήστης πιέζει ένα κλειδί, καθώς το applet του είναι ενεργό. Ένας ακέραιος κωδικός κλειδιού επιστρέφεται δείχνοντας ποιο

κλειδί ήταν πιεσμένο. Θα θελήσεις να το μετατρέψεις σε ένα χαρακτήρα και να λάβεις το πραγματικό γράμμα.

public boolean keyDown(Event e, int x) {

theList.addItem("The " + (char) x + " key was pressed.");

return false;

}

Ζωγραφίζοντας κείμενο (Drawing Text)

Ζωγραφίζοντας γραφικά: Γραμμές, κύκλους, ορθογώνια και χρώματα

Ζωγραφίζοντας ορθογώνια

Στη συνέχεια θα γράψουμε ένα applet που συμπληρώνει την οθόνη με πολλά τυχαίου μεγέθους και θέσεως ορθογώνια στο στυλ του Piet Mondrian.Στην πορεία θα μάθουμε τις βάσεις των γραφικών του applet.Στο πρώτο applet θα ζωγραφίσουμε απλά ένα ορθογώνιο στην οθόνη. Θα πάρουμε το μέγεθος του applet, όπως ορίζεται στο html αρχείο και μετά θα ζωγραφίσουμε ένα ορθογώνιο γύρω από το applet. Εδώ είναι ο κώδικας:

```
//Draw a rectangle

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

public class Mondrian1 extends Applet {

  int height, width;

  public void init() {

    Dimension d = size();
    height = d.height;
    width = d.width;
    repaint();

  }

  public void paint(Graphics g) {

    g.drawRect(0, 0, width, height);

  }

}
```

Μεταγλώττισε αυτό το applet, μετακίνησε το αρχείο κλάσης που προκύπτει στο directory της κλάσης και δημιούργησε ένα html αρχείο που να δείχνει σ' αυτό. Όρισε το ύψος του applet στα 300 pixels και το πλάτος στα 300 pixels επίσης. Φόρτωσε αυτό το αρχείο στον browser. Τι βλέπεις; Προφανώς όχι αυτό που περίμενες. Θα δεις μισό ορθογώνιο. Τι συνέβη στο άλλο μισό;

Αυτό καλείται λάθος fencepost. Το applet λαμβάνει χώρα σε ένα τετράγωνο με 300 pixels ύψος και 300 pixels φάρδος. Όμως, η υψηλότερη αριστερή γωνία του applet αρχίζει στα (0,0) και όχι στα (1,1). Αυτό σημαίνει ότι το applet συμπεριλαμβάνει τα σημεία με χ και y συντεταγμένες ανάμεσα στο 0 και 299, όχι ανάμεσα στο 0 και 300. Ζωγραφίσαμε ένα ορθογώνιο με 301 pixels πλάτος και έτσι οι γραμμές αποκόπηκαν. Αυτό είναι τυχαίο όμως. Όχι μόνο μας δίνει την ευκαιρία να απομακρύνουμε ένα fencepost λάθος, αλλά μας δείχνει επίσης και κάτι άλλο. Στη Java το σύστημα συντεταγμένων για ένα applet αρχίζει από την πάνω αριστερή γωνία και αυξάνει δεξιά και κάτω. Αυτό είναι κοινό στα γραφικά, αλλά είναι διαφορετικό από το σύστημα συντεταγμένων όπου η κατεύθυνση του αυξανόμενου y είναι γενικά θεωρημένο να είναι πάνω.

Είναι εύκολο να διορθώσεις ένα fencepost λάθος. Αλλάζουμε απλά το g.drawRect(0, 0, width, height); σε g.drawRect(0,0, width-1, height-1);

```
//Draw a rectangle

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

public class Mondrian2 extends Applet {

  int height, width;

  public void init() {

    Dimension d = size();
    height = d.height;
    width = d.width;
    repaint();

  }

  public void paint(Graphics g) {

    g.drawRect(0, 0, width-1, height-1);

  }

}
```

Όπως συνήθως μεταγλώττισέ το και φόρτωσέ το στο browser. Αν το πρόβλημα δεν έχει διορθωθεί έλεγξε να βεβαιωθείς ότι μετακινήθηκε το νέο αρχείο class στο directory της τάξης και ότι μετατράπηκε το αρχείο html σε Mondrian2.

Έχουμε εισάγει μια νέα γραμμή στον κώδικα, την drawRect που είναι μία μέθοδος στα γραφικά τάξης. Η γραμμή g.drawRect(0, 0,height-1, width-1) οδηγεί το γραφικό τάξης g να ζωγραφίσει ένα ορθογώνιο ξεκινώντας από το σημείο (0,0) και τελειώνοντας στο σημείο (299,299).

Αυτό το συγκεκριμένο ορθογώνιο περιστοιχίζει ολόκληρο το ορατό τμήμα του applet. Δεν μας εμποδίζει τίποτα να ζωγραφίσουμε έξω από το applet, στην πραγματικότητα το κάναμε αυτό παραπάνω όταν επεκτείναμε το ορθογώνιο σε (300,300). Οτιδήποτε, όμως, ζωγραφίσουμε εκεί δεν θα είναι ορατό από το χρήστη.

Η μέθοδος drawRect ζωγραφίζει ένα ανοικτό ορθογώνιο. Αν θέλουμε να ζωγραφίσουμε ένα γεμάτο ορθογώνιο χρησιμοποιούμε τη μέθοδο fillRect.Στο Mondrian3 θα σχεδιάσουμε ένα γεμάτο ορθογώνιο στο κέντρο του applet. Ακολουθεί ο κώδικας:

```
//Draw a rectangle

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

public class Mondrian3 extends Applet {

  int AppletHeight;
  int AppletWidth;
  int RectHeight;
  int RectWidth;
  int RectTop;
  int RectLeft;


  public void init() {

    Dimension d = size();
    AppletHeight = d.height;
    AppletWidth = d.width;
    RectHeight = AppletHeight/3;
    RectWidth = AppletWidth/3;
    RectTop = (AppletHeight - RectHeight)/2;
    RectLeft= (AppletWidth - RectWidth)/2;
    repaint();

  }

  public void paint(Graphics g) {

    g.drawRect(0, 0, AppletWidth-1, AppletHeight-1);
    g.fillRect(RectLeft, RectTop, RectWidth-1, RectHeight-1);

  }

}
```

Το τελευταίο παράδειγμα αποδεικνύει και κάτι άλλο. Μέχρι τώρα περνούσαμε 2 σημεία στις μεθόδους drawRect και fillRect και ζωγραφίζαμε το ορθογώνιο που τα ενώνει .Έτσι

υλοποιούνται τα ορθογώνια στο QuickDraw. Όμως, αν ίσχυε αυτό το προηγούμενο ορθογώνιο θα είχε σχεδιαστεί ανάμεσα στο (100,100) και (100,100), δηλαδή ένα πολύ μικρό ορθογώνιο. Από τη στιγμή που δεν ισχύει αυτό η σχέση των δυο τελευταίων μεταβλητών (του πλάτους και του ύψους) πρέπει να είναι σωστή.

Ο υπερβολικά έξυπνος αναγνώστης ίσως διαμαρτυρηθεί σ' αυτό το σημείο. Μέχρι τώρα έχουμε ζωγραφίσει μόνο τετράγωνα. Αν και οι τελευταίες δυο μεταβλητές που πέρασαν στο drawRect και fillRect ήταν το ύψος και το πλάτος πώς ξέρουμε ποιο πέρασε σε ποιο. Ο πιο απλός τρόπος να απαντήσεις σ' αυτό είναι να γράψεις ένα δοκιμαστικό πρόγραμμα, που σχεδιάζει ένα μη-τετραγωνισμένο ορθογώνιο. Ας το δοκιμάσουμε τώρα.

```
//Draw a rectangle

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

public class Mondrian4 extends Applet {

  int RectHeight, RectWidth, RectTop, RectLeft, AppletWidth,
AppletHeight;

  public void init() {

    Dimension d = size();
    AppletHeight = d.height;
    AppletWidth = d.width;
    RectHeight = AppletHeight/3;
    RectWidth = (AppletWidth*2)/3;
    RectTop = (AppletHeight - RectHeight)/2;
    RectLeft= (AppletWidth - RectWidth)/2;

    repaint();

  }

  public void paint(Graphics g) {

    g.drawRect(0, 0, AppletWidth-1, AppletHeight-1);
    g.fillRect(RectLeft, RectTop, RectWidth-1, RectHeight-1);

  }

}
```

Βλέπεις, λοιπόν, ότι το τρίτο όρισμα είναι το πλάτος και το τέταρτο το ύψος.

Τώρα που έχουμε μάθει πως να ζωγραφίζουμε ορθογώνια, γεμάτα και άδεια, ας κάνουμε τη ζωή μας λίγο πιο συναρπαστική, επιλέγοντας τυχαία τη θέση και το μέγεθος του ορθογωνίου. Για να το κάνουμε αυτό θα χρειαστούμε τη μέθοδο Math.random() από το java.lang.Math. Αυτή η μέθοδος επιστρέφει ένα double ανάμεσα στο 0.0 και στο 1.0, έτσι θα χρειαστεί να πολλαπλασιάσουμε το αποτέλεσμα με το ύψος και το πλάτος του applet,

για να πάρουμε ένα λογικού μεγέθους ορθογώνιο που ταιριάζει στο χώρο του applet μας. Για να το κάνουμε αυτό θα δημιουργήσουμε την ακόλουθη Randomize μέθοδο.

```
private int Randomize( int range )

{

double rawResult;

rawResult = Math.random();

return (int) (rawResult * range);

}
```

Αυτή η μέθοδος μετατρέπει το αποτέλεσμα του Math.random σε έναν int στην κλίμακα που χρειαζόμαστε. Πρόσεξε ιδιαίτερα την τελευταία γραμμή. Όταν δεις έναν γυμνό τύπο σε παρενθέσεις όπως (int) ή (float) αυτό είναι ένα cast. Τα casts αλλάζουν τον τύπο μιας τιμής σε έναν άλλο. Εδώ αλλάζουμε έναν double σε έναν int.

