Πανεπιστήμιο Αιγαίου
Τμήμα Μηχανικών Σχεδίασης Προϊόντων & Συστημάτων

Σημειώσεις για το μάθημα

# Τεχνολογίες & Μεθοδολογίες Προγραμματισμού ΙΙ

Δ΄ εξάμηνο

Ιωάννης Γαβιώτης, gaviotis@aegean.gr

**Στόχοι του μαθήματος**

Οι φοιτητές που θα ολοκληρώσουν αυτό το μάθημα:

- θα αναπτύξουν δεξιότητα στην επίλυση προβλημάτων με υπολογιστές,
- θα ασκηθούν στη σχεδίαση και υλοποίηση ευανάγνωστων, τεκμηριωμένων και αποδοτικών προγραμμάτων,
- θα ελέγχουν την ορθότητα των προγραμμάτων και την καταλληλότητά τους σε σχέση με το πρόβλημα και το χρήστη,
- θα γνωρίσουν το συντακτικό και θα κατανοήσουν τη σημασιολογία των τυπικών προγραμματιστικών δομών, όπως η ανάθεση, η επιλογή, η επανάληψη,
- θα γνωρίσουν τους βασικούς τύπους δεδομένων και τη δυνατότητα ορισμού νέων,
- θα κατανοήσουν τη χρήση απλών δομών δεδομένων, όπως πίνακες, διανύσματα, συλλογές,
- θα γνωρίζουν το μηχανισμό ορισμού, έγερσης και εξυπηρέτησης εξαιρέσεων,
- θα αναπτύσσουν γραφικές διεπαφές για κλασικές εφαρμογές και για εφαρμογίδια (applets) που εκτελούνται ενσωματωμένα σε ιστοσελίδες,
- θα εισαχθούν στον αντικειμενοστρεφή προγραμματισμό με τον ορισμό κλάσεων και των ιδιοτήτων τους, τη στιγμιοποίηση (instantiation) αντικειμένων και την ανταλλαγή μηνυμάτων,
- θα εφαρμόζουν τις αρχές της αντικειμενοστρεφούς σχεδίασης, όπως η ενθυλάκωση (encapsulation), ο πολυμορφισμός και η κληρονομικότητα, και τέλος
- θα αποκτήσουν αίσθηση της πολυπλοκότητας των λειτουργιών ενός αλγορίθμου.

Θα χρησιμοποιηθεί η γλώσσα προγραμματισμού Java. Η συγκεκριμένη γλώσσα λειτουργεί ως όχημα για την κατανόηση των αρχών, εννοιών, τεχνικών και μεθόδων που περιγράφονται στο μάθημα. Ωστόσο, οι φοιτητές θα αποκτήσουν τη δυνατότητα να εφαρμόζουν όλα αυτά σε διάφορα περιβάλλοντα ανάπτυξης και γλώσσες.

**Ύλη**

Φάσεις ανάπτυξης προγραμμάτων - Ενδιάμεση γλώσσα και ιδεατή μηχανή – Περιβάλλοντα ανάπτυξης - Ασφάλεια κώδικα που εκτελείται απομακρυσμένα - Σύγκριση μεταγλωττιστών και διερμηνευτών – Αναγνωσιμότητα, σχόλια, οδόντωση κώδικα – Συντάκτης κατευθυνόμενου συντακτικού

Στρατηγικές επίλυσης προβλημάτων - Ο ρόλος των αλγορίθμων στη διαδικασία επίλυσης προβλημάτων - Στρατηγικές υλοποίησης για αλγόριθμους – Εκσφαλμάτωση - Τεκμηρίωση

Βασικό συντακτικό και σημασιολογία γλωσσών υψηλού επιπέδου - Μεταβλητές, σταθερές, τύποι δεδομένων, παραστάσεις, ανάθεση – Απλή είσοδος / έξοδος - Δομές ελέγχου επιλογής και επανάληψης - Πίνακες - Αντικείμενα – Συμβολοσειρές, επεξεργασία συμβολοσειρών

Μηχανισμοί αφαίρεσης - Βαθμιαία αποδόμηση - Μέθοδοι και πέρασμα παραμέτρων - Αντικειμενοστρεφής σχεδίαση - Κλάσεις, αντικείμενα, ιδιότητες, μέθοδοι - Ενθυλάκωση και απόκρυψη πληροφορίας - Διαχωρισμός συμπεριφοράς και υλοποίησης – Κατασκευαστές, μέθοδοι ανάγνωσης, εγγραφής - Κληρονομικότητα - Πολυμορφισμός

Εκχώρηση μνήμης, συλλογή απορριμμάτων – Διάρκεια ζωής, εμβέλεια ονομάτων - Αναδρομή, αναδρομικές μαθηματικές συναρτήσεις, στρατηγική «διαίρει και βασίλευε»

Εφαρμογίδια – Στοιχεία γραφικής διεπαφής χρήστη - Μέθοδοι χειρισμού συμβάντων - Μετάδοση συμβάντων

**Προαπαιτήσεις**

Οι φοιτητές πρέπει να έχουν τις βασικές γνώσεις για υπολογιστές, το Διαδίκτυο και προγραμματισμό. Τα σχετικά μαθήματα είναι «Πληροφορική Ι» (1151), «Πληροφορική ΙΙ» (2152) και «Τεχνολογίες Μεθοδολογίες Προγραμματισμού Ι» (3150). Ειδικότερα, θα πρέπει να γνωρίζουν τις έννοιες μεταβλητών, σταθερών και τύπων δεδομένων, τη σημασιολογία των τυπικών δομών ελέγχου (if-then-else, for, while, select) και τις αρχές του δομημένου προγραμματισμού (ορισμός & κλήση υποπρογραμμάτων, πέρασμα παραμέτρων).

## Περιεχόμενα

# 1 Εισαγωγή στη Java

## 1.1 Ιστορικό

- Η γλώσσα Java (= Ιάβα, το νησί[1]) ξεκίνησε το 1991 με την ονομασία Oak από μια ομάδα της εταιρίας Sun με υπεύθυνο τον James Gosling και με σκοπό τον αυτοματισμό οικιακών συσκευών. Το 1994 η ομάδα επαναπροσδιορίζει τη Java με στόχο το Διαδίκτυο. Η Java αναγγέλθηκε επίσημα την 23η Μαΐου 1995, στο συνέδριο SunWorld. Τότε η ομάδα ανάπτυξης αποτελούνταν από 30 άτομα, δείτε στο http://java.sun.com/features/1998/05/birthday.html.
  - Το 1995 το πρόγραμμα επόπτευσης ιστοσελίδων Netscape Navigator 2.0 δίνει τη δυνατότητα της εκτέλεσης προγραμμάτων Java μέσα από ιστοσελίδες. Έτσι προσφέρει τη δυνατότητα δυναμικού περιεχομένου και διαδραστικής συμπεριφοράς στις –κατά τα άλλα στατικές– ιστοσελίδες HTML.
  - Αργότερα η Microsoft ενσωματώνει στο δικό της πρόγραμμα πλοήγησης, τον Internet Explorer 4.0, μια ιδεατή μηχανή Java μέσω της οποίας μπορούν να εκτελεστούν εφαρμογίδια γραμμένα σε Java, αλλά με παραλλαγές που ακυρώνουν την φιλοσοφία «run-anywhere». Η Sun μηνύει τη Microsoft και κερδίζει τη δίκη. Μέχρι το Φεβρουάριο του 2003 οι δύο εταιρίες βρίσκονται στα δικαστήρια για το αν τα Windows θα πρέπει να υποστηρίζουν την εκτέλεση προγραμμάτων Java ή αν αυτό το χαρακτηριστικό θα εγκαθίσταται κατ' απαίτηση από το Διαδίκτυο. Πίσω από αυτή τη διένεξη βρίσκεται η βούληση της Microsoft να επιβάλλει τη δική της ενδιάμεση πλατφόρμα εκτέλεσης προγραμμάτων.

**Εικόνα 1 Το λογότυπο και η μασκότ**

- Η Java σχεδιάστηκε με **στόχο** να είναι (α) αντικειμενοστρεφής, (β) κατανεμημένη (distributed), (γ) απλή, (δ) πολυνηματική (multi-threaded), (ε) ασφαλής, (στ) ανεξάρτητη υπολογιστικής πλατφόρμας.
  - Η Java υποστηρίζει εγγενώς τις αρχές του αντικειμενοστρεφούς προγραμματισμού που θα αναλύσουμε από το επόμενο κεφάλαιο. Σε αντιδιαστολή, οι Basic και C είναι διαδικαστικές γλώσσες (procedural languages) βασισμένες στο μοντέλο του δομημένου προγραμματισμού (structured programming).
  - Η Java είναι μια *πλήρης* γλώσσα προγραμματισμού. Υποστηρίζει εξαιρέσεις, νηματική εκτέλεση (threaded execution), κά. Επίσης παρέχει τυποποιημένες βιβλιοθήκες κλάσεων (Java class libraries) μέσω των οποίων υποστηρίζονται γραφικά και γραφικές διεπαφές (Graphical User Interface, GUI), πολυμέσα, επικοινωνία πάνω σε δίκτυα, επαναχρησιμοποιούμενα εξαρτήματα (components), σύνδεση με βάσεις δεδομένων, κά.

- Το κύριο χαρακτηριστικό των προγραμμάτων Java είναι ότι εκτελούνται στο Διαδίκτυο σε περιβάλλον πλοηγητή ιστοσελίδων (web browser), καθώς και σε διάφορες (από πλευράς υλικού και λειτουργικού συστήματος) υπολογιστικές πλατφόρμες – στις οποίες περιλαμβάνονται υπολογιστές, ραδιόφωνα και κινητά τηλέφωνα. Αυτή η φιλοσοφία αποκαλείται Write Once Run Anywhere, WORA ή σε ελεύθερη απόδοση γράφεις-άπαξ-εκτελείται-παντού.
  - Ακόμη κι όταν ένα πρόγραμμα Java λειτουργεί σε διαφορετικές υπολογιστικές πλατφόρμες, στην πραγματικότητα εκτελείται πάνω στην ίδια «μηχανή» – την **ιδεατή μηχανή Java** (Java Virtual Machine, JVM). Η ιδεατή μηχανή Java είναι ένα πρόγραμμα που διερμηνεύει τα εκτελέσιμα προγράμματα Java σε εντολές που μπορεί να εκτελέσει η υπολογιστική πλατφόρμα - στόχος.
  - Η Sun έχει παραχωρήσει σε αρκετές εταιρίες (Oracle, Novell, IBM, κά) την άδεια να κατασκευάσουν ιδεατές μηχανές Java για διάφορες υπολογιστικές πλατφόρμες. Καθώς η ιδεατή μηχανή Java έχει μεταφερθεί σε μεγάλη γκάμα από υπολογιστικές πλατφόρμες και είναι ευρέως εγκατεστημένη, δημιουργούνται συνεχώς νέες γλώσσες προγραμματισμού ή παραλλαγές παλαιότερων γλωσσών προγραμματισμού (Basic, Lisp, Prolog) που μεταγλωττίζονται ώστε το πρόγραμμα – στόχος να εκτελείται στην ιδεατή μηχανή Java. Σύμφωνα με μια πρόσφατη καταμέτρηση (http://www.robert-tolksdorf.de/vmlanguages.html), υπάρχουν 200 τέτοια εργαλεία που δίνουν τη δυνατότητα σε πηγαίο κώδικα που δεν είναι Java να μεταφράζεται και να εκτελείται στην ιδεατή μηχανή Java.
  - Παρά τις διαφορές που έχουν όλες οι υπολογιστικές πλατφόρμες, εντάσσονται στο ίδιο ιδεατό υπολογιστικό μοντέλο: το εκτελέσιμο πρόγραμμα ανακαλείται από τη δευτερεύουσα μνήμη, που είναι συνήθως παραμένουσα (non-volatile), πχ δίσκος ή μνήμη flash, στην πρωτεύουσα ευμετάβλητη (volatile) μνήμη τυχαίας προσπέλασης, πχ RAM, και από εκεί μεταφέρεται εντολή προς εντολή στην κεντρική μονάδα επεξεργασίας. Παρατηρήστε ότι λεπτομέρειες, όπως το μέγεθος της μνήμης, η ύπαρξη καταχωρητών (registers), τα χαρακτηριστικά των μονάδων εισόδου / εξόδου, σκόπιμα δεν περιλαμβάνονται στις προδιαγραφές του αφαιρετικού μοντέλου.

- **Πατρίδα** της Java στο Διαδίκτυο είναι το http://java.sun.com. Μπορείτε να το επισκεφθείτε για να ενημερωθείτε για τις τρέχουσες εκδόσεις της γλώσσας, εκπαιδευτικό υλικό (tutorials), παραδείγματα, κλπ. Για να αναπτύξετε προγράμματα χρειάζεται το Java Development Kit[2], του οποίου η τρέχουσα έκδοση είναι η JDK SE 6u5 (Μάρτιος 2008) και για τα Windows έχει μέγεθος ~71Mb[3]. Για να εκτελέσετε προγράμματα, χρειάζεται το Java Runtime Environment, του οποίου η τρέχουσα έκδοση είναι η JRE SE 6u5 μεγέθους ~15Mb για τα Windows. Το JRE αποτελείται από τη JVM και τις τυ-

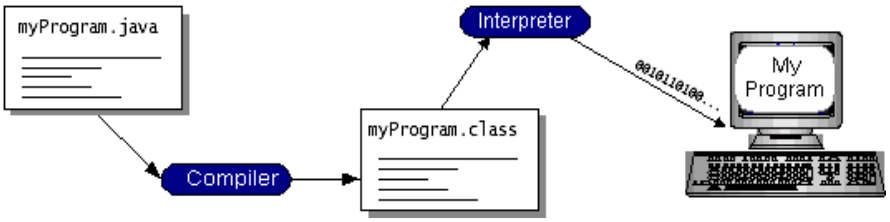
[1] Το όνομα προέκυψε καθώς κάποια μέλη της ομάδας ανάπτυξης έπιναν καφέ σε μια καφετέρια. Έτσι εξηγείται και το λογότυπο που είναι μια κούπα αχνιστού καφέ (Εικόνα 1). Στην αρχική επίδειξη της Java προβαλλόταν μέσα σε μια ιστοσελίδα μια κινούμενη φιγούρα, ο Duke, που έγινε η μασκότ της γλώσσας.
[2] Κυκλοφορούν τρεις εκδόσεις της Java: Enterprise Edition (EE) για επιχειρηματικά περιβάλλοντα, Mobile Edition (ME) για κινητές συσκευές και Standard Edition (SE) που θα χρησιμοποιήσουμε εμείς.
[3] Διατίθενται εκδόσεις για άλλα λειτουργικά συστήματα, όπως Sun Solaris, Linux.

ποποιημένες βιβλιοθήκες κλάσεων.

- Υπάρχουν εκδόσεις του περιβάλλοντος Java για Windows, Linux, Solaris, MacOS, κά. Εσείς επιλέγετε την έκδοση Windows Platform.
- Στο http://java.sun.com/docs/books/tutorial/index.html υπάρχει ένας οδηγός εκμάθησης με πολλά παραδείγματα κώδικα, τόσο για τη βασική χρήση της γλώσσας Java, όσο και για εξειδικευμένα θέματα, πχ ανάπτυξη γραφικών διεπαφών χρήστη, γραφικά, ήχο, κά.
- Μια άλλη πηγή πληροφόρησης είναι το http://www.developer.com/java, ενώ πολλά προγράμματα μπορείτε να βρείτε στο http://javaboutique.internet.com.

## 1.2 Φάσεις ανάπτυξης

- Για να εκτελεστούν τα προγράμματα της Java περνούν από πέντε φάσεις: σύνταξη (edit), μεταγλώττιση (compile), φόρτωση (load), εξακρίβωση (verify), και εκτέλεση (execute).




**Εικόνα 2 Μετάφραση και εκτέλεση προγράμματος**

- Στην πρώτη φάση ο προγραμματιστής συγγράφει τον πηγαίο κώδικα (source code) ακολουθώντας τους κανόνες σύνταξης της γλώσσας. Ο πηγαίος κώδικας αποθηκεύεται ως αρχείο που έχει όνομα το όνομα της κλάσης που υλοποιεί και κατάληξη `.java`. Αρκεί ένας συντάκτης κειμένου (text editor), αλλά συνήθως χρησιμοποιείται **συντάκτης κατευθυνόμενου συντακτικού** (syntax-directed editor), όπως ο jEdit (www.jedit.org, έκδοση 4.2, ~2Mb) που είναι και ο ίδιος γραμμένος σε Java! Αυτές οι εφαρμογές έχουν «αντίληψη» του συντακτικού των εντολών και αναγνωρίζουν τις δεσμευμένες λέξεις της γλώσσας. Έτσι παρέχουν ευκολίες, όπως προκατασκευασμένους σκελετούς προγράμματος, χρωματισμό του κειμένου ώστε να διακρίνονται οι δεσμευμένες λέξεις από ονόματα, αυτόματη οδόντωση, ένδειξη ζευγαρώματος για παρενθέσεις και αγκύλες.

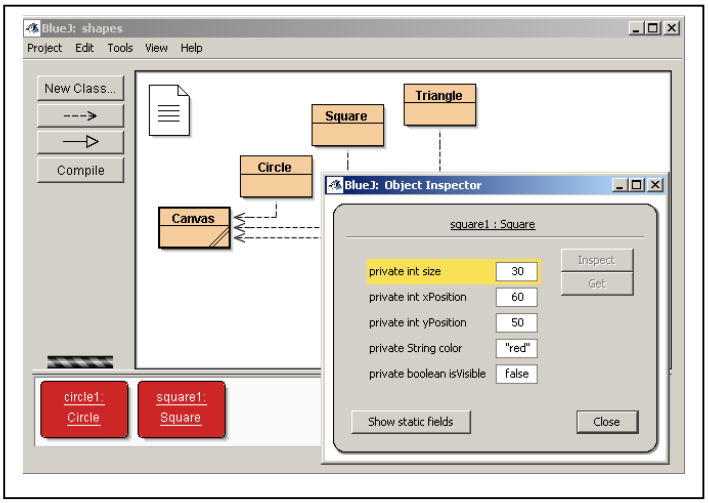
```
C:\>type hello.java
public class hello {
  public static void main(String[] args) {
    System.out.println("hello world");
  }
}

C:\>javac hello.java

C:\>dir hello.*
...
25/02/2004  02:14 μμ               415 hello.class
25/02/2004  02:09 μμ               118 hello.java
...
C:\>java hello
hello world
```

**Εικόνα 3 Οι εντολές μεταγλώττισης και εκτέλεσης**

- Ο συντάκτης μπορεί να είναι ενσωματωμένος σε ένα **ολοκληρωμένο περιβάλλον ανάπτυξης** (Integrated Development Environment, IDE), όπως τα:
  1. netBeans (www.netbeans.org, έκδοση 6.5, ~171Mb) της Sun,
  2. Eclipse (www.eclipse.org, έκδοση 3.4, ~140Mb),
  3. Turbo JBuilder 2007 (http://cc.codegear.com/free/jbuilder, ~507Mb) της Borland.

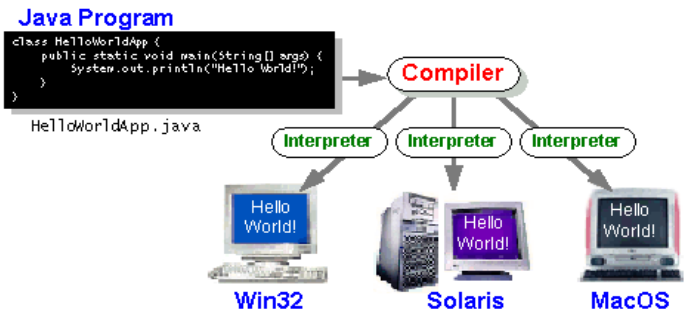
- Τέτοια περιβάλλοντα παρέχουν εργαλεία για εύκολη ανάπτυξη γραφικών διεπαφών (GUI designer), επόπτες ιεραρχίας και τεκμηρίωσης κλάσεων, έλεγχο εκδόσεων (version control), δυνατότητα εκτέλεσης γραμμή-προς-γραμμή, εκσφαλμάτωση με σημεία διακοπής (breakpoints), συγχρονισμό του κώδικα με διαγράμματα UML, κά.

- Σε αντιδιαστολή με τα παραπάνω επαγγελματικά εργαλεία ανάπτυξης, υπάρχουν εργαλεία με εκπαιδευτικό προσανατολισμό.
  - Το BlueJ (www.bluej.org, έκδοση 2.5.0, 4Mb) είναι ένα απλό ολοκληρωμένο περιβάλλον ανάπτυξης σχεδιασμένο για εκπαιδευτικούς σκοπούς. Είναι εξαιρετικά απλό στη χρήση και οπτικοποιεί τη δομή των κλάσεων με διαγράμματα προάγοντας την αντικειμενοστρεφή προσέγγιση. Απαιτεί να είναι εγκατεστημένο το JDK.
  - Ένα άλλο εκπαιδευτικό εργαλείο είναι το Greenfoot (www.greenfoot.org, έκδοση 1.5.1, 5 Mb) που δίνει έμφαση στην οπτικοποίηση των αντικειμένων πάνω σε ένα δισδιάστατο πλέγμα. Η χρήση του ταιριάζει σε εφαρμογές που έχουν απλά γραφικά και είναι οργανωμένες σε σενάρια που κατά την



εκτέλεσή τους επιτρέπουν στον χρήστη να αλληλεπιδρά με τα αντικείμενα που έχουν κατασκευαστεί και είναι ορατά στο πλέγμα. Το Greenfoot συνιστάται να χρησιμοποιείται με τα έτοιμα σενάριά του γιατί δίνει μια οπτική επαφή με το αντικείμενο του προγράμματος – μάλιστα στοχεύει σε μαθητές δευτεροβάθμιας εκπαίδευσης.

- Όποιο περιβάλλον ανάπτυξης κι αν χρησιμοποιείται, το πηγαίο πρόγραμμα πρέπει να μετατραπεί σε κώδικα που να είναι κατανοητός και εκτελέσιμος από την ιδεατή μηχανή Java. Αυτή τη λειτουργία κάνει ο μεταγλωττιστής `javac`, δηλαδή παράγει το ομώνυμο αρχείο bytecode, μόνο που αυτό έχει κατάληξη `.class`. Το αρχείο αυτό περιέχει εντολές της ιδεατής μηχανής Java και μπορεί να είναι μια εφαρμογή ή ένα εφαρμογίδιο (applet).
  - Όταν ο μεταγλωττιστής ανακοινώνει σφάλμα μετάφρασης, το πρόβλημα συνήθως είναι στην γραμμή που αναφέρεται, αλλά μπορεί να εκδηλώνεται εκεί εξαιτίας σφάλματος σε προηγούμενη γραμμή.




**Εικόνα 4 Ο ίδιος bytecode, χωρίς επαναμεταγλώττιση, εκτελείται σε όλες τις πλατφόρμες.**

- Τα προγράμματα που φορτώνονται από το δίκτυο ελέγχονται από τον εξακριβωτή (verifier) bytecode. Αυτό το βήμα είναι απαραίτητο για να ελεγχθεί ότι ο bytecode δεν παραβιάζει τις **αρχές ασφαλείας** της γλώσσας, πχ απαγορεύεται σε εφαρμογίδια να κάνουν ανάγνωση ή εγγραφή στον τοπικό σκληρό δίσκο. Κανονικά, ο μεταγλωττιστής ελέγχει τον πηγαίο κώδικα και παράγει ασφαλή bytecode. Είναι όμως δυνατόν να χρησιμοποιηθεί μεταγλωττιστής που δεν είναι πιστοποιημένος από πλευράς ασφάλειας, ή κάποιος να «πειράξει» τον bytecode. Έτσι ένα πρόγραμμα bytecode δεν μπορεί να θεωρηθεί ασφαλές, γι αυτό και είναι απαραίτητη η εξακρίβωση του bytecode αμέσως πριν την εκτέλεσή του.

- Η ιδεατή μηχανή Java εκτελεί τα προγράμματα Java, διερμηνεύοντας τον bytecode, δηλαδή μετατρέπει τις εντολές που περιέχονται στο .class αρχείο σε εντολές που της πλατφόρμας (πχ υπολογιστής Intel+Linux, ή κινητό SonyEricsson+Symbian) πάνω στην οποία λειτουργεί η ιδεατή μηχανή. Τα προγράμματα Java τα διακρίνουμε σε εφαρμογές και εφαρμογίδια. Οι εφαρμογές (applications) που εκτελούνται κατευθείαν πάνω στην πλατφόρμα-στόχος. Τα εφαρμογίδια (applets) μπορούν να κληθούν από μια ιστοσελίδα και να εκτελεστούν από ή κάποιον επόπτη (browser) ιστοσελίδων που έχει εγκατεστημένη την ιδεατή μηχανή Java ή για δοκιμαστικούς λόγους από το βοηθητικό πρόγραμμα `appletviewer`.