Τα casting στην Java είναι ασφαλέστερα από τα casting στη C ή άλλες γλώσσες που επιτρέπουν αυθαίρετα casting. Η Java επιτρέπει τα casts να συμβούν, όταν έχουν νόημα, όπως ένα cast ανάμεσα σε έναν float και έναν int. Όμως π.χ. δεν μπορούμε να έχουμε cast ανάμεσα σε έναν int και ένα string

```
//Draw a rectangle

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

public class Mondrian5 extends Applet {

  int RectHeight, RectWidth, RectTop, RectLeft, AppletWidth,
AppletHeight;

  public void init() {

    Dimension d = size();
    AppletHeight = d.height;
    AppletWidth = d.width;
    RectTop = Randomize(AppletHeight);
    RectLeft= Randomize(AppletWidth);
    RectHeight = Randomize(AppletHeight - RectTop);
    RectWidth = Randomize(AppletWidth - RectLeft);

    repaint();

  }
```

```
  public void paint(Graphics g) {

    g.drawRect(0, 0, AppletWidth-1, AppletHeight-1);
    g.fillRect(RectLeft, RectTop, RectWidth-1, RectHeight-1);

  }


  private int Randomize(int range)
  {
    double  rawResult;

    rawResult = Math.random();
    return (int) (rawResult * range);

  }

}
```

Αυτό το applet παράγει τυχαία ένα ορθογώνιο που είναι πολύ μικρό, για να είναι ορατό. Έτσι, αν δεν βλέπεις τίποτα, ξαναφόρτωσέ το. Φόρτωσέ το ξανά πολλές φορές. Κάθε φορά θα βλέπεις να εμφανίζεται ένα ορθογώνιο διαφορετικού μεγέθους. Ας κάνουμε τον κόσμο μας λίγο πιο πολύχρωμο. Θα αλλάξουμε το χρώμα του ορθογωνίου σε κόκκινο. Για να το κάνουμε αυτό θα χρησιμοποιήσουμε μια νέα μέθοδο τη setColor().

```
//Draw a colored rectangle

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

public class Mondrian6 extends Applet {

  int RectHeight, RectWidth, RectTop, RectLeft, AppletWidth,
AppletHeight;

  public void init() {

    Dimension d = size();
    AppletHeight = d.height;
    AppletWidth = d.width;
    RectTop = Randomize(AppletHeight);
    RectLeft= Randomize(AppletWidth);
    RectHeight = Randomize(AppletHeight - RectTop);
    RectWidth = Randomize(AppletWidth - RectLeft);

    repaint();

  }

  public void paint(Graphics g) {

   // g.setBackground(Color.white);

    g.setColor(Color.red);
    g.drawRect(0, 0, AppletWidth-1, AppletHeight-1);
```

```
    g.fillRect(RectLeft, RectTop, RectWidth-1, RectHeight-1);

  }


  private int Randomize(int range)
  {
    double  rawResult;

    rawResult = Math.random();
    return (int) (rawResult * range);

  }

}
```

Το awt ορίζει έναν αριθμό από χρώματα συμπεριλαμβανομένων των παρακάτω:

- μαύρο
- μπλε
- κυανό
- σκούρο γκρι
- πράσινο
- ανοιχτό γκρι
- ποροκαλί
- ροζ
- κόκκινο
- άσπρο
- κίτρινο

Αν αυτά δεν σου αρκούν μπορείς να ορίσεις άλλα χρησιμοποιώντας το τριπλό RGB που χρησιμοποιείται για να ορίσει background χρώμα σε πολλές σελίδες του web. Ακόμη μπορείς να χρησιμοποιήσεις δεκαδικούς αριθμούς αντί για τους δεκαεξαδικούς που πρέπει να χρησιμοποιήσεις για την ετικέτα bgcolor. Για παράδειγμα για να επιλέξεις ένα μεσαίο γκρι θα χρησιμοποιήσεις color(127,127,127). Για το λευκό θα είναι color(255,255,255) ενώ για το κόκκινο είναι color(255,0,0) κ.ο.κ. Χρησιμοποιώντας τον κατασκευαστή χρωμάτων μπορούμε να επεκτείνουμε το πρόγραμμά μας, ώστε να επιλέγει όχι μόνο ένα τυχαίο ορθογώνιο, αλλά επίσης ένα τυχαίο χρώμα. Εδώ είναι ο κώδικας:

```
//Draw a randomly colored rectangle

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

public class Mondrian7 extends Applet {

  int RectHeight, RectWidth, RectTop, RectLeft, AppletWidth,
AppletHeight;
  Color RectColor;
```

```
  public void init() {

    Dimension d = size();
    AppletHeight = d.height;
    AppletWidth = d.width;
    RectTop = Randomize(AppletHeight);
    RectLeft= Randomize(AppletWidth);
    RectHeight = Randomize(AppletHeight - RectTop);
    RectWidth = Randomize(AppletWidth - RectLeft);
    RectColor = new
Color(Randomize(255),Randomize(255),Randomize(255));

    repaint();

  }

  public void paint(Graphics g) {

   // g.setBackground(Color.white);
    g.setColor(RectColor);
    g.drawRect(0, 0, AppletWidth-1, AppletHeight-1);
    g.fillRect(RectLeft, RectTop, RectWidth-1, RectHeight-1);

  }


  private int Randomize(int range)
  {
    double  rawResult;

    rawResult = Math.random();
    return (int) (rawResult * range);

  }

}
```

Στο επόμενο παράδειγμα πρόκειται να ζωγραφίσουμε ορθογώνια τυχαίου μεγέθους και χρώματος. Εφόσον θέλουμε κάθε ορθογώνιο να είναι διαφορετικό ακολουθεί ο κώδικας:

```
//Draw many randomly colored rectangles

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

public class Mondrian8 extends Applet {

  int RectHeight, RectWidth, RectTop, RectLeft, AppletWidth,
AppletHeight;
  Color RectColor;
  int numberRectangles = 100;

  public void init() {

    Dimension d = size();
    AppletHeight = d.height;
```

```
    AppletWidth = d.width;

    repaint();

  }

  public void paint(Graphics g) {

    g.setColor(Color.black);
    g.drawRect(0, 0, AppletWidth-1, AppletHeight-1);

    for (int i=0; i < numberRectangles; i++)  {
      RectTop = Randomize(AppletHeight);
      RectLeft= Randomize(AppletWidth);
      RectHeight = Randomize(AppletHeight - RectTop);
      RectWidth = Randomize(AppletWidth - RectLeft);
      RectColor = new
Color(Randomize(255),Randomize(255),Randomize(255));
      g.setColor(RectColor);
      g.fillRect(RectLeft, RectTop, RectWidth-1, RectHeight-1);
  }

 }


  private int Randomize(int range)
  {
    double  rawResult;

    rawResult = Math.random();
    return (int) (rawResult * range);

  }

}
```

Τέλος ας αφήσουμε το html να καθορίσει τον αριθμό των ορθογωνίων να ορίζονται με ένα πέρασμα. Θα διατηρήσουμε την εξ' ορισμού τιμή και θα την αντικαταστήσουμε μόνο αν το html συμπεριλαμβάνει έναν αριθμό Param.

```
//Draw many random rectangles

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

public class Mondrian9 extends Applet {

  int RectHeight, RectWidth, RectTop, RectLeft, AppletWidth,
AppletHeight;
  Color RectColor;
  int numberRectangles = 100;

  public void init() {

    Dimension d = size();
    AppletHeight = d.height;
```

```
    AppletWidth = d.width;
    String s = getParameter("Number");
    if (s != null) {
      numberRectangles = Integer.valueOf(s).intValue();
  }

    repaint();

  }

  public void paint(Graphics g) {

    g.setColor(Color.black);
    g.drawRect(0, 0, AppletWidth-1, AppletHeight-1);

    for (int i=0; i < numberRectangles; i++)  {
      RectTop = Randomize(AppletHeight);
      RectLeft= Randomize(AppletWidth);
      RectHeight = Randomize(AppletHeight - RectTop);
      RectWidth = Randomize(AppletWidth - RectLeft);
      RectColor = new
Color(Randomize(255),Randomize(255),Randomize(255));
      g.setColor(RectColor);
      g.fillRect(RectLeft, RectTop, RectWidth-1, RectHeight-1);
  }

 }


  private int Randomize(int range)
  {
    double  rawResult;

    rawResult = Math.random();
    return (int) (rawResult * range);

  }

}
```

Αυτά προς το παρόν, αλλά θα επιστρέψουμε στο Mondrian στο τέλος αυτού του κεφαλαίου, όταν προσθέτουμε threading.

Σχεδιάζοντας γραμμές

Το awt περιλαμβάνει αρκετές γραφικές λειτουργίες. Μια απ΄αυτές είναι τα ορθογώνια με τα οποία ασχοληθήκαμε στο προηγούμενο κεφάλαιο. Οι γραμμές είναι μια άλλη. Μέσα σε ένα γραφικό κέιμενο υπάρχει μια μέθοδος για να ζωγραφίζουμε γραμμές, η drawLine, drawLine(int x1, int y1, int x2, int y2). Αυτή η μέθοδος ζωγραφίζει μια ίσια γραμμή ανάμεσα στο σημείο (x1, y1) και στο σημείο (x2, y2). Εδώ είναι ένα απλό applet που ζωγραφίζει μια διαγώνια γραμμή κατά μήκος του applet frame:

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

public class SimpleLine extends Applet {

  int AppletHeight, AppletWidth;

  public void init() {
    Dimension d = size();
    AppletHeight = d.height;
    AppletWidth = d.width;
  }

 public void paint(Graphics g) {

    g.drawLine(0, 0, AppletWidth, AppletHeight);

 }

}
```

Πρόκειται να αποδείξουμε τη χρήση της μεθόδου drawLine() και να ζωγραφίσουμε διακριτικά άνισες φιγούρες. Αρχίζουμε με το σκελετό του applet.Θα χρειαστεί να προσθέσουμε λίγο κώδικα στη μέθοδο paint του applet για να το κάνουμε να ζωγραφίσει κάτι. Ας αρχίσουμε ζωγραφίζοντας ένα λαμπερό κύμα από την αριστερή πλευρά της εικόνας στην δεξιά. Εδώ είναι το ολοκληρωμένο πρόγραμμα.

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

public class GraphApplet extends Applet {

  int x0, xN, y0, yN;

  public void init() {
    // How big is the applet?
    Dimension d = size();

    x0 = 0;
    xN = d.width-1;
    y0=0;
    yN=d.height-1;
  }

  public void paint(Graphics g) {

    for (int x = x0; x < xN; x++) {
      g.drawLine(x,(int) (yN*Math.sin(x)),x+1, (int)
(yN*Math.sin(x+1)));
    }

  }

}
```

Η ουσία αυτού του applet βρίσκεται στο for loop της μεθόδου paint.

```
for (int x = x0; x < xN; x++) {

g.drawLine(x,

(int)(yN*Math.sin(x)),x+1,

(int) (yN*Math.sin(x+1)));

}
```

Εδώ έχουμε loop κατά μήκος κάθε x pixel του appet. Σε κάθε ένα υπολογίζουμε τη φωτεινότητα από τα pixel. Επίσης υπολογίζουμε τη φωτεινότητα του επόμενου pixel. Αυτό μας δίνει δύο σημεία και ζωγραφίζουμε μια γραμμή αναμεσά τους. Αφού η φωτεινότητα ενός πραγματικού αριθμού είναι πάντα ανάμεσα στο 1 και στο -1, μετράμε την τιμή του y με yN. Τελικά μετατρέπουμε τις τιμές y σε ακέραιο, αφού η φωτεινότητα παίρνει πραγματικές τιμές ενώ τα drawLine παίρνουν ακέραιες τιμές.

Αυτό το applet τρέχει, αλλά έχει πολλά προβλήματα. Όλα αυτά μπορούν να σχετιστούν με δυο παράγοντες.

Η φωτεινότητα είναι λειτουργία κινητής υποδιαστολής. Για να κάνουμε ένα πραγματικό γραφικό applet πρέπει να είμαστε ικανοί να χρησιμοποιούμε αριθμούς κινητής υποδιαστολής. Το σύστημα συντεταγμένων ενός applet ξεκινά από την πάνω αριστερή γωνία και προχωρά προς τα κάτω και δεξιά.

Υπάρχουν πολλοί τρόποι για να δώσουμε λύσεις σ' αυτό. Η λύση για όλα αυτά, όμως, είναι να διαχωρίσουμε τα δεδομένα από την έκθεση. Αφού σχεδιάζουμε περισσότερο ή λιγότερο καλά συμπεριφερόμενες μαθηματικές συναρτήσεις μπορούμε να θεωρήσουμεε ότι τα δεδομένα μας περιγράφονται εξ' ολοκλήρου από ένα ορθογώνιο στον καρτεσιανό χώρο , το οποίο ευχόμαστε να σχεδιάσει μια συνάρτηση. Η έκθεση, από την άλλη πλευρά, περιγράφεται με ένα ορθογώνιο από διακριτικά σημεία καθορισμένου μήκους και πλάτους. Χρειάζεται να είμαστε ικανοί να υπολογίζουμε στο γενικό καρτεσιανό χώρο και να εκθέτουμε στο συγκεκριμένο παράθυρο του applet. Θα χρειαστούμε μια μέθοδο που θα μετατρέπει ένα σημείο του παράθυρου σ' ένα άλλο σημείο στον καρτεσιανό χώρο και ένα άλλο που θα το μετατρέπει ξανά πίσω.

Ακολουθεί ο κώδικας:

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

public class GraphApplet extends Applet {

  int x0, xN, y0, yN;
  double xmin, xmax, ymin, ymax;
  int AppletHeight, AppletWidth;
```

```
  public void init() {
    // How big is the applet?
    Dimension d = size();
    AppletHeight = d.height;
    AppletWidth  = d.width;
    x0 = 0;
    xN = AppletWidth-1;
    y0=0;
    yN=AppletHeight-1;
    xmin = -10.0;
    xmax =  10.0;
    ymin = -1.0;
    ymax =  1.0;
  }

  public void paint(Graphics g) {

    double x1,y1,x2,y2;
    int i, j1, j2;

    j1 = yvalue(0);
    for (i = 0; i < AppletWidth; i++) {
      j2 = yvalue(i+1);
      g.drawLine(i, j1 ,i+1, j2);
      j1 = j2;

    }

  }

  private int yvalue(int ivalue)  {

    // Given the  xpoint we're given calculate the Cartesian equivalent
    double x, y;
    int jvalue;

    x = (ivalue * (xmax - xmin)/(AppletWidth - 1)) + xmin;

    // Take the sine of that x
    y = Math.sin(x);

    // Scale y into window coordinates
    jvalue = (int) ((y - ymin)*(AppletHeight - 1)/(ymax - ymin));

    // Switch jvalue from cartesian coordinates to computer graphics
coordinates
    jvalue = AppletHeight - jvalue;

    return jvalue;

  }



}
```

Τρέξε αυτό το applet. Η φωτεινότητα του κύματος δεν είναι τώρα πολύ πιο εμφανής; Υπάρχουν ακόμα μερικά πράγματα που μπορούμε να προσθέσουμε για να κάνουμε το applet πιο πολύπλοκο. Το πιο σημαντικό θα είναι να προσθέσουμε κάποιους παράγοντες έτσι που να μπορέσουμε να ορίσουμε το μέγεθος του applet σε html. Η ακόλουθη τροποποίηση των μεθόδων init και paint ζητά να καθορίσει μέσω παραμέτρων τις xmin, ymin και ymax. Παραθέτουμε κάποιες εξ' ορισμού λογικές τιμές σε περίπτωση που ο συγγραφέας του html ξεχάσει να τα εξειδικεύσει.

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

public class GraphApplet extends Applet {

  int x0, xN, y0, yN;
  double xmin, xmax, ymin, ymax;
  int AppletHeight, AppletWidth;

  public void init() {
    String ParamString;

    // How big is the applet?
    Dimension d = size();
    AppletHeight = d.height;
    AppletWidth  = d.width;
    x0 = 0;
    xN = AppletWidth-1;
    y0=0;
    yN=AppletHeight-1;

    ParamString = getParameter("xmin");
    if (ParamString != null) {
      xmin = Double.valueOf(ParamString).doubleValue();
    }
    else {
      xmin = -1.0;
    }
    ParamString = getParameter("xmax");
    if (ParamString != null) {
      xmax = Double.valueOf(ParamString).doubleValue();
    }
    else {
      xmax = 1.0;
    }
    ParamString = getParameter("ymax");
    if (ParamString != null) {
      ymax = Double.valueOf(ParamString).doubleValue();
    }
    else {
      ymax = 1.0;
    }
    ParamString = getParameter("ymin");
    if (ParamString != null) {
      ymin = Double.valueOf(ParamString).doubleValue();
    }
    else {
```

```
      ymin = -1.0;
    }
  }

  public void paint(Graphics g) {

    double x1,y1,x2,y2;
    int i, j1, j2;

    j1 = yvalue(0);
    for (i = 0; i < AppletWidth; i++) {
      j2 = yvalue(i+1);
      g.drawLine(i, j1 ,i+1, j2);
      j1 = j2;
    }

  }

  private int yvalue(int ivalue)  {

    // Given the  xpoint we're given calculate the Cartesian equivalent
    double x, y;
    int jvalue;

    x = (ivalue * (xmax - xmin)/(AppletWidth - 1)) + xmin;

    // Take the sine of that x
    y = Math.sin(x);

    // Scale y into window coordinates
    jvalue = (int) ((y - ymin)*(AppletHeight - 1)/(ymax - ymin));

    // Switch jvalue from cartesian coordinates to computer graphics
coordinates
    jvalue = AppletHeight - jvalue;

    return jvalue;

  }

}
```

Ασκήσεις:

1.Πρόσθεσε σύστημα συντεταγμένων στο γράφημα.

2.Η μέθοδος του γραφήματός μας χρησιμοποιεί μαθηματικές συναρτήσεις. Τι πρέπει να αλλάξεις και ποια χαρακτηριστικά πρέπει να προσθέσεις για να το καταστήσεις κατάλληλο να σχεδιάζει πειραματικά δεδομένα;

Ένα άπειρο σύνολο με μηδενικό μήκος.

Τώρα πρόκειται να χρησιμοποιήσουμε τη Java για να θέσουμε σε εφαρμογή μερικά κλασσικά παραδείγματα από τη γεωμετρία. Θα αρχίσουμε με ένα μονοδιάστατο σύνολο με ένα αόριστο αριθμό σημείων, που καλύπτουν μηδενικό μήκος. Έπειτα θα μελετήσουμε το snowflake του Koch. Τέλος, στο επόμενο κεφάλαιο θα ερευνήσουμε το Mandelbrot set.

Το μεσαίο τρίτο σύνολο ορίζεται αρχίζοντας με όλους τους πραγματικούς αριθμούς ανάμεσα στο 0 και στο 1. Έπειτα, αποκόβουμε το τρίτο μισό από το set , π.χ. καθένα ανάμεσα στο 1/3 και στο 2/3. Στη συνέχεια κόβουμε το τρίτο μισό από τα τμήματα 2 γραμμών που απομένουν π.χ. καθένα ανάμεσα στο 1/9 και 2/9 και ανάμεσα στο 7/9 και 8/9. Συνεχίζουμε αυτήν την διαδικασία επ' άπειρον.

Ήταν πολύπλοκο; Καλά. Μια εικόνα αξίζει όσο χίλιες λέξεις και ένα καλό πρόγραμμα σε Java αξίζει όσο χίλιες εικόνες. Τώρα προχωράμε να δείξουμε ένα πρόγραμμα σε Java, που ζωγραφίζει επιτυχώς εικόνες για να επιδείξει το τρίτο μέσο σύνολο.