- Ένα μειονέκτημα της Java είναι οι **μειωμένες επιδόσεις** εκτέλεσης των προγραμμάτων της. Τα εκτελέσιμα αρχεία που παράγουν οι μεταγλωττιστές άλλων γλωσσών προγραμματισμού περιέχουν κώδικα σε γλώσσα μηχανής που εκτελείται εγγενώς στην υπολογιστική πλατφόρμα-στόχος (native machine language). Ένα πρόγραμμα Java είναι πιο αργό, επειδή ο Java bytecode διερμηνεύεται από την ιδεατή μηχανή για να μετατραπεί σε γλώσσα μηχανής.
  - Η τεχνική της ιδεατής μηχανής έχει χρησιμοποιηθεί και παλιότερα: οι γλώσσες Pascal και Visual Basic μεταφράζονται σε έναν ενδιάμεσο κώδικα που ονομάζεται p-code. Επίσης, η γλώσσα μηχανής των μικροεπεξεργαστών CISC (Complex Instruction Set Computer) δεν εκτελείται στην πραγματικότητα από το hardware του μικροεπεξεργαστή, αλλά διερμηνεύεται σε μικροκώδικα (microcode) που τελικά εκτελείται στα υποσυστήματα της CPU.
  - Ένα πλεονέκτημα της μεταγλώττισης σε bytecode είναι ότι το .class αρχείο είναι μικρότερο με μέγεθος - τυπικά το μισό - από το αντίστοιχο αρχείο γλώσσας μηχανής που προκύπτει από τη μεταγλώττιση ενός C/C++ προγράμματος (δείτε Εικόνα 3).

## 1.3 Το βασικό λεξιλόγιο

- Μια ενότητα κώδικα, ένα πρόγραμμα κατά την ορολογία των διαδικαστικών γλωσσών προγραμματισμού, ονομάζεται **κλάση** (class). Οι υπορουτίνες της ονομάζονται **μέθοδοι** (methods). Μια κλάση μπορεί να ορίζει μεθόδους για δική της χρήση (ιδιωτικές μέθοδοι) ή για χρήση από άλλες κλάσεις (δημόσιες μέθοδοι). Όταν καλούμε μια μέθοδο, στέλνουμε ένα **μήνυμα** (message) στο αντικείμενο του οποίου η κλάση υλοποιεί την ομώνυμη μέθοδο. Το πλήρες όνομα της μεθόδου είναι *ΌνομαΚλάσης.ΌνομαΜεθόδου* και ακολουθεί η λίστα των παραμέτρων.

- Οι κλάσεις λειτουργούν και όπως οι εγγραφές (records) στις διαδικαστικές γλώσσες προγραμματισμού. Μπορούμε να δηλώσουμε τις **ιδιότητες** (properties) που περιλαμβάνει μια κλάση, κατ' αντιστοιχία με τα πεδία (fields) των εγγραφών σαν να ορίζαμε ένα νέο τύπο δεδομένων. Σε αυτή την περίπτωση οι κλάσεις λειτουργούν ως πρότυπα ή καλούπια δεδομένων. Κατά την εκτέλεση δημιουργούνται **αντικείμενα** (objects) που είναι στιγμιότυπα των κλάσεων.

## 1.4 Συμβάσεις για τα προγράμματα Java

- Συχνά ο φοιτητής-προγραμματιστής πέφτει στην παγίδα να φτιάχνει το πρόγραμμα με σκοπό αυτό απλώς να τρέξει, ωστόσο αυτό είναι μια πέρα-για-πέρα λανθασμένη πρακτική. Το πρόγραμμα πρέπει να είναι φτιαγμένο έτσι ώστε: (α) να τρέχει σωστά, (β) να είναι εύκολο να συντηρηθεί και (γ) να τρέχει γρήγορα – με αυτή τη σειρά προτεραιότητας.

- Χρησιμοποιούμε συγκεκριμένες τεχνικές, όπως η οδόντωση του κώδικα και άλλες συμβάσεις, πχ στην ονομασία των μεταβλητών, που βελτιώνουν την αναγνωσιμότητα, για να επιτύχουμε τους παραπάνω στόχους. Παρότι οι συμβάσεις αυτές δεν είναι υποχρεωτικές για τη συντακτική ορθότητα του προγράμματος, αποτελούν καλές συνήθειες που μακροπρόθεσμα αποδίδουν στην παραγωγικότητα του προγραμματιστή και στην αξιοπιστία του προγράμματος.

1. Ξεκινάμε να γράφουμε από το αριστερό περιθώριο
2. Αποφασίζουμε πόσο θα είναι το βήμα της οδόντωσης, πχ 2-4 χαρακτήρες
3. Κάθε φορά που ξεκινά ένα μπλοκ εντολών, γράφουμε τις εντολές του σώματός του ένα βήμα προς τα μέσα (δεξιά).
4. Όταν τελειώνει ένα μπλοκ εντολών, βγαίνουμε ένα βήμα οδόντωσης προς τα έξω (αριστερά).

**Εικόνα 5 Κανόνες οδόντωσης**

- **Οδόντωση** (indentation) είναι η διαμόρφωση του αριστερού περιθωρίου των εντολών, έτσι ώστε εντολές που ανήκουν στην ίδια ενότητα (βάθος φωλιάσματος), να είναι στοιχισμένες αριστερά. Για παράδειγμα, στην Εικόνα 6, οι δύο εντολές που ξεκινούν με `System` είναι στην ίδια ενότητα, φωλιασμένες ως σώμα της μεθόδου `main`. Επίσης, η `main` είναι φωλιασμένη μέσα στην κλάση.
- Η Java διακρίνει τα πεζά και τα κεφαλαία γράμματα, δηλαδή τα `iCounter`, `icounter`, `ICounter` είναι διαφορετικά ονόματα. Όλες οι δεσμευμένες λέξεις της γλώσσας, όπως οι εντολές, οι (πρωτογενείς) τύποι δεδομένων, κλπ, γράφονται με αμιγώς με πεζά.
- Κάθε κλάση πρέπει να αποθηκεύεται σε ομώνυμο αρχείο με την κατάληξη `.java`, διαφορετικά ο μεταγλωττιστής διαμαρτύρεται. Το πρόγραμμα στην Εικόνα 6 πρέπει να αποθηκευτεί σε αρχείο που να έχει όνομα `HelloWorldApp.java`, προκειμένου να το μεταγλωττίσει ο `javac` στο αρχείο `HelloWorldApp.class` που περιέχει τον bytecode.
  - Η εκτέλεση του προγράμματος αρχίζει από μια μέθοδο που ονομάζεται `main` και έχει επικεφαλίδα **`public static void`** `main (String[] args)`.
  - Κατά την επικρατούσα προγραμματιστική σύμβαση, τα ονόματα των κλάσεων ξεκινούν με κεφαλαίο. Τα ονόματα μεθόδων, αντικειμένων και μεταβλητών ξεκινούν με πεζό. Κάθε λέξη μέσα στο όνομα ξεκινάει με κεφαλαίο, πχ `sFirstDayOfMonth`, `fMhkosGrammhs`.
  - Τα ονόματα μεθόδων ακολουθούνται υποχρεωτικά από παρενθέσεις που περικλείουν τις παραμέτρους τους, πχ `System.out.println("hello")`.

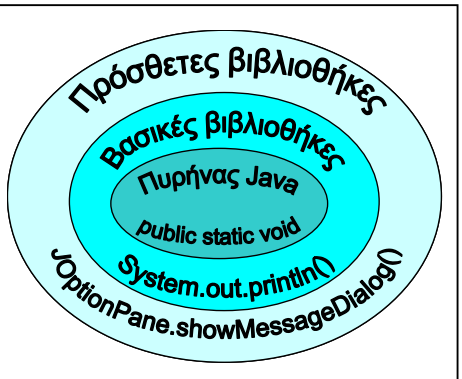
```
/*
 * Η κλάση HelloWorldApp υλοποιεί μια
 * εφαρμογή που προβάλλει "Hello World!"
 */
class HelloWorldApp {
  public static void main(String[] args) {
    System.out.println("Hello World!");
    System.exit(0);
  } // end main
} // end class
```

**Εικόνα 6 Το πρώτο πρόγραμμα Java**

- Υποστηρίζονται δύο τρόποι σύνταξης για **σχόλια**.
  - Το σχόλιο ξεκινά με `//` οπότε αγνοείται οτιδήποτε ακολουθεί μέχρι το τέλος της γραμμής (inline comments).
  - Ο μεταγλωττιστής αγνοεί οτιδήποτε βρίσκεται ανάμεσα σε `/*` και `*/`. Τέτοια σχόλια μπορούν να εκτείνονται σε πολλές γραμμές (δες Εικόνα 6).
  - Σχόλια που ξεκινούν με `/**` περιέχουν τεκμηρίωση (documentation) για την κλάση. Η τεκμηρίωση δημιουργείται αυτόματα από το εργαλείο javadoc (περισσότερα στην ενότητα 4.3 Σχολιασμός και τεκμηρίωση προγράμματος).

## 1.5 Βιβλιοθήκες

- Οι βιβλιοθήκες επεκτείνουν τη βασική λειτουργικότητα της γλώσσας. Επειδή, με την τυπική έννοια, οι βιβλιοθήκες δεν είναι κομμάτι της προδιαγραφής της γλώσσας, δεν τροποποιούν τον πυρήνα της γλώσσας. Για παράδειγμα, οι βιβλιοθήκες δεν μπορούν να αλλάξουν τη σύνταξη ή τη σημασιολογία των δεσμευμένων στοιχείων της γλώσσας, όπως τύπους δεδομένων, εντολές, κλπ. Σταδιακά, προστίθενται και νέες βιβλιοθήκες με βελτιωμένα χαρακτηριστικά που αντιμετωπίζουν νέες απαιτήσεις των προγραμματιστών ή υποστηρίζουν νέες αρχιτεκτονικές εξοπλισμού.
  - Οι σχεδιαστές της Java επέλεξαν να διατηρήσουν μικρό τον πυρήνα της γλώσσας και να υλοποιούν τις πρόσθετες δυνατότητες σε βιβλιοθήκες. Με αυτόν τον τρόπο ο προγραμματιστής αρκεί να γνωρίζει τις εντολές του πυρήνα - για όποια πρόσθετη λειτουργικότητα χρειαστεί, πχ δικτυακό προγραμματισμό, κατασκευή γραφικών διεπαφών, κλπ, απαιτείται να μάθει μόνο την αντίστοιχη βιβλιοθήκη.




**Εικόνα 7 Επεκτάσεις Java**

- Η Java περιλαμβάνει πολλές έτοιμες κλάσεις που ομαδοποιούνται σε υποκαταλόγους κατά κατηγορίες που ονομάζονται **πακέτα** (packages). Αυτό το σύνολο πακέτων αναφέρεται και ως βιβλιοθήκη κλάσεων της Java ή **Java API** (Application Programmer's Interface). Το πλήρες όνομα μιας κλάσης αποτελείται από τα ονόματα των πακέτων στα οποία ανήκει διαχωρισμένα με τελείες, πχ `javax.swing.JApplet`.
- Εν γένει για να χρησιμοποιήσουμε στοιχεία μιας βιβλιοθήκης πρέπει να δηλώσουμε στην αρχή του προγράμματος το

όνομα της κλάσης που τα περιέχει. Αυτό γίνεται με την εντολή `import` και ακολουθεί το πλήρες όνομα της κλάσης.
- Εναλλακτικά, μπορούμε να καλούμε μια μέθοδο μιας κλάσης μέσα στον κώδικα χρησιμοποιώντας το πλήρες της όνομα, οπότε δεν χρειάζονται εντολές **import**.
- Εάν θέλουμε να χρησιμοποιήσουμε πολλές βιβλιοθήκες, για να μην εισάγουμε κάθε κλάση ξεχωριστά, χρησιμοποιούμε τον ειδικό χαρακτήρα πασπαρτού (wildcard) `*`. Για παράδειγμα, η εντολή **import** `javax.swing.*` επιτρέπει τη χρήση όλων των κλάσεων που ανήκουν στο πακέτο Swing[4].

- Οι βασικές βιβλιοθήκες είναι πάντοτε φορτωμένες και τα στοιχεία τους, πχ οι μέθοδοί τους είναι άμεσα διαθέσιμες.
  - Το πακέτο `java.lang` εισάγεται αυτόματα σε όλα τα προγράμματα. Γι αυτό μπορούμε να καλέσουμε τη μέθοδο `System.out.println()` χωρίς κάποια εντολή **import**.
  - Το πακέτο αυτό περιέχει την κλάση `System` που περιλαμβάνει την μέθοδο `exit`. Τα προγράμματα σε γραφικό περιβάλλον για να τερματίσουν κανονικά πρέπει να εκτελούν την `System.exit(0)` ως τελευταία τους εντολή. Σε περίπτωση μη κανονικού τερματισμού, το σωστό είναι να χρησιμοποιείται ο μηχανισμός εξαιρέσεων (exceptions) και να εκτελείται μια `System.exit` που επιστρέφει μη μηδενική τιμή. Τυπικά επιστρέφεται `-1`, αλλά μπορούμε να χρησιμοποιήσουμε και άλλες τιμές που θα υποδηλώνουν τον τύπο του σφάλματος που προέκυψε.