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

public class MiddleThird extends Applet {

int AppletWidth;
int AppletHeight;

Vector endpoints = new Vector();

  public void init() {
    Dimension d = size();
    AppletHeight = d.height;
    AppletWidth = d.width;
    endpoints.addElement(new Float(0.0f));
    endpoints.addElement(new Float(1.0f));
  }

 public void paint(Graphics g) {

   float x1, x2;
   Float tempFloat;
   for (int i = 0; i <  AppletHeight; i+= 5) {
     // draw the lines
     for (int j=0; j < endpoints.size(); j += 2) {
       tempFloat = (Float) endpoints.elementAt(j);
       x1 = tempFloat.floatValue();
       tempFloat = (Float) endpoints.elementAt(j+1);
       x2 = tempFloat.floatValue();
       g.drawLine( Math.round(x1*AppletWidth), i,
Math.round(x2*AppletWidth), i);
     }
     //remove the middle third of the lines
     CutSegments();
     // Now check to see if we've exceeded the resolution of our screen
     tempFloat = (Float) endpoints.elementAt(0);
```

```
     x1 = tempFloat.floatValue();
     tempFloat = (Float) endpoints.elementAt(1);
     x2 = tempFloat.floatValue();
     if (Math.round(x1*AppletWidth) == Math.round(x2*AppletWidth))
break;
   }


 }

 private void CutSegments() {

   int index = 0;
   float gap;
   float x1, x2;
   Float tempFloat1, tempFloat2;
   int stop = endpoints.size();

   for (int i=0; i < stop; i+=2) {
      CutMiddleThird(index, index+1);
      index += 4;
   }

 }


  private void CutMiddleThird(int left, int right) {

   float gap;
   float x1, x2;
   Float tempFloat1, tempFloat2;

   tempFloat1 = (Float) endpoints.elementAt(left);
   tempFloat2 = (Float) endpoints.elementAt(right);
   gap = tempFloat2.floatValue() - tempFloat1.floatValue();
   x1  = tempFloat1.floatValue() + gap/3.0f;
   x2  = tempFloat2.floatValue() - gap/3.0f;
   endpoints.insertElementAt(new Float(x2), right);
   endpoints.insertElementAt(new Float(x1), right);

 }

}
```

Μεταγλώτισε και φόρτωσε αυτό το applet. Αυτό δεν είναι πιο ξεκάθαρο; Ανάλογα από το πόσο μεγάλο είναι το παράθυρο που δίνεις στο applet, θα δεις περίπου 6 με 12 επαναλήψεις, πριν χρειαστεί να αρχίσουμε να δουλεύουμε με κλασματικούς αριθμούς.

Ιπτάμενες γραμμές.

Το επόμενο παράδειγμα είναι πιο δύσκολο να περιγραφεί, από ότι να γραφεί ο κώδικας. Όπως και το Mondrian τρέχει σ' ένα αόριστο loop, αλλά δεν είναι απλά τυχαίες εικόνες.

Μεταγλώτισε τον ακόλουθο κώδικα, εκτέλεσε το πρόγραμμα και μετά μελέτησε τον κώδικα να δεις αν μπορείς να καταλάβεις τον αλγόριθμο.

```
//Bounce lines around in a box

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

public class FlyingLines extends Applet {

  int NUM_LINES = 25;
  int gDeltaTop=3, gDeltaBottom=3;
  int gDeltaLeft=2, gDeltaRight=6;
  int AppletWidth, AppletHeight;
  int gLines[][] = new int[NUM_LINES][4];

  public void init() {

    AppletWidth = size().width;
    AppletHeight = size().height;

  }

  public void start() {
    gLines[0][0] = Randomize(AppletWidth);
    gLines[0][1] = Randomize(AppletHeight);
    gLines[0][2] = Randomize(AppletWidth);
    gLines[0][3] = Randomize(AppletHeight);
    for (int i=1; i < NUM_LINES; i++ ) {
      LineCopy(i, i-1);
      RecalcLine(i);
    }
    repaint();

  }

  public void paint(Graphics g) {

     while (true) {
      for (int i=NUM_LINES - 1; i > 0; i--) {
        LineCopy(i, i-1);
      }
      RecalcLine(0);
      g.setColor(Color.black);
      g.drawLine(gLines[0][0], gLines[0][1], gLines[0][2],
gLines[0][3]);
      g.setColor(getBackground());
      g.drawLine(gLines[NUM_LINES-1][0], gLines[NUM_LINES-1][1],
        gLines[NUM_LINES-1][2], gLines[NUM_LINES-1][3]);
    }

  }

  private void LineCopy (int to, int from)  {

    for (int i = 0; i < 4; i++) {
```

```
      gLines[to][i] = gLines[from][i];
    }

  }

  public int Randomize( int range ) {
    double  rawResult;

    rawResult = Math.random();
    return (int) (rawResult * range);

  }

  private void RecalcLine( int i ) {

    gLines[i][1] += gDeltaTop;
    if ((gLines[i][1] < 0) || (gLines[i][1] > AppletHeight)) {
      gDeltaTop *= -1;
      gLines[i][1] += 2*gDeltaTop;
    }

    gLines[i][3] += gDeltaBottom;
    if ( (gLines[i][3] < 0) || (gLines[i][3] > AppletHeight) ) {
      gDeltaBottom *= -1;
      gLines[i][3] += 2*gDeltaBottom;
    }

    gLines[i][0] += gDeltaLeft;
    if ( (gLines[i][0] < 0) || (gLines[i][0] > AppletWidth) ) {
      gDeltaLeft *= -1;
      gLines[i][0] += 2*gDeltaLeft;
    }

    gLines[i][2] += gDeltaRight;
    if ( (gLines[i][2] < 0) ||  (gLines[i][2] > AppletWidth) ) {
      gDeltaRight *= -1;
      gLines[i][2] += 2*gDeltaRight;
    }

  }  //RecalcLine ends here

} // FlyingLines ends here
```

Αναλαμβάνοντας Δράση:Νήματα

Ανάλογα με το λειτουργικό σύστημα και τον Java browser, ίσως έχεις παρατηρήσει ότι τα προγράμματα Mondrian και Flying line, έχουν την τάση να ‘’κρεμούν’’. Στα Windows NT η Hot Java σταματά να ανταποκρίνεται μετά από αρκετές χιλιάδες επαναλήψεις εντολών.

Τα loop της ζωγραφικής στο Mondrian και στο Flying Lines είναι ιδανικά για ένα thread, ένα ξεχωριστό stream από λειτουργίες, που λαμβάνουν χώρα ταυτόχρονα και ανεξάρτητα από οτιδήποτε άλλο ίσως συμβεί. Σαν ένας γενικός κανόνας όλες οι εντατικές εργασίες

της CPU πρέπει να τοποθετούνται στα δικά τους theads. Παρακάτω είναι ένας τρόπος για να το κάνεις αυτό:

```
//Draw infinitely many random rectangles

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

public class ThreadedMondrian extends Applet implements Runnable {

  int RectHeight, RectWidth, RectTop, RectLeft, AppletWidth,
AppletHeight;
  Color RectColor;
  Thread kicker = null;
  int pause;

  public void init() {

    Dimension d = size();
    AppletHeight = d.height;
    AppletWidth = d.width;
    repaint();
  }

  public void paint(Graphics g) {

    g.setColor(Color.black);
    g.drawRect(0, 0, AppletWidth-1, AppletHeight-1);

    for (int i=0; i < 10; i++) {
      RandomRect();
      g.setColor(RectColor);
      g.fillRect(RectLeft, RectTop, RectWidth-1, RectHeight-1);
    }

 }

  public void run() {
    Thread.currentThread().setPriority(Thread.MIN_PRIORITY);

    while (true) {  // infinite loop
      repaint();
      try {
        Thread.sleep(100);
       }
       catch (Exception e) {

       }
    }
  }

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

```
  }

  public void stop() {
    kicker = null;
  }

  public void RandomRect()  {
      RectTop   = Randomize(AppletHeight);
      RectLeft  = Randomize(AppletWidth);
      RectHeight= Randomize(AppletHeight - RectTop);
      RectWidth = Randomize(AppletWidth - RectLeft);
      RectColor = new
Color(Randomize(255),Randomize(255),Randomize(255));
  }

  private int Randomize(int range)
  {
    double  rawResult;

    rawResult = Math.random();
    return (int) (rawResult * range);

  }

}
```

Εδώ είναι μια threaded έκδοση του flying lines.

```
//Bounce lines around in a box

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

public class FlyingLines extends Applet implements Runnable {

  int NUM_LINES = 25;
  int gDeltaTop=3, gDeltaBottom=3;
  int gDeltaLeft=2, gDeltaRight=6;
  int AppletWidth, AppletHeight;
  int gLines[][] = new int[NUM_LINES][4];

  public void init() {

    AppletWidth = size().width;
    AppletHeight = size().height;

  }

  public void start() {
    gLines[0][0] = Randomize(AppletWidth);
    gLines[0][1] = Randomize(AppletHeight);
    gLines[0][2] = Randomize(AppletWidth);
    gLines[0][3] = Randomize(AppletHeight);
    for (int i=1; i < NUM_LINES; i++ ) {
      LineCopy(i, i-1);
      RecalcLine(i);
```

```
    }
    repaint();
    Thread t = new Thread(this);
    t.start();

  }

  public void run () {

    Thread.currentThread().setPriority(Thread.MIN_PRIORITY);

    while (true) {
      for (int i=NUM_LINES - 1; i > 0; i--) {
        LineCopy(i, i-1);
      }
      RecalcLine(0);
      System.out.println(gLines[0][0] + ", " + gLines[0][1] + "," +
gLines[0][2] + ", " + gLines[0][3]);
      repaint();
      try {
        Thread.currentThread().sleep(10);
      }
      catch (Exception e) {

      }
    }

  }

  public void paint(Graphics g) {

    for (int i=0; i < NUM_LINES; i++) {
      g.drawLine(gLines[i][0], gLines[i][1], gLines[i][2],
gLines[i][3]);
    }

  }

  private void LineCopy (int to, int from)  {

    for (int i = 0; i < 4; i++) {
      gLines[to][i] = gLines[from][i];
    }

  }

  public int Randomize( int range ) {
    double  rawResult;

    rawResult = Math.random();
    return (int) (rawResult * range);

  }

  private void RecalcLine( int i ) {

    gLines[i][1] += gDeltaTop;
```

```
    if ((gLines[i][1] < 0) || (gLines[i][1] > AppletHeight)) {
      gDeltaTop *= -1;
      gLines[i][1] += 2*gDeltaTop;
    }

    gLines[i][3] += gDeltaBottom;
    if ( (gLines[i][3] < 0) || (gLines[i][3] > AppletHeight) ) {
      gDeltaBottom *= -1;
      gLines[i][3] += 2*gDeltaBottom;
    }

    gLines[i][0] += gDeltaLeft;
    if ( (gLines[i][0] < 0) || (gLines[i][0] > AppletWidth) ) {
      gDeltaLeft *= -1;
      gLines[i][0] += 2*gDeltaLeft;
    }

    gLines[i][2] += gDeltaRight;
    if ( (gLines[i][2] < 0) ||  (gLines[i][2] > AppletWidth) ) {
      gDeltaRight *= -1;
      gLines[i][2] += 2*gDeltaRight;
    }

  }  //RecalcLine ends here

} // FlyingLines ends here
```