- Όταν φτιάχνουμε προγράμματα που αποτελούνται από πολλές κλάσεις, τις εντάσσουμε στο ίδιο πακέτο βάζοντας ως πρώτη εντολή του αρχείου την **package** *όνομαΠακέτου*. Μέσα στο αρχείο ακολουθούν οι δηλώσεις των πακέτων / κλάσεων που χρησιμοποιούνται από την κλάση με τις εντολές `import` και έπειτα ξεκινά η ίδια η κλάση ή περισσότερες κλάσεις. Στην Εικόνα 8 φαίνεται η ακριβής σύνταξη ενός αρχείου που περιέχει κώδικα Java και εκφράζει με τυπικό τρόπο αυτό που διατυπώθηκε πρωτύτερα. Οι τετράγωνες παρενθέσεις *[]* υποδηλώνουν προαιρετική παρουσία, ο αστερίσκος *** ότι μπορεί να υπάρχουν καμιά, μία ή περισσότερες **import**, και ο σταυρός *+* ότι πρέπει να υπάρχει τουλάχιστον μια κλάση.

```
[ package όνομαΠακέτου; ]
[ import όνομαΒιβλιοθήκης; ]*
[ class ΌνομαΚλάσης {
    σώμαΚλάσης
  } ]+
```

**Εικόνα 8 Σειρά δηλώσεων σε ένα αρχείο**

  - Ακολουθώντας αυτές τις συμβάσεις, ορίζουμε μια γραμματική. Η γραμματική είναι μια **μετα-γλώσσα**, δηλαδή μια γλώσσα που περιγράφει τη σύνταξη για άλλες γλώσσες – στη συγκεκριμένη περίπτωση τη Java. *Παρατήρηση:* οι χαρακτήρες *[ ] * +* είναι σύμβολα της γραμματικής με συγκεκριμένη σημασία, όπως εξηγήσαμε παραπάνω – δεν εμφανίζονται στο αρχείο της Java.

- Σύμφωνα με τη φιλοσοφία της Java, καθήκον του προγραμματιστή είναι να επεκτείνει το προγραμματιστικό περιβάλλον της Java με τρόπο που να υλοποιεί την επιθυμητή λειτουργικότητα, σε αντιδιαστολή με τις διαδικαστικές γλώσσες προγραμματισμού όπου το πρόγραμμα είναι μια αυτόνομη οντότητα που απλώς χρησιμοποιεί τη λειτουργικότητα κάποιων βιβλιοθηκών.

### 1.6 Βασικοί τύποι δεδομένων

- Όλες οι μεταβλητές πρέπει να δηλώνονται πριν να χρησιμοποιηθούν. **Τοπικές** (local) ονομάζονται οι μεταβλητές που δηλώνονται μέσα στην μέθοδο στην οποία χρησιμοποιούνται και ισχύουν. Η σύνταξη της δήλωσης περιγράφεται από τη γραμματική:

  *τύποςΔεδομένων όνομαΜεταβλητής [* **=** *τιμή ]* **;**

  - Αυτό σημαίνει ότι κάθε δήλωση στην αρχή της έχει έναν τύπο δεδομένων και ακολουθεί το όνομα της μεταβλητής που δηλώνεται. Κατά τη δήλωση, ο προγραμματιστής μπορεί να αρχικοποιεί τη μεταβλητή σε μια συγκεκριμένη τιμή. Η φράση *= τιμή* εγκλείεται σε τετράγωνες παρενθέσεις επειδή είναι προαιρετική.

```
char cEpilogh; // χωρίς αρχικοποίηση
int i = 0;     // με αρχικοποίηση – προσοχή δεν βάζουμε [ ] στην αρχική τιμή
```

  - Δεν επιτρέπεται το διάβασμα της τιμής μιας μεταβλητής, πριν να της ανατεθεί κάποια τιμή.
  - Μπορούμε να δηλώσουμε πολλές μεταβλητές του ίδιου τύπου χωρίζοντας τα ονόματά τους με κόμμα, πχ

```
int i, j = 10, k = 100;
float r, s = 1.5, t;
```

  - Η συμπληρωμένη γραμματική που περιγράφει το συντακτικό της εντολής δήλωσης είναι:

    *τύποςΔεδομένων όνομαΜεταβλητής [* **=** *τιμή ] [* **,** *όνομαΜεταβλητής[* **=** *τιμή ] ]* ***** **;**

    Προσθέσαμε τη φράση που είναι κλεισμένη μέσα σε `[ ]` και ακολουθείται από `*`. Ο αστερίσκος σημαίνει 0, μία ή περισσότερες επαναλήψεις. Παρατηρήστε ότι η γραμματική επιβάλλει ότι μια δήλωση θα έχει κατ' ελάχιστον έναν τύπο δεδομένων και ένα όνομα μεταβλητής. Αν υπάρχουν κι άλλες δηλώσεις μεταβλητών στην ίδια εντολή δήλωσης θα είναι χωρισμένες με κόμματα. Προαιρετικά, μπορεί κάποιες από τις μεταβλητές που δηλώνονται να αρχικοποιούνται κιόλας.

  - Μια καλή προγραμματιστική συνήθεια είναι να δηλώνουμε μία-μία τις μεταβλητές σε κάθε γραμμή και να εξηγούμε με ένα σχόλιο για τι χρησιμοποιούνται.
  - Οι επαγγελματίες προγραμματιστές χρησιμοποιούν προθέματα στα ονόματα των μεταβλητών που υποδηλώνουν τον τύπο τους. Οι ακέραιες μεταβλητές έχουν το πρόθεμα `i`, οι πραγματικές το `f` και οι συμβολοσειρές το `s`.

```
int iPlithos = 0; // μετράει πόσοι πελάτες εξυπηρετήθηκαν
```

[4] Ο μεταγλωττιστής δεν φορτώνει *κάθε* κλάση του πακέτου όταν βρίσκει μια `import` με `*`. Φορτώνει μόνο τις κλάσεις που πραγματικά χρησιμοποιούνται στο πρόγραμμα.

```
float fPlatosPortas = 0.92; // το πλάτος της πόρτας σε μέτρα
String sOnomaXrhsth; //το όνομα με το οποίο έχει κάνει login ο χρήστης
```

- Οι **βασικοί τύποι δεδομένων** (primitive data types) της Java είναι οι εξής:
  - **boolean**, που παίρνει τιμές **true** και **false**,
  - **char**, που καταλαμβάνει 16 bit για να αποθηκεύει Unicode χαρακτήρες, πχ
    ```
    char ch = 't';
    char cAll = 'u00F2';
    ```
    (Το παραπάνω σημαίνει να αποδοθεί στη μεταβλητή `cA` η δεκαεξαδική τιμή `00F2`.) Οι τιμές πρέπει να περικλείονται σε μονά εισαγωγικά – τα διπλά εισαγωγικά χρησιμοποιούνται στις συμβολοσειρές που θα δούμε αργότερα.

| Τύπος | Μέγεθος | Ελάχιστη τιμή | Μέγιστη τιμή |
|---|---|---|---|
| **byte** | 8 bit | -128 | 127 |
| **short** | 16 bit | -32.768 | 32.767 |
| **int** | 32 bit | -2.147.483.648 | 2.147.483.647 |
| **long** | 64 bit | -9.223.372.036.854.775.808 | $2^{63}$-1 |
| **float** | 32 bit | $\pm 3{,}4 \times 10^{38}$ με ακρίβεια 7 δεκαδικών ψηφίων | |
| **double** | 64 bit | $\pm 1{,}7 \times 10^{308}$ με ακρίβεια 15 δεκαδικών ψηφίων | |

**Εικόνα 9 Μέγεθος και εύρος τιμών αριθμητικών τύπων**

  - Για τους ακεραίους υπάρχουν οι παραλλαγές **byte**, **short**, **int** και **long** που όλες αποθηκεύουν προσημασμένους ακέραιους σε αναπαράσταση συμπληρώματος ως προς δύο (Θυμόσαστε πώς αναπαριστάνονται οι αρνητικοί αριθμοί;). Μια ακέραια τιμή εκλαμβάνεται ως **int**. Ο κωδικός `L` επιβάλλει στην τιμή να εκληφθεί ως **long**. Πχ
    ```
    int iCount = 7;
    long lBigNum = -342193943L;
    ```
  - Για τους πραγματικούς υπάρχουν οι παραλλαγές **float** και **double** που αποθηκεύουν αριθμούς χρησιμοποιώντας αναπαράσταση κινητής υποδιαστολής. Μια πραγματική τιμή εκλαμβάνεται ως **double**. Οι κωδικοί `F` και `D` επιβάλλουν στην τιμή να εκληφθεί ως **float** και **double**, αντίστοιχα. Πχ
    ```
    float fInch = 2.54F;
    float fDrxPerEuro = 340.75F;
    double dAvogadro = 6.022E23D;
    ```
    Παρατηρήστε ότι εκτός από την μορφή ακέραιο μέρος - υποδιαστολή - δεκαδικό μέρος, μπορούμε να χρησιμοποιούμε την εκθετική μορφή για τους πολύ μεγάλους ή μικρούς αριθμούς. Μετά το `E` ακολουθεί θετικός ή αρνητικός ακέραιος για τον εκθέτη.

- Παρότι δεν ανήκει στους βασικούς τύπους δεδομένων, πρέπει να αναφέρουμε το `String` που δηλώνει μια συμβολοσειρά. Οι τιμή μιας συμβολοσειράς (αναφέρεται και ως αλφαρηθμητικό) μπορεί να περιλαμβάνει χαρακτήρες, αριθμούς και ειδικούς χαρακτήρες. Η τιμή μιας συμβολοσειράς περικλείεται σε διπλά εισαγωγικά[5]. Πχ
  ```
  String sMontelo = "911 Turbo";
  String sMarka = "Porshe" + " " + montelo;
  ```
  Παρατηρήστε τη χρήση του τελεστή `+` για τη συνένωση συμβολοσειρών.

- Εάν προθέσουμε τη δεσμευμένη λέξη **final** σε μια δήλωση μεταβλητής, τότε αυτή μετατρέπεται σε **σταθερά**.
  - Συνηθίζεται τα ονόματα των σταθερών να έχουν μόνο κεφαλαίους χαρακτήρες για να είναι ευδιάκριτα, πχ
    ```
    final int SIZE = 10;
    ```
  - Χρησιμοποιούμε τον χαρακτήρα της υπογράμμισης για να χωρίσουμε τις λέξεις που απαρτίζουν το όνομα της σταθεράς, πχ
    ```
    final double CM_PER_INCH = 2.54; // τόσα εκατοστά έχει μια ίντσα
    ```

- Για κάθε βασικό τύπο δεδομένων υπάρχει ομώνυμη κλάση «περιτυλίγματος» (wrapper class) πχ `Integer`, `Float`, `Double`, κλπ, που προσφέρει χρήσιμες μεθόδους και σταθερές για τον τύπο, όπως `Integer.toString()`, `Double.MAX_VALUE` – ερευνήστε τι κάνει η μέθοδος και η σταθερά (πώς καταλαβαίνετε τι είναι;).
  - Είναι σημαντικό να ανατρέχετε στην τεκμηρίωση των βιβλιοθηκών της γλώσσας για να βρίσκετε κλάσεις, σταθερές, μεθόδους και τον τρόπο χρήσης τους, πχ τις παραμέτρους τους, κά.

- Οι τύποι δεδομένων της Java λειτουργούν το ίδιο σε όλες τις υπολογιστικές πλατφόρμες που υποστηρίζουν την ιδεατή μηχανή Java. Έτσι ο κώδικας που γράφεται και δοκιμάζεται σε ένα περιβάλλον ανάπτυξης πάνω σε συγκεκριμένη πλατφόρμα, πχ Wintel ή SPARC / Unix, θα έχει την ίδια συμπεριφορά παντού και θα υπάρχει συμβατότητα δεδομένων σε δυαδικό επίπεδο. Αυτή είναι μια σημαντική συνιστώσα για να επιτευχθεί **μεταφερσιμότητα** των προγραμμάτων Java.
  - Σε άλλες γλώσσες προγραμματισμού, η αναπαράσταση των δεδομένων εξαρτάται από τα χαρακτηριστικά της υπολογιστικής πλατφόρμας για την οποία μεταγλωττίζονται, όπως το μήκος λέξης (word), η αναπαράσταση big endian / little endian, κά.

## 1.7 Είσοδος / έξοδος

- Ακολουθώντας το μοντέλο της C, η είσοδος / έξοδος στη Java δεν είναι κομμάτι της γλώσσας, όπως σε παλιότερες

[5] Το 'A' είναι ένας χαρακτήρας, ενώ το "A" είναι μια συμβολοσειρά μήκους 1.

γλώσσες. Υλοποιείται με βιβλιοθήκες. Θα γνωρίσουμε τη βιβλιοθήκη `System` που χρησιμοποιεί τα ρεύματα `in` και `out` για είσοδο και έξοδο αντίστοιχα από και προς τη γραμμή διαταγών (command line). Επίσης, θα δούμε τη βιβλιοθήκη Swing που παρέχει πολλά στοιχεία για την υλοποίηση γραφικών διεπαφών χρήστη.

- Η μέθοδος `System.out.println` τυπώνει στην προκαθορισμένη έξοδο. Η `println` (και η παραλλαγή της, `print` που δεν αλλάζει γραμμή όταν τυπώνει), ανήκει στην κλάση `System` που περιλαμβάνεται στο πακέτο `java.lang`.
  - Για να τυπώσουμε αυτούσιο κείμενο πρέπει να το εγκλείουμε σε διπλά –όχι απλά- εισαγωγικά, πχ `"abc"`.
  - Για να τυπώσουμε την τιμή ενός ονόματος, πχ μεταβλητής, το γράφουμε κατευθείαν, χωρίς εισαγωγικά.
  - Συνδέουμε συμβολοσειρές / ονόματα με τον τελεστή `+`.
  - Για να τυπώσουμε ειδικούς χαρακτήρες ή σύμβολα χρησιμοποιούμε **ακολουθίες διαφυγής** (escape sequences) που στη Java γράφονται ως μια ανάποδη μπάρα (backslash) `\` και ακολουθεί ένα γράμμα. Η αλλαγή γραμμής δίνεται μέσω της ακολουθίας διαφυγής `\n` (συμβολίζει το newline). Άλλες ακολουθίες διαφυγής είναι `\\`, `\"`, `\'`, `\t` για την ανάποδη μπάρα, τα διπλά και τα μονά εισαγωγικά και τον στηλοθέτη (tab), αντίστοιχα.

  Εισάγοντας (**`import`**) την κλάση `javax.swing.JOptionPane` μπορούμε να χρησιμοποιήσουμε τις μεθόδους `showMessageDialog` και `showInputDialog` που προσφέρονται για εκτύπωση σε αναδυόμενο παράθυρο και είσοδο από προτρεπτικό, αντίστοιχα.

  - Αργότερα θα δούμε μεθόδους για να εμφανίζουμε στοιχεία σε παράθυρο είτε σε ρυθμό χαρακτήρων (δες κλάση `JTextArea`), είτε σε ρυθμό γραφικών (`java.awt.Graphics`). Επίσης θα δούμε πώς να δημιουργούμε διεπαφές χρήστη που ενσωματώνουν αντικείμενα όπως πλαίσια κειμένου, πλήκτρα εντολών, κτλ.

```
import javax.swing.JOptionPane;
public class swingIO {
  public static void main(String[] args) {

    String sFirst, sSecond;
    sFirst = JOptionPane.showInputDialog("First number:");
    sSecond = JOptionPane.showInputDialog("Second number:");

    float fFirst, fSecond, fResult;
    fFirst = Float.parseFloat(sFirst);
    fSecond = Float.parseFloat(sSecond);

    fResult = fFirst + fSecond;

    JOptionPane.showMessageDialog(null,
      "First number is " + fFirst + "\n" +
      "Second number is " + fSecond + "\n" +
      "Result is " + fResult + "\n"       );

    System.exit(0);
  }
}
```

**Εικόνα 10 Είσοδος / έξοδος με πακέτο Swing**

- Επειδή η συνάρτηση εισόδου επιστρέφει πάντα `String`, πρέπει να το μετατρέψουμε στον κατάλληλο τύπο με τις `Integer.parseInt`, `Float.parseFloat`, `Double.parseDouble`, κοκ (δείτε την Εικόνα 10). Παρατηρήστε ότι η μέθοδος `parseFloat` ανήκει στην κλάση `Float` – δεν έχει σχέση με τη δήλωση του τύπου δεδομένων **`float`**.

- Η Java χειρίζεται την είσοδο / έξοδο ως ροή (stream). Τυποποιημένες ροές είναι οι `System.in` και `System.out` που ορίζονται στο πακέτο `java.io`. Το διάβασμα από την τυποποιημένη είσοδο (πληκτρολόγιο), δεν είναι και τόσο απλό να εξηγηθεί τώρα, καθώς απαιτεί έννοιες που θα καλυφθούν αργότερα (για ένα απλό παράδειγμα δείτε στην ενότητα 1.14 Είσοδος / Έξοδος στην τυπική ροή).

## 1.8 Παραστάσεις, τελεστές και μετατροπές

- Οι **τελεστές** (operators) έχουν τις συνηθισμένες προτεραιότητες που συναντούμε σε όλες τις γλώσσες προγραμματισμού – απλώς αλλάζει ο συμβολισμός των τελεστών μερικών πράξεων, όπως οι λογικές. Γενικά, πρώτα εκτελούνται οι πράξεις μέσα σε παρενθέσεις, έπειτα οι αριθμητικοί τελεστές (πρώτα οι πολλαπλασιαστικοί, μετά οι προσθετικοί) και ακολουθούν οι τελεστές σύγκρισης και οι λογικοί τελεστές (δείτε Εικόνα 11). Σε περίπτωση που σε μια παράσταση συμμετέχουν τελεστές ίδιας προτεραιότητας, εφαρμόζεται ο κανόνας της προσεταιριστικότητας, πχ αριστερή προσεταιριστικότητα `7 - 3 + 1` → `4 + 1` → `5` και δεξιά προσεταιριστικό-

| Τελεστές | Προσεταιριστικότητα | Τύπος τελεστών |
|---|---|---|
| `()` | Αριστερά προς δεξιά | Παρενθέσεις |
| `++ -- + - !(τύπος)` | Δεξιά προς αριστερά | Μοναδιαίοι |
| `* / %` | Αριστερά προς δεξιά | Πολλαπλασιαστικοί |
| `+ -` | Αριστερά προς δεξιά | Προσθετικοί |
| `< <= > >=` | Αριστερά προς δεξιά | Σύγκρισης |
| `== !=` | Αριστερά προς δεξιά | (Αν)Ισότητας |
| `&&` | Αριστερά προς δεξιά | Λογικό Και |
| `\|\|` | Αριστερά προς δεξιά | Λογικό Ή |
| `?:` | Δεξιά προς αριστερά | Επιλογής |
| `= += -= */ /= %=` | Δεξιά προς αριστερά | Ανάθεσης |

**Εικόνα 11 Σειρά προτεραιότητας τελεστών (υψηλή προς χαμηλή)**

τητα `++ - 2` → `++ -2` → `-1`

- Μοναδιαίοι (unary) ονομάζονται οι τελεστές που δέχονται ένα όρισμα. Οι μοναδιαίοι τελεστές έχουν πάντα υψηλότερη προτεραιότητα από τους δυαδικούς τελεστές που δέχονται δύο ορίσματα.
- Ο τελεστής της διαίρεσης / όταν έχει ως ορίσματα ακεραίους, δίνει το αποτέλεσμα της ακέραιας διαίρεσης, πχ `10/4` → `2`, ενώ `10.0/4` → `2.5`. Το υπόλοιπο της ακέραιας διαίρεσης συμβολίζεται με `%`, πχ `10 % 4` → `2`.
- Η ισότητα συμβολίζεται με == για να διαχωρίζεται από την ανάθεση =, μια και τα δύο είναι τελεστές. Η ανισότητα είναι `!=`, και η άρνηση είναι `!`. Οι υπόλοιποι λογικοί τελεστές είναι `&&` για τη σύζευξη (και), `||` για τη διάζευξη (ή)[6], `^` για την αποκλειστική διάζευξη (είτε)[7].
- Πέρα από τον βασικό τελεστή ανάθεσης =, υπάρχουν και οι τελεστές *πράξη*=, που εκτελούν την πράξη και αναθέτουν το αποτέλεσμα στον αριστερό όρο. Πιο συγκεκριμένα, υπάρχουν οι +=, -=, *=, /=. Για παράδειγμα,
  `currentSum += newValue` ισοδυναμεί με `currentSum = currentSum + newValue`.
- Η παράσταση `(a > b > c)` δεν είναι έγκυρη στις γλώσσες προγραμματισμού.

- Για να μετατρέψουμε μια παράσταση σε άλλο τύπο δεδομένων, προθέτουμε ένα **εκμαγείο** (cast), δηλαδή τον τύπο στον οποίο θέλουμε να μετατραπεί το αποτέλεσμα της έκφρασης μέσα σε παρενθέσεις. Πχ `(float) 3/4` επιστρέφει `0.75`, ενώ αν ήταν απλώς `3/4` θα επέστρεφε `0` εκλαμβάνοντας το `/` ως ακέραια διαίρεση, αφού εφαρμόζεται πάνω σε δύο ακεραίους.

- Σε παραστάσεις με μεικτούς τύπους δεδομένων ή κατά την αντιστοίχιση πραγματικών παραμέτρων με τις τυπικές, η γλώσσα, εφόσον είναι δυνατό, μετατρέπει τα δεδομένα στον κατάλληλο τύπο, ακόμη κι αν ο προγραμματιστής δεν χρησιμοποιήσει εκμαγείο. Αυτό ονομάζεται **καταναγκασμός** (coercion). Για παράδειγμα, `Math.sqrt(4)` θα μετατρέψει τον ακέραιο 4 σε πραγματικό διπλής ακρίβειας Οι μετατροπές αυτές ακολουθούν τον κανόνα της **προαγωγής** (promotion, widening). Κατά σειρά, η προαγωγή μετατρέπει από τον πιο περιοριστικό τύπο προς τον πιο εκτεταμένο, κατά τη σειρά **`byte` → `short` → `char` → `int` → `long` → `float` → `double`**. Ο τύπος **`boolean`** δεν μετατρέπεται. Εάν σε μια μεικτή παράσταση υπάρχουν τμήματα που δεν έχουν συμβατούς τύπους και δεν μπορούν να υποστούν καταναγκασμό, ο μεταγλωττιστής επιστρέφει μήνυμα λάθους.
  - Παρότι η Java χρησιμοποιώντας τον κανόνα της προαγωγής μετατρέπει μεικτές παραστάσεις σε αμιγείς, μια ασφαλής πρακτική είναι ο προγραμματιστής να χρησιμοποιεί εκμαγεία και να κάνει ο ίδιος τις απαραίτητες μετατροπές.

- Ο τελεστής + είναι **υπερφορτωμένος** (overloaded), καθώς εκτελεί πρόσθεση όταν εφαρμόζεται σε αριθμητικά δεδομένα και συνένωση (concatenation) όταν εφαρμόζεται σε αλφαριθμητικά. Ο κώδικας στην Εικόνα 12 στην πρώτη `println` θα τυπώσει `10.5` εκτελώντας το + ως πρόσθεση, στη δεύτερη `Λάθος: 73.5` εκτελώντας συνένωση

```
int i = 7;
double d = 3.5;
System.out.println(i + d);                    //10.5
System.out.println("Λάθος:" + i + d);         //73.5
System.out.println("Αθροισμα:" + (i + d));    //10.5
```

**Εικόνα 12 Μια συνέπεια της υπερφόρτωσης**

  - Για την ακρίβεια, όλοι οι αριθμητικοί τελεστές είναι υπερφορτωμένοι, ακόμα και αυτοί που εφαρμόζονται πάνω σε αμιγώς αριθμητικά δεδομένα. Ανάλογα με τον τύπο των δεδομένων επί των οποίων εφαρμόζονται, μεταφράζονται σε διαφορετικό bytecode, πχ για την αφαίρεση `int`, `long`, `float`, `double`, κλπ.
    `23L + 35.0f` εκτελεί **`float`** πρόσθεση
    `6 / 3L - 2.0` εκτελεί **`long`** διαίρεση και **`double`** αφαίρεση

- Οι περισσότεροι τελεστές είναι δυαδικοί με την έννοια ότι δέχονται δύο ορίσματα, πχ `4 + 7` ή `x > y`. Υπάρχει ο μοναδιαίος τελεστής − που επιστρέφει τον αντίθετο ενός αριθμού, πχ αν `a = 7`, τότε `a + (- a)` → `0`. Μοναδιαίος είναι και ο τελεστής `!` της λογικής άρνησης.
  - Υπάρχει ένας και μοναδικός τριαδικός τελεστής που ονομάζεται **τελεστής υπόθεσης** και συντάσσεται με τρία ορίσματα, εκ των οποίων το πρώτο πρέπει να είναι λογικό. Η σύνταξη του τελεστή υπόθεσης είναι `b?s:t` και επιστρέφει `s` αν το `b` είναι αληθές, `t` διαφορετικά. Για παράδειγμα
    ```
    bOK = Math.abs(a - b) < 0.1E-7 ? true : false;
    System.out.println("Τερματισμός με " + (bOK ? "επιτυχία" : "αποτυχία"));
    ```