Bozo Sort

Ένα applet που πραγματικά κάνει καλή χρήση των νημάτων είναι το bozo sort. Στο bozo sort η ίδια συλλογή από διαφορετικού μεγέθους sticks πετάγονται στον αέρα. Αν προσγειωθούν σε ταξινομημένη σειρά, ο αλγόριθμος σταματά. Διαφορετικά, πετάμε όλα τα sticks ξανά στον αέρα. Ο αλγόριθμος τρέχει σε περίπου 0(N!) χρόνο, όπου N είναι ο αριθμός των sticks. Απαιτεί απαγορευτικό χρόνο για περισσότερο από μια ντουζίνα sticks. Είναι ένας τρομερά ηλίθιος αλγόριθμος, αλλά μια πραγματικά μεγάλη ευκαιρία για threading.

```
class BozoSortAlgorithm extends SortAlgorithm {

  void sort(int a[]) {

    boolean sorted = false;

    while (!sorted) {
      int index1 = Randomize(a.length);
      int index2 = Randomize(a.length);

      int temp = a[index2];
      a[index2] = a[index1];
      a[index1] = temp;
      // Is a[] sorted?
      sorted = true;
      for (int i = 1; i < a.length; i++)  {
        if (a[i-1] > a[i]) {
```

```
          sorted = false;
          break;
        }  // end if
      }  // end for
    } // end while
  }  // end sort

  private int Randomize( int range )  {

    double  rawResult;

    rawResult = Math.random();
    return (int) (rawResult * range);

  }

}  // end BozoSortAlgorithm
```

Αλληλεπίδραση:Είσοδος με το ποντίκι και το πληκτρολόγιο

(Interaction:Mouse and Keyboard Input)

Τώρα έχεις τα εργαλεία για να ζωγραφίσεις πολλές πραγματικά ωραίες κινήσεις και φιγούρες στις σελίδες του web. Αυτό μας τοποθετεί πάνω από το μέσο σχεδιαστή των σελίδων του web. Το άλλο μισό είναι η αλληλεπίδραση με το χρήστη. Τα applet μπορούν να δεχτούν είσοδο από το χρήστη και να ανταποκριθούν σ' αυτόν.

Είσοδος με το ποντίκι: Java Doodle

Εδώ είναι ένα απλό applet που σου επιτρέπει να τραβάς γραμμές με το ποντίκι πάνω σε ένα applet.

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

public class JavaDoodle extends Applet {

  Vector points = new Vector();


  public void paint(Graphics g) {

        int x1, y1, x2, y2;
        Point tempPoint;

    if (points.size() > 1) {
      tempPoint = (Point) points.elementAt(0);
          x1 = tempPoint.x;
          y1 = tempPoint.y;
      for (int i = 1; i < points.size(); i++) {
        tempPoint = (Point) points.elementAt(i);
                x2 = tempPoint.x;
```

```
                y2 = tempPoint.y;
          g.drawLine(x1, y1, x2, y2);
          x1 = x2;
          y1 = y2;
        } // end for
      } // end if
    }

    public boolean mouseDown(Event e, int x, int y) {
      points.addElement(new Point(x, y));
          return true;
    }

    public boolean mouseDrag(Event e, int x, int y) {
      points.addElement(new Point(x, y));
      repaint();
          return true;
    }

    public boolean mouseUp(Event e, int x, int y) {
      points.addElement(new Point(x, y));
      repaint();
          return true;
    }



}
```

Είσοδος με το πληκτρολόγιο(Keyboard Input): TypeWriter

Εδώ είναι ένα απλό applet που χρησιμοποεί την μέθοδο KeyDown για να πληκτρολογήσει κάποιο κείμενο.

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

public class typewriter extends Applet {

  int numcols = 80;
  int numrows = 25;
  int row = 0;
  int col = 0;
  char page[][] = new char[numrows][];

  public void init() {

        for (int i = 0; i < numrows; i++) {
      page[i] = new char[numcols];
    }
    for (int i = 0; i < numrows; i++) {
        for (int j = 0; j < numcols; j++) {
          page[i][j] = '\0';
        }
```

```
  }

}


public boolean keyDown(Event e, int key) {

  char c = (char) key;

  switch (key) {
    case Event.HOME:
      row = 0;
      col = 0;
      break;
    case Event.END:
      row = numrows-1;
      col = numcols-1;
      break;
    case Event.UP:
      if (row > 0) row--;
      break;
    case Event.DOWN:
      if (row < numrows-1) row++;
      break;
    case Event.LEFT:
      if (col > 0) col--;
      else if (col == 0 && row > 0) {
        row--;
        col=numcols-1;
      }
      break;
    case Event.RIGHT:
      if (col < numcols-1) col++;
      else if (col == numcols-1 &&  row < numrows-1) {
        row++;
        col=0;
      }
      break;
    default:
      if (c == '\n' || c == '\r') {
        row++;
        col = 0;
      }
      else if (row < numrows) {
        if (col >= numcols) {
          col = 0;
          row++;
        }
        page[row][col] = c;
        col++;
      }
          else { // row >= numrows
           col++;
          }
       }
       repaint();
       return true;
```

```
  }


  public void paint(Graphics g) {

        for (int i=0; i < numrows; i++) {
          String tempString = new String(page[i]);
      g.drawString(tempString, 5, 15*(i+1));
    }

  }

}
```

ΜΕΡΟΣ 4: Αντικείμενα, Κλάσεις, Μέθοδοι και Διεπαφές

Ο αντικειμενοστρεφής προγραμματισμός (εν συντομία OOP) περιλαμβάνει όλα τα χαρακτηριστικά του δομημένου προγραμματισμού καθώς και ισχυρούς τρόπους οργάνωσης αλγορίθμων και δομών δεδομένων. Οι γλώσσες OOP έχουν τρία χαρακτηριστικά: ενθυλάκωση, πολυμορφία και κληρονομικότητα.

Κλάσεις και Αντικείμενα

Το πρωταρχικό χαρακτηριστικό των γλωσσών OOP είναι η κλάση (class). Η κλάση είναι μια δομή δεδομένων που μπορεί να συσχετιστεί με μεθόδους που ενεργούν πάνω σε ένα αντικείμενο με το αντικείμενο αυτό καθ' εαυτό. Στις πριν-OOP γλώσσες μέθοδοι και δεδομένα ήταν ξεχωριστά. Στις OOP γλώσσες όλα αυτά είναι μέρη μιας κλάσης. Οι γλώσσες προγραμματισμού παρέχουν ορισμένους απλούς τύπους δεδομένων όπως int,float και string. Όμως, πολύ συχνά τα δεδομένα δεν είναι τόσο απλά όπως ints, floats ή strings. Οι κλάσεις επιτρέπουν στους προγραμματιστές να ορίσουν δικούς τους, πιο πολύπλοκους τύπους δεδομένων. Για παράδειγμα ας υποθέσουμε ότι το πρόγραμμά σου χρειάζεται να κρατάει μια βάση δεδομένων του site. Για κάθε site έχεις ένα όνομα, ένα URL και μια περιγραφή. Στις παραδοσιακές γλώσσες προγραμμαισμού θα έχει τρεις διαφορετικές μεταβλητές string για κάθε site του WEB. Με μια κλάση συνδυάζεις όλα αυτά σε ένα πακέτο, όπως παρακάτω:

```
class website {

  String name;
  String url;
  String description;

}
```

Αυτές οι μεταβλητές (όνομα, url και περιγραφή) ονομάζονται μέλη της κλάσης. Σου λένε τι είναι μια κλάση και ποιες είναι οι ιδιότητές της. Στη βάση δεδομένων του site του web

θα έχουμε πολλές χιλιάδες websites. Κάθε ξεχωριστό web site είναι ένα αντικείμενο. Μια κλάση ορίζει τι είναι ένα αντικέιμενο, αλλά δεν είναι αυτό καθ' εαυτό ένα αντικείμενο. Ένα αντικείμενο είναι ένα ιδιαίτερο στιγμιότυπο της κλάσης. Όμως, όταν δημιουργούμε ένα νέο αντικείμενο λέμε ότι δημιουργούμε ένα στιγμιότυπο της κλάσης. Κάθε κλάση υπάρχει μόνο μια φορά σε ένα πρόγραμμα, αλλά υπάρχουν πολλές χιλιάδες από αντικείμενα που είναι στιγμιότυπα κλάσεως.

Για να δημιουργήσουμε ένα αντικείμενο στη Java χρησιμοποιούμε τη λειτουργία new. Παρακάτω βλέπουμε πως θα δημιουργήσουμε το καινούργιο web site.

website x = new website();

Καθώς έχουμε ένα website θέλουμε να μάθουμε κάτι γι' αυτό. Για να παρουμε τις μεταβλητές μέλους του website μπορούμε να χρησιμοποιήσουμε τον τελεστή '.'. Το website έχει τρεις μεταβλητές μέλους: όνομα,url και περιγραφή. Έτσι το x έχει τρεις μεταβλητές μέλους τις x.name, x.url, x.description. Μπορούμε να το χρησιμοποιήσουμε ακριβώς όπως θα χρησιμοποιούσαμε οποιαδήποτε άλλη μεταβλητή string.