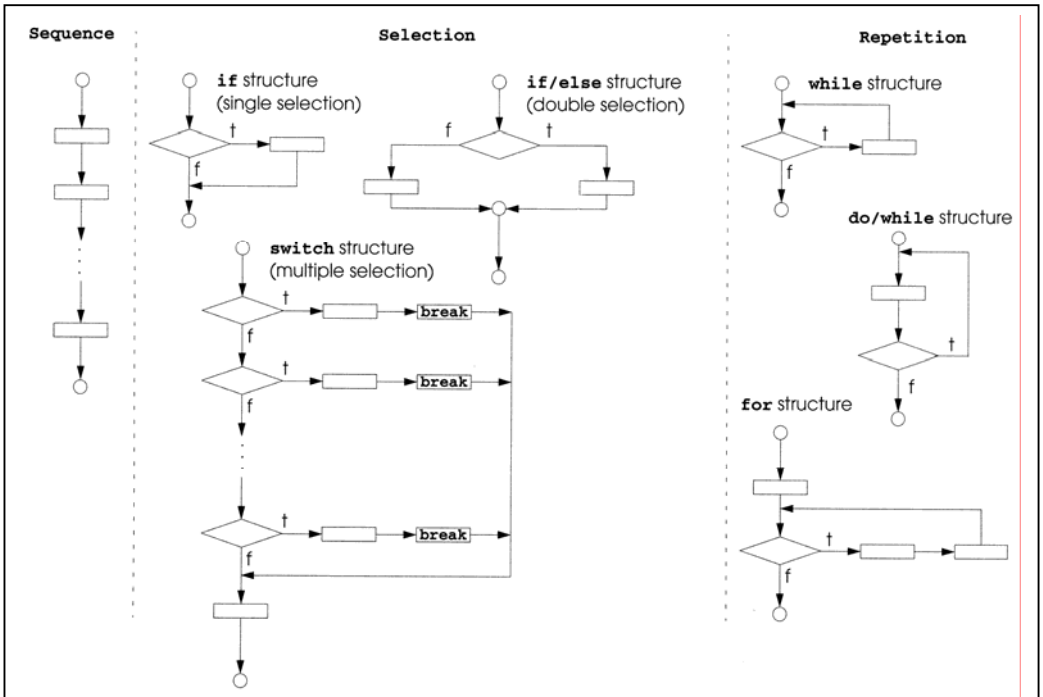
- Η παράσταση `x = y` εκτός από την **ανάθεση** της τιμής του `y` στο `x`, επιστρέφει ως τιμή της παράστασης την τιμή του `y`. Επειδή ως τελεστής το = είναι δεξιά προσεταιριστικός, η παράσταση `x = y = z` ισοδυναμεί με `x=(y=z)` που αναθέτει την τιμή του `z` διαδοχικά στα `y` και `x`.
  - Σε μια ανάθεση αριστερά του = πρέπει να έχουμε ένα όνομα (μεταβλητής), και δεξιά μια παράσταση. Κανονικά η παράσταση πρέπει να επιστρέφει μια τιμή τύπου όμοιου με τον τύπο του ονόματος που βρίσκεται στο αριστερό μέρος, αλλιώς γίνεται προαγωγή. Πχ **`int`** `i=3;` **`long`** `h=3*(i-2);` η παράσταση δεξιά του = έχει τύπο **`int`** και προάγεται σε **`long`** για να ανατεθεί στη μεταβλητή `h`.

---

[6] Οι δύο αυτοί τελεστές λειτουργούν με **υπολογισμό περιορισμένης έκτασης** (short circuit evaluation). Υπάρχει και μια παραλλαγή τους, ο & για την σύζευξη και ο | για τη διάζευξη που υπολογίζουν όλα τα ορίσματά τους πάντοτε.

[7] Υπενθυμίζεται ότι α && β είναι αληθές μόνον αν και το α και το β είναι αληθή. α || β είναι ψευδές μόνον αν και το α και το β είναι ψευδή. α ^ β είναι αληθές αν είτε το α, είτε το β είναι αληθές, αλλά όχι και τα δύο μαζί.

- Η Java έχει κληρονομήσει από τη γλώσσα C τους τελεστές αύξησης / μείωσης[8]. Η παράσταση `x += 4` ισοδυναμεί με `x = x + 4`. Ανάλογα δουλεύουν οι τελεστές `-=`, `*=`, `/=`, `%/`.
  - Οι μοναδιαίοι (unary) τελεστές `++` και `--` αυξάνουν και μειώνουν κατά 1 αντίστοιχα την τιμή της μεταβλητής.
  - Μια κακή προγραμματιστική πρακτική είναι να ανακατεύονται σε μια παράσταση τελεστές που προκαλούν **παρενέργειες** (side-effects), πχ αν x = 3, τότε η ανάθεση `y = x ++ * 2;` δίνει στο y την τιμή 6 και έπειτα αυξάνει το x κατά 1. Αν πάλι x = 3, η ανάθεση `y = ++ x * 2;` αυξάνει το x κατά 1 και έπειτα αναθέτει στο y την τιμή 8. Με τέτοιου είδους παραστάσεις μπορεί να εξοικονομήσουμε λίγο χώρο στη μνήμη ή λίγο χρόνο εκτέλεσης, αλλά παίρνουμε το ρίσκο να εισάγουμε στο πρόγραμμα ένα λογικό σφάλμα που είναι δύσκολο να εντοπιστεί και κάνουμε τον κώδικα πιο δυσνόητο.




**Εικόνα 13 Διαγράμματα ροής για δομές ελέγχου της Java**

## 1.9 Μαθηματικές μέθοδοι

- Στην κλάση `Math` υλοποιούνται πολλές χρήσιμες μαθηματικές μέθοδοι, όπως τετραγωνική ρίζα `sqrt`, νεπέρειο εκθετικό και λογάριθμος `exp` και `log`, ύψωση σε δύναμη `pow`, απόλυτη τιμή `abs`, οροφή `ceil`, πάτωμα `floor`, οι τριγωνομετρικές `cos`, `sin`, `tan`, κά.
  - Ανατρέξτε στην τεκμηρίωση της Java για να εξοικειωθείτε με τη δόμηση σε πακέτα και κλάσεις.
- Η μέθοδος `random` επιστρέφει πραγματικούς αριθμούς διπλής ακρίβειας (**`double`**) τυχαίους αριθμούς στο διάστημα [0.0, 1.0).
  - Εάν χρειαζόμαστε τυχαίους ακέραιους αριθμούς στο διάστημα [x, y], τότε απλώνουμε το εύρος πολλαπλασιάζοντας και μεταθέτουμε προσθέτοντας ως εξής: `x + (int) ((y-x+1) * Math.random())`. Για παράδειγμα αν θέλουμε τυχαίους αριθμούς από το 1 ως το 6 (ζάρι), η παράσταση διαμορφώνεται

```
1 + (int) ((6-1+1) * Math.random())     ή     1 + (int) (6 * Math.random())
```

## 1.10 Δομές ελέγχου

- Σε ένα απλό πρόγραμμα οι εντολές εκτελούνται ακολουθιακά με τη σειρά που είναι γραμμένες στο κείμενο του πηγαίου κώδικα. Δομές ελέγχου (control structures) ονομάζονται οι γλωσσικές κατασκευές που αλλάζουν την φυσιολογική ροή εκτέλεσης του προγράμματος.
  - Οι δομές ελέγχου ορίζουν περιοχές κώδικα που εκτελούνται, επαναλαμβάνονται ή παραλείπονται ανάλογα με μια συνθήκη.
  - Για να είναι ευδιάκριτα τα όρια του σώματος μιας τέτοιας εντολής, είναι απαραίτητο να χρησιμοποιούμε οδόντωση.
- Η **εντολή επιλογής** έχει σύνταξη:

```
if ( συνθήκη )
  εντολήΑ
[else εντολήΒ]
```

Οι τετράγωνες παρενθέσεις υποδηλώνουν ότι η πρόταση **`else`** είναι προαιρετική. Οι παρενθέσεις γύρω από την συνθήκη είναι υποχρεωτικές. Πχ

```
if (a == b) // αν a ισούται με το b
  System.out.println("ΙΣΑ"); // τυπώνει
if (a >= b) // μεγαλύτερο ή ίσο
```

[8] Παλαιότερα, η χρήση των τελεστών αυτών μέσα σε πολύπλοκες παραστάσεις οδηγούσε τον μεταγλωττιστή να παράγει πιο γρήγορο κώδικα. Πλέον με τη χρήση μεταγλωττιστών βελτιστοποίησης (optimizing compilers), αυτή η πρακτική δεν χρειάζεται, τουναντίον αποθαρρύνεται γιατί οδηγεί σε κώδικα δυσανάγνωστο και επιδεκτικό σφαλμάτων.

```
  max = a;
else          // διαφορετικά
  max = b;
```

- Κατά την εκτέλεση, εκτιμάται η συνθήκη και αν είναι αληθής ο έλεγχος μεταβιβάζεται στην εντολήΑ. Αν είναι ψευδής, εκτελείται η εντολήΒ, αν υπάρχει πρόταση **else**.
- Σημειώστε ότι *δεν* χρησιμοποιείται δεσμευμένη λέξη then. Η συνθήκη πρέπει να είναι μια λογική παράσταση, δηλαδή μια παράσταση που όταν εκτιμηθεί δίνει αληθές ή ψευδές.
- Για να τοποθετήσουμε πολλές εντολές στη θέση της *εντολήΑ*, τις περικλείουμε με αγκύλες δημιουργώντας έτσι ένα μπλοκ εντολών. Παράδειγμα:

```
if ( a == b ) {
  y = 0;
  System.out.println("Ισότητα");
} else
  y = 1;
```

- Στο παραπάνω παράδειγμα, παρατηρήστε την οδόντωση του κώδικα που υποδηλώνει (αλλά δεν επιβάλει) την οργάνωση του κώδικα σε μπλοκ.Πάντοτε οδοντώνουμε τις **if**, για να είναι ευανάγνωστος ο κώδικας.

- **Μπλοκ εντολών** χρησιμοποιούμε όπου χρειάζεται να ομαδοποιήσουμε εντολές για να τις κάνουμε να λειτουργούν ως σύνολο. Για παράδειγμα, το σώμα κάθε μεθόδου είναι ένα μπλοκ εντολών.
  - Είναι ατυχές ότι οι σχεδιαστές της Java προκειμένου να διατηρήσουν την ομοιότητα με την C / C++ υιοθέτησαν τη σύνταξη του μπλοκ εντολών με τις αγκύλες {}, αντί να χρησιμοποιήσουν οριοθετημένες εντολές, όπως κάνει για παράδειγμα η Basic με τις Sub...End Sub, For...Next, While...Wend, If...Then...Else...Endif. Αν και ο μεταγλωττιστής ελέγχει τα ζεύγη από αγκύλες που οριοθετούν ένα μπλοκ, κάποιος άνθρωπος που διαβάζει τον πηγαίο κώδικα δεν έχει μια σαφή ένδειξη ποιες εντολές ανήκουν σε ποιο μπλοκ – εκτός κι αν ο κώδικας είναι γραμμένος με οδόντωση. Επιπρόσθετα, είναι χρήσιμο να χρησιμοποιούμε σχόλια που να εξηγούν ποια εντολή κλείνει μια αγκύλη – δείτε ως παράδειγμα το πρόγραμμα στην ενότητα 1.15 Γεννήτρια τυχαίων αριθμών. Προσοχή, ο μεταγλωττιστής αγνοεί τόσο την οδόντωση, όσο και τα σχόλια και ταιριάζει τις αγκύλες σύμφωνα με την σειρά τους στο κείμενο.
  - Ένα κοινό λογικό σφάλμα είναι να βάζουμε ; μετά τη συνθήκη της **if**, όπως φαίνεται στην Εικόνα 14. Ο μεταγλωττιστής δεν βρίσκει συντακτικό σφάλμα γιατί θεωρεί ως σώμα της **if** την κενή εντολή ; και ακολουθεί ένα μπλοκ τριών εντολών που είναι άσχετες με την **if**. Κατά την εκτέλεση, το μπλοκ που περικλείει τις τρεις αναθέσεις θα εκτελείται πάντα, είτε a > b, είτε όχι.

```
if (a > b);
{
  temp = a;
  a = b;
  b = temp;
} //end if
```

**Εικόνα 14 Λογικό σφάλμα σε if**

- Για την περίπτωση που θέλουμε να ελέγχουμε για πολλές (περισσότερες από δύο) περιπτώσεις, υπάρχει η **εντολή πολλαπλής επιλογής** που συντάσσεται ως εξής:

```
switch ( παράσταση ) {
  case τιμή1: εντολή1
  case τιμή2: εντολή2
  case τιμή3: εντολή3
  …
}
```

  - Στην αρχή της εντολής αυτής εκτιμάται η παράσταση. Έπειτα ελέγχονται με τη σειρά οι επιλογές (φράσεις **case**) που ακολουθούν – όταν η τιμή της παράστασης ταιριάζει με κάποια σταθερά, εκτελείται η εντολή που ακολουθεί. Κάθε επιλογή συνήθως τελειώνει με την εντολή **break** – διαφορετικά η ροή εκτέλεσης ελέγχει και τις υπόλοιπες επιλογές και αν κάποια ή κάποιες ισχύουν, τις εκτελεί όλες με τη σειρά που τις συναντάει.
  - Αντί για μια σταθερά στη φράση **case**, μπορούμε να έχουμε την ετικέτα **default:**. Το αντίστοιχο σώμα εκτελείται για όποια τιμή της παράσταση - φτάνει να μην έχει προηγηθεί κάποια break που θα τερματίσει τη **switch** προτού να φτάσει στη **default**.

```
switch (c){
  case '1': case '3': case '5': case '7': case '9':
    System.out.println("περιττός αριθμός");
    break;
  case '0': case '2': case '4': case '6': case '8':
    System.out.println("άρτιος αριθμός");
    break;
  case ' ':
    System.out.println("κενός χαρακτήρας");
    break;
  default :
    System.out.println("ούτε ψηφίο, ούτε κενό");
  }
```

**Εικόνα 15 Παράδειγμα εντολής case**

  - Η λειτουργία της εντολής πολλαπλής επιλογής είναι ίδια με πολλαπλές **if-else** (φωλιασμένες, αν υπάρχουν **break**). Προτιμούμε όμως την **switch** όταν έχουμε πολλές επιλογές, γιατί είναι πιο ευανάγνωστη. Η **case** είναι πιο περιορισμένων δυνατοτήτων από την **if**, γιατί μπορεί να ελέγχει αν η παράσταση ισούται με μια (πεπερασμένη) γκάμα τιμών, ενώ δεν μπορεί να ελέγχει εκφράσεις όπως (a>b) ή (x==0.0 && y==1.0). Επίσης, δεν μπορεί να χρησιμοποιηθεί για να επιλέξει ανάμεσα σε τιμές για συμβολοσειρές, πχ s.equals("γεια").

- Οι **εντολές επανάληψης** είναι οι **`while`**, και **`do`**…**`while`** και **`for`**. Παρακάτω περιγράφεται το συντακτικό και εξηγείται η σημασιολογία τους. Η εντολή

  **`while (`** *παράσταση* **`)`**
  *σώμαΕπανάληψης*

  υπολογίζει την παράσταση και όσο είναι αληθής εκτελεί τις εντολές στο *σώμαΕπανάληψης*. Προφανώς, μέσα στο σώμα της επανάληψης πρέπει να συμβαίνει κάτι που να τροποποιεί κάποιον από τους όρους που συμμετέχουν στην παράσταση, έτσι ώστε κάποτε αυτή να γίνει ψευδής και να τερματιστεί η επανάληψη. Αλλιώς έχουμε έναν ατέρμονα βρόχο. Αν στο σώμα της επανάληψης χρειάζεται να συμπεριλάβουμε πολλές εντολές (σχεδόν πάντα έτσι είναι – γιατί;), τις εγκλείουμε σε ένα μπλοκ εντολών με `{}`.
  - Η εντολή

    **`do`**
    *σώμαΕπανάληψης*
    **`while (`** *παράσταση* **`);`**

    έχει την ίδια συμπεριφορά με την **`while`** μόνο που η παράσταση υπολογίζεται μετά το σώμα της επανάληψης, άρα

```
if (x > y)
  ab = x - y;
else {
  z = x * y;
  ab = y - x;
} //end if
```

```
switch (r) {
  case 1:
    s = 100;
    break;
  case 2:
    s = 200;
    break;
  default:
    s = 0;
}//end switch
```

```
int i = 1;
long sum = 0;
while (i <= 10) {
  sum += i;
  i++;
} //end while
```

```
int i = 1;
long sum = 0;
do {
  sum += i;
  i++;
}
while (i <= 10);
```

```
long sum = 0;
for (int i = 1;
     i <= 10;
     i++)
  sum += i;
```

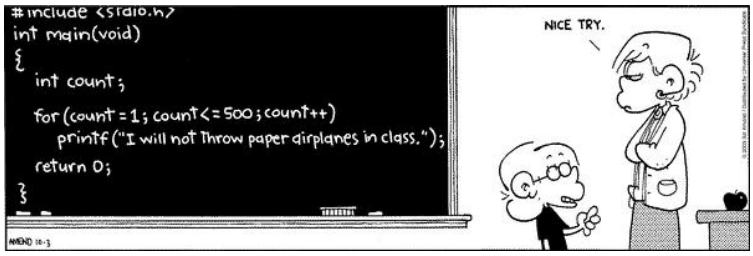
**Εικόνα 16 Παραδείγματα χρήσης εντολών επιλογής και επανάληψης**

ο βρόχος θα εκτελεστεί τουλάχιστον μια φορά.

- Η εντολή

  **`for (`** *αρχικοποίησηΒρόχου; ελεγχοςΕπανάληψης; ενημέρωσηΜεταβλητών* **`)`**
  *σώμαΕπανάληψης*

  είναι πιο πολύπλοκη. Αρχικά εκτελείται άπαξ η παράσταση *αρχικοποίησηΒρόχου*. Υπολογίζεται η δυαδική παράσταση *ελεγχοςΕπανάληψης* και αν είναι αληθής, εκτελείται το *σώμαΕπανάληψης*, και έπειτα η *ενημέρωσηΜεταβλητών*. Έπειτα επαναλαμβάνεται ο υπολογισμός του ελέγχου, κοκ. Η **`for`** καλό είναι να χρησιμοποιείται σε επαναλήψεις που ελέγχονται με μετρητές.




**Εικόνα 17 Η χρησιμότητα της for**

- Με την εντολή **`break`** η ροή εκτέλεσης αμέσως εξέρχεται της επανάληψης. Με την εντολή **`continue`** παραλείπεται η εκτέλεση των υπόλοιπων εντολών του σώματος της επανάληψης και εκτελείται ο επόμενος βρόχος. Ωστόσο χρησιμοποιήστε τις με προσοχή γιατί προκαλούν μη δομημένο κώδικα, καθώς αντιστοιχούν σε `goto`.
- Η Java δεν έχει εντολή `goto`[9].

## 1.11 Πίνακες και συμβολοσειρές

- Ο **πίνακας** (array) είναι ένα σύνολο από συνεχείς θέσεις μνήμης που έχουν το ίδιο όνομα και τον ίδιο τύπο. Για να αναφερθούμε σε ένα στοιχείο του πίνακα προσδιορίζουμε τον αριθμό της θέσης του στον πίνακα που λέγεται υποδείκτης (subscript).
  - Οι πίνακες είναι στατικές δομές δεδομένων – το μέγεθος τους δεν μπορεί να αλλάξει μετά τη δημιουργία τους.
  - Με τη δήλωση **`int`**`[] x` δηλώνεται το όνομα για ένα πίνακα με στοιχεία ακέραιους. Υποστηρίζεται και το εναλλακτικό συντακτικό **`int`** `x[]`.
  - Με την εντολή `x =` **`new int`**`[10]` κατασκευάζεται πίνακας με υποδείκτες 0...9 (τότε δεσμεύεται η απαραίτητη μνήμη). Η ιδιότητα `length` ενός πίνακα δίνει τον αριθμό των κελιών που έχει, πχ `x.length` → 10.
  - Η δήλωση μιας μεταβλητής πίνακα και η δημιουργία του

```
int[] x = new int[10];
int iLen = x.length; //έξω από loop
for (int i = 0; i < iLen; i ++) {
  x[i] = 1 + (int) (6 * Math.random())
}
```

**Εικόνα 18 Διαπέραση πίνακα**

[9] Έχει κάτι λιγότερο κακό, τις εντολές **`break`** και **`continue`** με ετικέτες, που σκόπιμα *δεν* παρουσιάζουμε.

πίνακα μπορούν να συνδυαστούν σε μια εντολή:

```
int[] x = new int[10];
```

- Μπορούμε να κατασκευάσουμε και να αρχικοποιήσουμε έναν πίνακα με τη σύνταξη `int[] y = {4, -2, 7}`. Από το πλήθος των στοιχείων στις αγκύλες προκύπτει το μέγεθος του πίνακα. Έτσι η παραπάνω εντολή φτιάχνει πίνακα τριών θέσεων `y[0]=4, y[1]=-2, y[2]=7`.
- Ο πίνακας `float[][] p = new float[10][10];` αναπαριστά μια μήτρα δύο διαστάσεων. Τα στοιχεία του προσπελαύνονται ως `p[0][0]` ή `p[6][3]`. Κατ' αντίστοιχο τρόπο μπορούμε να ορίσουμε πολυδιάστατους πίνακες. Ένας έμμεσος τρόπος για να ορίσουμε πολυδιάστατους πίνακες είναι να τους δηλώσουμε ως πίνακες που έχουν ως στοιχεία τους άλλους πίνακες.
- Όλοι οι πίνακες είναι αντικείμενα, όχι απλές μεταβλητές – ακόμη και αυτοί που έχουν ως στοιχεία τους απλούς τύπους. Γι' αυτό δημιουργούνται με τη δεσμευμένη λέξη **new**. Περισσότερα γι' αυτό στο επόμενο κεφάλαιο.

- Οι τιμές των **συμβολοσειρών** περικλείονται σε διπλά εισαγωγικά (ενώ οι χαρακτήρες σε μονά εισαγωγικά). Οι συμβολοσειρές που είναι αντικείμενα της κλάσης `String` δεν μπορούν να αλλάξουν περιεχόμενο.
  - Τώρα που γνωρίζουμε πίνακες και συμβολοσειρές, γίνεται κατανοητό ότι το `String[] args` στην επικεφαλίδα της μεθόδου `main` είναι η δήλωση ενός πίνακα συμβολοσειρών. Όταν εκτελούμε το πρόγραμμα από τη γραμμή διαταγών (command prompt), μετά το όνομά του μπορούμε να έχουμε μια λίστα παραμέτρων χωρισμένες με κενά. Μέσα στο σώμα της `main` τις προσπελαύνουμε ως `args[0]`, `args[1]`, ..., `args[args.length-1]` (δείτε Εικόνα 21).
  - Χρησιμοποιείτε τη μέθοδο `s.equals(t)` για να συγκρίνετε δύο συμβολοσειρές `s`, `t`. Αν συγκρίνουμε συμβολοσειρές με τον τελεστή `==` θα είναι πάντα άνισες, διότι είναι στιγμιότυπα διαφορετικών αντικειμένων – περισσότερα γι αυτό στο επόμενο κεφάλαιο.
  - Επειδή οι συμβολοσειρές είναι πίνακες, ο υποδείκτης για τους χαρακτήρες τους αρχίζει από 0 και φτάνει ως `length()`-1. Παρατηρήστε ότι `length` ενός `String` είναι μέθοδος (όπως δηλώνεται από τις παρενθέσεις που ακολουθούν), ενώ `length` ενός πίνακα είναι ιδιότητα.
  - Κατά τη χρήση διαφόρων μεθόδων συμβολοσειρών πρέπει να θυμόμαστε ότι η μέτρηση για το πρώτο γράμμα της συμβολοσειράς ξεκινάει από το 0 – για παραδείγματα δείτε στην Εικόνα 19.
  - Η μέθοδος `String.valueOf` μετατρέπει άλλους τύπους δεδομένων σε συμβολοσειρές.
  - Η κλάση `String` χειρίζεται αμετάβλητα αντικείμενα συμβολοσειρών, δηλαδή κάθε φορά που κάτι αλλάζουμε πάνω σε ένα `String` δημιουργείται άλλο `String` για να αποθηκεύσει το αποτέλεσμα. Για συμβολοσειρές που μεταβάλλονται χρησιμοποιούμε την κλάση `StringBuffer`. Με τέτοια αντικείμενα μπορούμε να προσθέσουμε χαρακτήρα στο τέλος – `append (char c)`, να εισάγουμε ένα χαρακτήρα σε συγκεκριμένη θέση – `insert(int index, char c)`, να αλλάξουμε χαρακτήρα σε συγκεκριμένη θέση – `setChar(int index, char c)`, κά. Τα αποτελέσματα των ενεργειών αυτών τοποθετούνται (μεταβάλλουν) το αρχικό αντικείμενο. Έτσι εξοικονομούμε χώρο μνήμης και χρόνο εκτέλεσης, αφού δεν δημιουργούνται νέα αντικείμενα.

```
String s = "καλημερα";
boolean b = s.equals("καληΜερα"); // false
int i = s.length();               // 8
String t = s.substring(0, 4);     // "καλη"
char c = t.charAt(2);             // 'λ'
String u = t.replace('λ', 'κ');   // "κακη"
int j = s.indexOf('μ');           // 4
String v = String.valueOf(Math.PI);
```