Για παράδειγμα:

```
website x = new website();

x.name = "Cafe Au Lait";

x.url = "http://metalab.unc.edu/javafaq/";

x.description = "Really cool!";

System.out.println(x.name + " at " + x.url + " is " + x.description);
```

Μέθοδοι (Methods)

Οι τύποι δεδομένων δεν είναι πολύ εύχρηστοι αν δεν μπορείς να κάνεις πράγματα με αυτούς, Γι' αυτό το σκοπό οι κλάσεις έχουν μεθόδους. Τα μέλη λένε τι είναι μια κλάση. Οι μέθοδοι λένε τι κάνει μια κλάση. Π.χ. η κλάση του website ίσως έχει μια μέθοδο για να τυπώσει τα δεδομένα του. Εάν είναι έτσι αυτό θα μοιάζει όπως παρακάτω:

```
class website {

  String name;
  String url;
  String description;

  print() {
    System.out.println(name + " at " + url + " is " + description);
  }
```

```
}
```

Έξω από τη μέθοδο website καλούμε τη μέθοδο print, ακριβώς όπως αναφερόμασταν στις μεταβλητές μέλους, χρησιμοποιώντας το όνομα του συγκεκριμένου αντικειμένου που θέλουμε να τυπώσουμε και τον τελεστή '.'.

```
    website x = new website();

    x.name = "Cafe Au Lait";
    x.url = "http://metalab.unc.edu/javafaq/";
    x.description = "Really cool!";

    x.print();
```

Παρατήρησε ότι μέσα στην κλάση του website δεν χρειάζεται να χρησιμοποιούμε x.name ή x.url, name και url είναι αρκετά. Αυτό ισχύει γιατί η μέθοδος print πρέπει να καλείται από ένα συγκεκριμένο στιγμιότυπο κλάσης του website.

Η μέθοδος print() έχει ολοκληρωτικά ενθυλακωθεί μέσα στην κλάση του website. Όλοι οι μέθοδοι στη Java πρέπει να ανήκουν σε μια κλάση.

Κατασκευαστής (Constructors)

Η πρώτη μέθοδος που χρειάζονται οι περισσότερες κλάσεις είναι ο κατασκευαστής. Μια δομή δημιουργεί ένα νέο στιγμιότυπο της κλάσης. Αρχικοποιεί όλες τις μεταβλητές και εκτελεί οποιαδήποτε ενέργεια χρειάζεται για να προετοιμαστεί η κλάση και να χρησιμοποιηθεί. Στη γραμμή website x = new website(); η website( ) είναι μια συνάρτηση κατασκευής. Αν δεν υπάρχει τέτοια συνάρτηση η Java δημιουργεί εξ' ορισμού μια, αλλά είναι καλύτερο να είσαι σίγουρος ότι έχεις ορίσει τη δική σου. Ο κατασκευαστής γράφει μια public μέθοδο, η οποία έχει το ίδιο όνομα με την κλάση. Έτσι, η δομή του website μας καλείται website(). Εδώ είναι μια απλή website κλάση με μια δομη που αρχικοποιεί όλα τα μέλη με μηδένικά strings.

```
class website {

  String name;
  String url;
  String description;

  public website() {
    name = "";
    url  = "";
    description = "";
  }

}
```

Ακόμα καλύτερα πρέπει να δημιουργήσουμε μια δομή που να δέχεται τρεία strings σαν ορίσματα και να τα χρησιμοποιεί για να αρχικοποιήσει τις μεταβλητές μέλους ως εξής:

```
class website {

  String name;
  String url;
  String description;

  public website(String n, String u, String d) {
    name = n;
    url  = u;
    description = d;
  }

}
```

θα το χρησιμοποιήσουμε ως εξής:

```
    website x = new website("Cafe Au Lait",
"http://metalab.unc.edu/javafaq/", "Really cool!");
    x.print();
```

Αυτό ταιριάζει με την προθέσή μας να κρατάμε τον κώδικα σχετικό με την κατάλληλη λειτουργία της κλάση μέσα σε μια τάξη.

Όμως τι συμβαίνει όταν θέλουμε να δημιουργήσουμε ένα website του οποίου μερικές φορές ξέρουμε το url,το όνομα και την περιγραφή και μερικές φορές δεν το ξέρουμε; Καλύτερα ας τα χρησιμοποιήσουμε και τα δυο:

```
class website {

  String name;
  String url;
  String description;

  public website(String n, String u, String d) {
    name = n;
    url  = u;
    description = d;
  }

  public website() {
    name = "";
    url  = "";
    description = "";
  }

}
```

Αυτό καλείται μέθοδος υπερφόρτωσης ή πολυμορφισμού. Ο πολυμορφισμός είναι ένα χαρακτηριστικό των αντικειμενοστρεφών γλωσσών που αφήνουν ένα όνομα να αναφέρεται σε διαφορετικές μεθόδους,εξαρτώμενο από το περιεχόμενό τους. Το

σημαντικό στο περιεχόμενο είναι ο τύπος και ο αριθμός των ορισμάτων της μεθόδου. Σ'αυτήν την περίπτωση χρησιμοποιούμε την πρώτη έκδοση της μεθόδου αν έχουμε τρεία ορίσματα και την δεύτερη έκδοση αν δεν έχουμε καθόλου ορίσματα. Αν έχεις ένα, δύο ή τέσσερα ορίσματα string στη δομή, ή ορίσματα που δεν είναι string, ο compiler δημιουργεί ένα λάθος γιατί δεν έχει μια μέθοδο που να ταιριάζει με τη μέθοδο που ζητήθηκε.

Μέθοδοι toStrings

Η μέθοδος print είναι κοινή σε κάποιες γλώσσες, αλλά τα περισσότερα προγράμματα της Java την χειρίζονται διαφορετικά. Μπορείς να χρησιμοποιήσεις System.out.println() για να τυπώσεις κάποιο αντικείμενο. Όμως για καλά αποτελέσματα η classs σου πρέπει να έχει μια μέθοδο toString(), που να μορφοποιεί τα δεδομένα των αντικειμένων με ένα λογικό τρόπο και να επιστρέφει ένα string. Παρακάτω βλέπουμε πως χρησιμοποιείται αυτό στο παράδειγμα του website:

```
public class ClassTest {

  public static void main(String args[]) {

    website x = new website("Cafe Au Lait",
"http://metalab.unc.edu/javafaq/", "Really cool!");
    System.out.println(x);

  }

}


class website {

  String name;
  String url;
  String description;

  public website(String n, String u, String d) {
    name = n;
    url  = u;
    description = d;
  }

  public website() {
    name = "";
    url  = "";
    description = "";
  }

  public String toString() {
    return  (name + " at " + url + " is " + description);
  }
```

```
}
```

Ένα σημαντικό παράδειγμα: Complex αριθμοί

Όπως αναφέρθηκε στο κεφάλαιο 2 ένα χαρακτηριστικό που απαιτείται για επιστημονικούς υπολογισμούς είναι οι μιγαδικοί αριθμοί. Δυστυχώς καμία διάσημη γλώσσα εκτός από τη Fortran δεν τους παρέχει σαν έναν δομημένο τύπο δεδομένων. Ας δούμε πως τα χειρίζεται η Java. Από τη σκοπιά του τύπου δεδομένων δεν χρειάζεσαι πολλά. Από μαθηματική σκοπιά ένας αριθμός complex αποτελείται από ένα πραγματικό και ένα φανταστικό μέρος ν.

Μπορούμε να δημιουργήσουμε μια τέτοια κλάση με τον παρακάτω τρόπο:

```
public class ComplexNumber extends Object {

  public double u;
  public double v;

}
```

Αυτό είναι αρκετό για να εσωκλείσει όλα τα δεδομένα που χρειάζεται κάποιος σε έναν complex αριθμό, δεν είναι, όμως, πολύ καλό παράδειγμα αντικειμενοστρεφούς προγραμματισμού. Για να κάνουμε κάτι μ' αυτούς πρέπει να ξέρουμε πως ορίζεται η δομή δεδομένων. Αν αλλάξουμε τη δομή δεδομένων ορίζοντας π.χ. ένα σύνθετο αριθμό στα τμήματα του μεγέθους r και του ορίσματος theta, αντί στα τμήματα του πραγματικού και του φανταστικού μέρους, πρέπει να αλλάξουμε όλον τον κώδικα που το αφορά.

Πρέπει, επίσης, να γράψουμε κώδικα για να προσθέσουμε αριθμούς, να τους πολλαπλασιάσουμε ή να κάνουμε οτιδήποτε άλλο έχει σχέση με αριθμούς complex. Αν χρειαζόμαστε να προσθέσουμε σύνθετους αριθμούς σε περισσότερο από ένα μέρος, τότε χρειάζεται να γράψουμε τον επιπλέον κώδικα ξανά ή για περισσότερη ευκολία να τον κάνουμε copy και paste.

Μια καλύτερη εφαρμογή της κλάσης σύνθετων αριθμών θα μας προστατεύσει από την ακριβή αποθήκευση των δεδομένων. Θα μας παρέχει, επίσης, μεθόδους που μας αφήνουν να παρουσιάσουμε οποιαδήποτε λειτουργία χρειαστούμε για να παρουσιάσουμε τους σύνθετους αριθμούς.

Πριν γράψουμε τον κώδικα χρειάζεται να αναρωτηθούμε τι θα κάνουμε με τους σύνθετους αριθμούς. Τα περισσότερα αντικείμενα χρειάζονται πρώτα μια constructor, μια μέθοδο που καλείται όταν δημιουργείς έναν νέο σύνθετο αριθμό. Ένα πιο πολύπλοκο αντικείμενο ίσως χρειαστεί μια μέθοδο destructor, που καλείται όταν διώχνεις ένα αντικείμενο.

Αφού οι σύνθετοι αριθμοί είναι 'αριθμοί' είναι πιθανόν να χρειαστεί να τους προσθέσουμε, να τους αφαιρέσουμε , να τους πολλαπλασιάσουμε και να τους διαιρέσουμε. Θα θέλουμε, επίσης, να μπορούμε να χειριστούμε το πραγματικό και το

φανταστικό τους μέρος καθώς, επίσης, τις απόλυτες τιμές και τα ορίσματα. Η ακόλουθη class τα κάνει όλα αυτά:

```
//
public class Complex extends Object {

  private double u;
  private double v;

  Complex (double x, double y) {

    u=x;
    v=y;

  }

  public double Real () {

    return u;

  }

  public double Imaginary () {

    return v;

  }

  public double Magnitude () {

    return Math.sqrt(u*u + v*v);

  }

  public double Arg () {

    return Math.atan2(u, v);

  }

  // Add z to w; i.e. w += z
  public Complex Plus (Complex z) {


    return new Complex(u + z.u, v + z.v);

  }

  // Subtract z from w
  public Complex Minus (Complex z) {

    return new Complex(u - z.u, v - z.v);

  }

  public Complex Times (Complex z) {
```

```
    return new Complex(u*z.u - v*z.v, u*z.v + v*z.u);

  }


  // divide w by z
  public Complex DivideBy (Complex z) {

    double rz = z.Magnitude();

    return new Complex((u * z.u + v * z.v)/(rz*rz),(v * z.u - u *
z.v)/(rz*rz));

  }

}
```

Παρατήρησε ότι τα x και τα y είναι τώρα ιδιωτικά και δεν μπορούν να προσπελασθούν από εξωτερικό κώδικα, ακόμα και αν το θέλουμε.

Η χρήση μιας από αυτές τις μεθόδους θα μοιάζει όπως παρακάτω. Πρόσθεσε την ακόλουθη class ComplexExamples στο αρχείο Complex.java και εκτέλεσέ το. Έπειτα τρέξε το ComplexExamples με το γνωστό τρόπο πληκτρολογώντας

```
//Complex Arithmetic Examples
class ComplexExamples  {

  public static void main (String args[]) {

    Complex u, v, w, z;

    u = new Complex(1,2);
    System.out.println("u: " + u.Real() + " + " + u.Imaginary() + "i");
    v = new Complex(3,-4.5);
    System.out.println("v: " + v.Real() + " + " + v.Imaginary() + "i");

    // Add u + v;
    z=u.Plus(v);
    System.out.println("u + v: "+ z.Real() + " + " + z.Imaginary() +
"i");
    // Add v + u;
    z=v.Plus(u);
    System.out.println("v + u: "+ z.Real() + " + " + z.Imaginary() +
"i");

    z=u.Minus(v);
    System.out.println("u - v: "+ z.Real() + " + " + z.Imaginary() +
"i");
    z=v.Minus(u);
    System.out.println("v - u: "+ z.Real() + " + " + z.Imaginary() +
"i");

    z=u.Times(v);
```

```
    System.out.println("u * v: "+ z.Real() + " + " + z.Imaginary() +
"i");
    z=v.Times(u);
    System.out.println("v * u: "+ z.Real() + " + " + z.Imaginary() +
"i");

    z=u.DivideBy(v);
    System.out.println("u / v: "+ z.Real() + " + " + z.Imaginary() +
"i");
    z=v.DivideBy(u);
    System.out.println("v / u: "+ z.Real() + " + " + z.Imaginary() +
"i");

  }

}
```

Ασκήσεις:

1. Τι θα συμβεί αν προσπαθήσουμε να προσθέσουμε έναν σύνθετο αριθμό με τον εαυτό του;

   π.χ. z=u.Add(u);

   Τι θα συμβεί αν πολλαπλασιάσουμε, αφαιρέσουμε ή διαιρέσουμε;

   ```
   z=u.Multiply(u);

   z=u.Divide(u);

   z=u.Minus(u);
   ```

2. Ξαναγράψε την class Complex έτσι ώστε να αποθηκεύει τα δεδομένα σαν r και theta αντί για u και v. Πρόσεξε το 0.
3. Πρόσθεσε τις μεθόδους PlusEqual, MinusEqual, DivideEqual και MultiplyEqual στην class Complex, οι οποιές μιμούνται τη συμπεριφορά των τελεστών +=, -=, *= και /=.
4. Πρόσθεσε μια μέθοδο ισότητας στην class Complex που ελέγχει πότε δυο αριθμοί είναι ισοδύναμοι και έπειτα επιστρέφει μια τιμή boolean.
5. Για ‘μαθηματικά’ μυαλά μόνο:Εξήγησε γιατί δεν θα είναι καλή ιδέα να προσθέσεις λιγότερες ή περισσότερες methods στην class Complex.
6. Για ‘μαθηματικά’ μυαλά μόνο:Πρόσθεσε μια μέθοδο logarithm στην class Complex number. Πάρε την αγκύλη ανάμεσα στο 0 και στο 2pi.
7. Για ‘μαθηματικά’ μυαλά μόνο:Πρόσθεσε μια μέθοδο power στην class Complex number.

Μέθοδοι tostrings

Η εκτύπωσή μας στο τελευταίο πρόγραμμα ήταν λίγο εξεζητημένη, γιατί χρειαστήκαμε να σπάσουμε έναν αριθμό complex στο πραγματικό και στο φανταστικό του μέρος, να τον τυπώσουμε και μετά να τον τοποθετήσουμε ξανά πίσω. Δεν θα ήταν ωραίο να μπορούσαμε απλά να γράψουμε:

System.out.println(u);

Αποδεικνύεται ότι μπορούμε. Όλα τα αντικείμενα έχουν μια μέθοδο toString που προέρχεται από την Object class. Όμως η εξ' ορισμού μέθοδος toString() δεν είναι πολύ χρήσιμη, έτσι θέλουμε να το ξεπεράσουμε αυτό μόνοι μας με τη χρήση των σύνθετων αριθμών. Πρόσθεσε την ακόλουθη μέθοδο στην Complex class:

```
 public String toString() {

    if (v >= 0) return (String.valueOf(u) + " + " + String.valueOf(v) +
"i");
    else return (String.valueOf(u) + " - " + String.valueOf(-v) + "i");
  }
```

Πρέπει, επίσης, να προσθέσεις την class ComplexExamples όπως παρακάτω:

```
class ComplexExamples  {

  public static void main (String args[]) {

    Complex u, v, z;

    u = new Complex(1,2);
    System.out.println("u: " + u);
    v = new Complex(3,-4.5);
    System.out.println("v: " + v);

    // Add u + v;
    z=u.Plus(v);
    System.out.println("u + v: " + z);
    // Add v + u;
    z=v.Plus(u);
    System.out.println("v + u: " + z);


    z=u.Minus(v);
    System.out.println("u - v: " + z);
    z=v.Minus(u);
    System.out.println("v - u: " + z);

    z=u.Times(v);
    System.out.println("u * v: " + z);
    z=v.Times(u);
    System.out.println("v * u: "+ z);

    z=u.DivideBy(v);
    System.out.println("u / v: " + z);
    z=v.DivideBy(u);
```

```
    System.out.println("v / u: " + z);

  }

}
```

Πολυμορφισμός

Μεχρι τώρα οι μέθοδοι μας έκαναν πράξεις με 2 σύνθετους αριθμούς. Είναι συνηθισμένο να θέλουμε να πολλαπλασιάσουμε έναν σύνθετο αριθμό με έναν πραγματικό. Για να προσθέσουμε αυτήν την δυνατότητα στην κλάση μας θα χρειαστεί να προσθέσουμε την ακόλουθη μέθοδο:

```
  public Complex Times (double x) {

    return new Complex(u*x, v*x);

  }
```

Εδώ είναι ένα απλό πρόγραμμα για τη νέα σας μέθοδο:

```
class RealComplex  {

  public static void main (String args[]) {

    Complex v, z;
    double x = 5.1;


    v = new Complex(3,-4.5);
    System.out.println("v: " + v);
    System.out.println("x: " + x);


    z=v.Times(x);
    System.out.println("v * x: " + z);


  }

}
```

Οι πιο τετραπέρατοι από εσάς ίσως πουν ότι ορίσαμε την μέθοδο Times. Τώρα πως μπορούμε να πολλαπλασιάσουμε 2 σύνθετους αριθμούς; Δεν υπάρχει πρόβλημα. Ο compiler παρατηρεί ότι τα ορίσματα των 2 μεθόδων που ονομάζονται Times είναι διαφορετικά. Το ένα πολλαπλασιάζει 2 σύνθετους άριθμους, το άλλο πολλαπλασιάζει έναν πραγματικό με έναν σύνθετο αριθμό. Ο compiler είναι αρκετά έξυπνος για να ξεκαθαρίσει ποια έκδοση του Times να χρησιμοποιεί κάθε φορά. Αυτό καλείται μέθοδος υπερφορτώματος ή πολυμορφισμού.

Ασκήσεις:

1. Πρόσθεσε μια μέθοδο στην Complex class που προσθέτει έναν complex αριθμό με έναν πραγματικό και επιστρέφει έναν complex.
2. Πρόσθεσε μεθόδους για αφαίρεση ενός πραγματικού αριθμού από έναν σύνθετο και για αφαίρεση ενός σύνθετου αριθμού από έναν πραγματικό. Πρόσεξε γιατί η αφαίρεση, σε αντίθεση με την πρόσθεση δεν είναι τόσο ευμετάβλητη.
3. Πρόσθεσε μεθόδους για διαίρεση ενός πραγματικού με έναν σύνθετο αριθμό και για διαίρεση ενός σύνθετου αριθμού με έναν πραγματικό. Πρόσεχε γιατί η διαίρεση, σε αντίθεση με τον πολλαπλασιασμό δεν είναι ευμετάβλητη.

Καλώντας την κλάση Complex από εξωτερικές κλάσεις

Μέχρι τώρα έχουμε αποθηκεύσει σχεδόν κάθε πρόγραμμα σε ένα μοναδικό αρχείο. Αυτό, όμως, είναι άχρηστο καθώς μεγαλώνουν τα προγράμματα. Καθίσταται αδύνατο να τα χειριστεί όταν περισσότερο από ένα άτομα δουλεύουν στο πρόγραμμα. Επίσης χάνεται ένα από τα πλεονεκτήματα του OOP, η ικανότητα να επαναχρησιμοποιηθεί ο κώδικας. Καθώς όλος ο κώδικας για ένα πρόγραμμα αποθηκεύεται σε ένα αρχείο, δεν μπορείς να τον ξαναχρησιμοποιήσεις αν δεν τον κάνεις πρώτα cut και paste, ακριβώς όπως σε μη αντικειμενοστρεφείς γλώσσες.

Υπάρχει κάποιος κώδικας που δεν ανήκει στα αρχεία πηγής μας. Θυμάσαι όλες αυτές τις εσωτερικές καταστάσεις στην κορυφή κάθε αρχείου; Αυτό που κάνουν είναι να τοποθετούνται σε αρχικό κώδικα(που δεν έχει γραφεί και εκτελεστεί) σε διάφορες περιοχές, έτσι που να μπορούμε να το χρησιμοποιήσουμε στα προγράμματά μας. Μπορείς να κάνεις το ίδιο με τις κλάσεις που γράφεις. Όμως, για να το κάνεις αυτό χρειάζεται να είσαι ενήμερος μερικών συμβάσεων και περιορισμών.

1. Κανένα αρχείο δεν πρέπει να περιέχει περισσότερη από μια δήμοσια κλάση. Αυτό σημαίνει ότι το Hello World, Goodbye World παράδειγμά μας δεν είναι πλέον έγκυρο, γιατί κάθε μια από τις κλάσεις ήταν δημόσια.
2. Όλα τα αρχεία πρέπει να έχουν τι ίδιο όνομα, καθώς οι μοναδικές δημόσιες κλάσεις τους ακολουθούνται από την επέκταση '.java'.
3. Τα αρχεία πηγαίου κώδικα πρέπει να αποθηκεύονται στο ίδιο directory με τα εκτελέσιμα αρχεία .class. Αυτό γίνεται για να μπορεί να βρει ο Java compiler τα κατάλληλα ορίσματα και interface για μια class, όταν η class αναφέρεται σε ένα διαφορετικό αρχείο.
4. Τα αρχεία πηγαίου κώδικα και .class πρέπει να είναι σε ένα directory, που να είναι μέρος της μεταβλητής περιβάλλοντος $CLASSPATH.

Θα το αποδείξουμε αυτό διαχωρίζοντας το παράδειγμα της προηγούμενης παραγραφου σε δύο ξεχωριστά αρχεία, κάθε ένα από τα οποία περιλαμβάνει μια class. Ξεκινάμε δημιουργώντας ένα αρχείο που περιλαμβάνει την class Complex. Σώσε αυτό το αρχείο σαν Complex.java.

Στη συνέχεια σώσε τα παραδείγματα από τις προηγούμενες ασκήσεις σε ένα ξεχωριστό αρχείο που ονομάζεται ComplexExamples.java στο ίδιο directory με το Complex.java. Τώρα compile και τα δυο αρχεία και τρέξε το Complex Examples.java.

The Mandelbrot Set

Το Mandelbrot Set είναι μια κλασσική εφαρμογή της μιγαδικής αριθμητικής.Τώρα και στο μέλλον πρόκειται να διαχωρίσουμε το μαθηματικό ορισμό των δεδομένων μας από την εμφάνισή τους στην οθόνη. Στην πραγματικότητα δεν θα προσθέσουμε την εμφάνιση στην οθόνη μέχρι τη δεύτερη επανάληψη του προγράμματος.

Η δική μας δομή δεδομένων θα είναι ένας διδιάστατος πίνακας του οποίου κάθε στοιχείο αναπαριστά ένα καθορισμένο σημείο στο επίπεδο των σύνθετων. Ο πίνακας είναι ορισμένος με την τιμή του χαμηλότερου δεξιού σημείου , το κενό ανάμεσα στα σημεία και τον αριθμό των pixels στη x και y κατεύθυνση. Έτσι, δοθέντος του σημείου (0,0) του πίνακα, ταιρίαζεται το σημείο complex (x0,y0) , κάθε σημείο διαχωρίζεται με μια τιμή κενού, ξέρουμε ότι το στοιχείο i,j του πίνακα αναπαριστά το σημείο (xo+i*gap, yo+j*gap) στο complex επίπεδο. Αφού το ξέρουμε αυτό από τη θέση του πίνακα στοιχείων μόνο δεν χρείαζεται να αποθηκεύσουμε αυτήν την τιμή στον πίνακα.

Τι θα αποθηκεύσουμε, όμως, σε κάθε στοιχείο του πίνακα; θα υπολογίσουμε ένα αριθμό ο οποίος θα μεταβεί εκεί με τον ακόλουθο τρόπο: Εστω ότι είναι το z=0+0*i και έστω ότι c είναι η θέση του στοιχείου του πίνακα στο χώρο complex. Υπολόγισε το z=z*z+c με περισσότερες από 200 επαναλήψεις.

```
for (i=0; i < 200; i++) {

z = z.Times(z) + c;

}
```

Το Mandelbrot Set αποτελείται από αυτά τα στοιχεία που, αναξάρτητα από το πόσο χρόνο κάνουμε, δεν πλησιάζουν το άπειρο. Από τη στιγμή που το άπειρο θα χρειαστεί πολύ χρόνο για να προσεγγιστεί, είναι τύχη που ένα στοιχειώδης θεώρημα στη σύνθετη μεταβλητή θεωρία εγγυάται ότι οποιοσδήποτε αριθμός του οποίου η σημασία ξεπερνά το 2 σ' αυτό το σχήμα επανάληψης, θα γίνει τόσο μεγάλο όσο επιθυμούμε; Από τη στιγμή που ένας αριθμός ξεπερνά το 2 μπορούμε να διαχωρίσουμε το loop και να πούμε οριστικά ότι αυτός ο αριθμός δεν ανήκει στο Mandelbrot Set.

Δυστυχώς δεν υπάρχει εγγύηση ότι επειδή ένα στοιχείο δεν φθάνει το 2 στις 200 επαναλήψεις δεν θα φθάσει ούτε στις 2000, ούτε στις 2000000 επαναλήψεις. Εδώ είναι πως θα δουλεύει ο κώδικας. Πρώτα θα επιλέξουμε την κάτω αριστερή γωνία του

ορθογωνίου στο χώρο complex, το μέγεθος του κενού ανάμεσα στα σημεία και τους αριθμούς των σημείων σε κάθε διάσταση. Για να κρατάς τους εσωτερικούς υπολογισμούς θα το χωρίσουμε σ' έναν πίνακα με 101 και 101 στοιχεία, που συνεπάγονται ένα κενό με μέγεθος 0.05.

Από τη στιγμή που θα δημιουργηθεί αυτός ο πίνακας θα τον κοιτάξουμε και θα συμπληρώσουμε κάθε στοιχείο του με μια τιμή boolean, true αν το στοιχείο είναι στο Mandelbrot Set και false αν δεν είναι.

Εδώ είναι ο κώδικας:

```
class MandelApp {

  public static void main(String args[]) {

    int xdim = 101;
    int ydim = 101;
    double xstart = -2.0;
    double ystart = -2.0;
    boolean Mandel[][] = new boolean[xdim][ydim];
    double gap = 0.05;
    int max_iterations = 200;
    int i,j,k;
    Complex z, c;

    for (i=0; i < xdim; i++) {
      for (j=0; j < ydim; j++) {

        c = new Complex(xstart + i*gap ,ystart + j*gap);
        z = new Complex(0.0, 0.0);
        k=0;
        while (z.Magnitude() < 2.0 && k < max_iterations) {
          z = z.Times(z);
          z = z.Plus(c);
          k++;
        }
        if (z.Magnitude() < 2.0) {
          Mandel[i][j] = true;
        }
        else Mandel[i][j] = false;
      }
    }

  }
}
```

Ζωγραφίζοντας το MandelbrotSet

Για να το κάνουμε πιο ενδιαφέρον θέλουμε να ζωγραφίσουμε εικόνες στο Mandelbrot Set. Για να το κάνουμε αυτό θα μετακινήσουμε τους πραγματικούς υπολογισμούς σ' ένα thread ενός applet και μετα θα ζωγραφίσουμε τα αποτελέσματα σ' ένα bitmap. Ακολουθεί ο κώδικας:

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

public class Mandelbrot extends Applet {

  int xdim;
  int ydim;
  double xstart = -2.0;
  double ystart = -1.25;
  int Mandel[][];
  double gap = 0.05;
  int max_iterations = 256;


  public void paint(Graphics g) {

    int i,j,k;
    Complex z, c;

    xdim = size().width;
    ydim = size().height;
    gap = 2.5/ydim;
    Mandel = new int[xdim][ydim];

    for (i=0; i < xdim; i++) {
      for (j=0; j < ydim; j++) {
        c = new Complex(xstart + i*gap, ystart + j*gap);
        z = new Complex(0.0, 0.0);
        for (k = 0; z.Magnitude() < 2.0 && k < max_iterations; k++) {
          z = z.Times(z);
          z = z.Plus(c);
        }
        g.setColor(selectColor(k));
        g.fillRect(i, j, 1, 1);
      }
    }


  }


  protected Color selectColor (int num_iterations) {

    if (num_iterations > max_iterations) return Color.black;
    else if (num_iterations > 9*max_iterations/10) return
Color.darkGray;
    else if (num_iterations > 8*max_iterations/10) return Color.gray;
    else if (num_iterations > 7*max_iterations/10) return
Color.magenta;
    else if (num_iterations > 6*max_iterations/10) return Color.cyan;
    else if (num_iterations > 5*max_iterations/10) return Color.blue;
    else if (num_iterations > 4*max_iterations/10) return Color.green;
    else if (num_iterations > 3*max_iterations/10) return Color.yellow;
    else if (num_iterations > 2*max_iterations/10) return Color.orange;
    else if (num_iterations > 1*max_iterations/10) return Color.red;
    else return Color.white;
```

```
    }

}
```

Αυτό το πρόγραμμα είναι εντελώς βασικό. Πρέπει να δημιουργεί έναν Image Producer που να ζωγραφίζει εικόνες στο Mandelbrot Set. Υπάρχουν, επίσης, πολλές προσθήκες που μπορούν να γίνουν στους παραμέτρους για να επιτρέψουν το zoom..Στην πραγματικότητα μπορείς να το χειριστείς σαν ένας Canvas σε ένα applet με διάφορους ελέγχους για να επιλέξεις την περιοχή που σε ενδιαφέρει.

Ασκήσεις:

1. Εξερεύνησε διαφορετικά σημεία εκκίνησης και μεγέθη κενών για το Mandelbrot Set. Για να γίνει αυτό πιο εύκολο πρόσθεσε είσοδο από το χρήστη για να επιλέγει δυναμικά το σημείο εκκίνησης και το μέγεθος του κενού.
2. Τι θα συμβεί αν αφήσεις να επιλέγεται αυθαίρετα το μέγεθος του κενού ανάμεσα στο x και το y;