**Εικόνα 19 Μέθοδοι συμβολοσειρών**

## 1.12 Ορισμός μεθόδων

- Στο εσωτερικό μιας κλάσης μπορούμε να ορίσουμε μια ή περισσότερες μεθόδους. Κάθε μέθοδος ξεκινάει με την επικεφαλίδα της και ακολουθεί το σώμα της που περιέχει τις εντολές της.
  - Στην επικεφαλίδα δηλώνουμε τουλάχιστον (α) τον τύπο του αποτελέσματος που επιστρέφεται, (β) το όνομά της και (γ) τη λίστα των τυπικών παραμέτρων που δέχεται – με τύπο και όνομα, όπως ακριβώς και οι συναρτήσεις όλων των γλωσσών προγραμματισμού. Η σύνταξη είναι:

    *[εμβέλεια] [τροποποιητής] τύποςΑποτελέσματος όνομαΜεθόδου* **(** *λίσταΠαραμέτρων* **) {**
    *σώμαΕντολώνΜεθόδου*
    **}**
  - Η *εμβέλεια* καθορίζει από πού είναι ορατή η μέθοδος (περισσότερα στην ενότητα 2.4). *τροποποιητής* μπορεί να είναι η δεσμευμένη λέξη **static**. Προς το παρόν, όλες οι μέθοδοι που θα ορίζουμε θα είναι **static** – στο επόμενο κεφάλαιο θα δούμε ποιες μέθοδοι δεν δηλώνονται ως **static**.
  - Κατά σύμβαση, τα ονόματα των μεθόδων αρχίζουν με πεζό γράμμα και αν περιλαμβάνουν πολλές λέξεις τις χωρίζουμε γράφοντας με κεφαλαίο το πρώτο γράμμα της καθεμιάς.
  - Η *λίσταΠαραμέτρων* αποτελείται από 0, 1 ή περισσότερα ονόματα παραμέτρων χωρισμένα με κόμματα. Τυπικά η σύνταξή της είναι της μορφής:

    *[τύπος όνομαΤυπικήςΠαραμέτρου] [ , τύπος όνομαΤυπικήςΠαραμέτρου ]**
  - Στην περίπτωση που επιστρέφεται κάτι, χρησιμοποιούμε τη δεσμευμένη λέξη **return** για να τερματίσουμε την εκτέλεση της μεθόδου και να επιστρέψουμε την τιμή. Μια από τις συμβάσεις του δομημένου προγραμματισμού προτρέπει να υπάρχει μόνον μια **return** (ένα σημείο επιστροφής) και να βρίσκεται στο τέλος του κειμένου της μεθόδου. Παραδείγματα:

```
static float mesosOros (float x, float y) {
    return (x + y) / 2
}
static void printMessage(String someMessage) {
    System.out.println("> " + someMessage);
}
```

- Αν η μέθοδος δηλωθεί ως **void**, όπως η γνωστή `main`, δεν επιστρέφει κάποια τιμή. Αντιστοιχεί στις υποροutίνες - subroutines (σε αντιπαράθεση με τις συναρτήσεις - functions) των δομημένων γλωσσών προγραμματισμού. Στο σώμα της μεθόδου, εάν υπάρχει εντολή **return**, συντάσσεται χωρίς τιμή. Γενικά, η **return** τερματίζει την εκτέλεση της μεθόδου προκαλώντας επιστροφή μετά το σημείο στο οποίο κλήθηκε.

- Η Java επιτρέπει σε μια μέθοδο να καλεί τον εαυτό της. Με την **αναδρομή** μπορούμε να επαναλαμβάνουμε κώδικα, χωρίς να χρησιμοποιήσουμε εντολές επανάληψης.

```
static int anadrom(int n) { //αθροισμα 1 + 2 + 3 + … + (n - 1) + n
    if (n == 0) return(0); else return (n + anadrom(n - 1));
}
```

## 1.13 Ασκήσεις

**Α1.** Μια φράση ονομάζεται καρκινική εάν διαβάζεται το ίδιο κανονικά και ανάποδα[10]. Φτιάξτε πρόγραμμα που θα διαβάζει μια φράση και θα προσδιορίζει αν είναι καρκινική. Υλοποιήστε δύο αλγορίθμους που λύνουν το πρόβλημα.

**Α2.** (α) Φτιάξτε πρόγραμμα που να υπολογίζει εάν ένα έτος είναι δίσεκτο. Υπενθυμίζεται ότι τα έτη που διαιρούνται με το 100 δεν είναι δίσεκτα, ενώ αυτά που διαιρούνται με το 400 είναι. (β) Ενσωματώστε τη λογική του προγράμματος σε μια στατική μέθοδο και καλείτε την για να υπολογίσετε πόσες ημέρες μεσολαβούν ανάμεσα σε δύο έτη.

```
import javax.swing.JOptionPane;
public class LeapYear {
  public static void main(String[] args) {
    String sYear = JOptionPane.showInputDialog(
                                  "Δώσε το έτος:");
    int iYear = Integer.parseInt(sYear);
    boolean bLeap = false;
    if (iYear %   4 == 0) bLeap = true;
    if (iYear % 100 == 0) bLeap = false;
    if (iYear % 400 == 0) bLeap = true;
    JOptionPane.showMessageDialog(null, "Το " +
    iYear + (bLeap?" ":" δεν ") + "είναι δίσεκτο.");
  }
}
```

**Εικόνα 20 Δίσεκτα έτη (α)**

**Α3.** Υπολογίστε την ηλικία ενός ανθρώπου εάν γνωρίζετε τα γενέθλιά του. Ανάλογα με μια παράμετρο, θα επιστρέφεται η ηλικία σε έτη, ή σε μήνες ή σε έτη και μήνες.

**Α4.** Γράψτε ένα πρόγραμμα που προσεγγίζει σταδιακά την τιμή του e = 2.718…. (βάση νεπέρειων λογαρίθμων) χρησιμοποιώντας τον τύπο $e = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \ldots$. Τυπώστε το πλήθος των όρων του αθροίσματος που συνυπολογίσατε και σε ποιο δεκαδικό ψηφίο διαφέρει η προσέγγισή σας σε σχέση με το `Math.E`.

**Α5.** Διαβάστε ένα χαρακτήρα από τους T, P, K που υποδηλώνουν ορθογώνιο τρίγωνο, ορθογώνιο παραλληλόγραμμο και κύκλο και ανάλογα με το σχήμα διάβασε τις κατάλληλες διαστάσεις και υπολόγισε εμβαδόν και περίμετρό του. Επεκτείνετε το πρόγραμμα να επαναλαμβάνεται από την αρχή μέχρι να διαβάσει τον χαρακτήρα E (Έξοδος), οπότε θα τερματίζει. Κατασκευάστε το πρόγραμμα με τέτοιο τρόπο που να είναι εύκολο να προσθέσετε υπολογισμούς και για άλλα γεωμετρικά σχήματα.

**Α6.** Γράψτε ένα πρόγραμμα που θα χρησιμοποιεί ένα βρόχο για να τυπώνει τα πολλαπλάσια του 2. Ανάλογα με τον ακέραιο τύπο μεταβλητής που χρησιμοποιείται (**byte**, **short**, **int**, **long**), σε ποια τιμή εκθέτη το πρόγραμμα σταματά να τυπώνει σωστά αποτελέσματα; Γιατί;

```
public class ExpoClass {
  public static void main(String[] args) {
    int n = 30;
    if (args.length == 1)
      n = Integer.parseInt(args[0]);
    for (int i = 0; i <= n; i++)
      System.out.println(i + " " + power2(i));
  }

  // υψώνει το 2 εις την δύναμη k, όπου k>=0
  static long power2(int k) {
    long l = 1L; // 2 εις την 0 κανει 1
    for (long i = 1; i <= k; i++)
      l *= 2L;
    return l;
  }
}
```

**Εικόνα 21 Δυνάμεις του 2**

**Α7.** Διαβάστε τις ώρες που έχει δουλέψει ένας εργαζόμενος και το ωρομίσθιό του. Τυπώστε τις μεικτές αποδοχές του (=ώρες Χ ωρομίσθιο) και τις καθαρές αποδοχές του (=μεικτές – κρατήσεις). Μέχρι €800 δεν γίνονται κρατήσεις, ενώ από εκεί και πάνω οι κρατήσεις είναι 20% του υπερβάλλοντος ποσού). Επεκτείνετε το πρόγραμμα να διαβάζει στοιχεία για πολλούς εργαζόμενους και να τερματίζει όταν του δοθεί ότι κάποιος δούλεψε (-1) ώρες. Τότε θα τυπώνει: πλήθος εργαζομένων, σύνολο ωρών που δού-

[10] «νιψονανομηματαμημονανοψιν».

λεψαν, συνολικό κόστος μισθοδοσίας, πόσο κρατήσεων και μέσο καθαρό ημερομίσθιο.

**Α8.** Ο αλγόριθμος του Ευκλείδη χρησιμοποιεί αφαιρέσεις για την εύρεση του μέγιστου κοινού διαιρέτη. Ένας άλλος αλγόριθμος που λύνει το ίδιο πρόβλημα χρησιμοποιώντας διαιρέσεις, λέει: $\text{ΜΚΔ}(x,y)=\begin{cases} x, y=0 \\ \text{ΜΚΔ}(y, x\%y), y\neq 0 \end{cases}$, όπου % είναι ο τελεστής της Java για το υπόλοιπο της ακέραιας διαίρεσης.

**Α9.** Γράψτε πρόγραμμα που να τυπώνει τον πίνακα αληθείας της δυαδικής παράστασης p και q ή r.

**Α10.** Ρίξτε ένα «ζάρι» 6000 φορές και ελέγξτε την συχνότητα εμφάνισης του αποτελέσματος φτιάχνοντας ένα ραβδόγραμμα. Για μια υλοποίηση, δείτε την ενότητα 1.15 Γεννήτρια τυχαίων αριθμών.

**Α11.** Τα σωστά αποτελέσματα ενός διαγωνίσματος με 10 ερωτήσεις διπλής επιλογής δίνονται με μια ακολουθία `L` και `S`, πχ `LSSLLLSSLS`. Γράψτε πρόγραμμα που να διαβάζει τις απαντήσεις των φοιτητών κωδικοποιημένες όπως πρωτύτερα και να υπολογίζει τον βαθμό τους. Το πρόγραμμα θα τερματίζει όταν δοθεί η ακολουθία `XXX` και θα τυπώνει το πλήθος και τον μέσο όρο των βαθμολογιών.

**Α12.** Γράψτε πρόγραμμα που να μετατρέπει αριθμό δευτερολέπτων σε μορφή ημέρες, ώρες, λεπτά, δευτερόλεπτα και το αντίστροφο.

**Α13.** Το 1820 ο Charles Babbage ζήτησε οικονομική υποστήριξη για να κατασκευάσει έναν υπολογιστή που θα υπολόγιζε τη συνάρτηση $f(n)=n^2+n+41$, η οποία φαίνεται να παράγει πρώτους αριθμούς όταν εκτιμηθεί για ακέραια *n*. Αληθεύει αυτό;

**Α14.** Διαβάστε μια γραμματοσειρά και τυπώστε το μήκος της χωρίς τα κενά. Επίσης μετρήστε τα διπλά γράμματα που έχει.

```
import javax.swing.JOptionPane;
public class Ora {
  public static void main(String[] args) {
    String sInput = JOptionPane.showInputDialog(null,
                    "Δώσε δευτερόλεπτα: ");
    long lInput = Long.parseLong(sInput);
    long lDays = lInput / 86400; lInput = lInput % 86400;
    long lOres = lInput / 3600;  lInput=lInput % 3600;
    long lLepta = lInput / 60;   lInput = lInput % 60;
    long lDeftera = lInput / 60; lInput = lInput % 60;
    JOptionPane.showMessageDialog(null,
      "Δευτερόλεπτα: " + sInput + "\nΜέρες: " + lDays +
      "\nΩρες: " + lOres + "\nΛεπτά: " + lLepta +
      "\nΔευτερόλεπτα: " + lDeftera);
    System.exit(0);
  } //end main
}
```

**Εικόνα 22 Μετατροπή δευτερολέπτων**

**Α15.** Γράψτε ένα πρόγραμμα που υπολογίζει την τιμή του $e^x$ χρησιμοποιώντας τον τύπο $e^x=1+\frac{x^1}{1!}+\frac{x^2}{2!}+\frac{x^3}{3!}+\ldots$. Συγκρίνετε την τιμή που υπολογίσατε με την τιμή που επιστρέφει η `Math.exp(x)`. Τυπώστε πόσους όρους του αθροίσματος πρέπει να πάρετε για να πετύχετε ακρίβεια 1, 2, …, `D` δεκαδικών ψηφίων.

**Α16.** Οι πυθαγόρειοι αριθμοί είναι ακέραιοι που αντιστοιχούν σε πλευρές ορθογώνιων τριγώνων. Για παράδειγμα, οι 3, 4, 5 είναι πυθαγόρειοι, επειδή $3^2+4^2=5^2$. Βρείτε όλες τις τριάδες πυθαγόρειων με αριθμούς μικρότερους του 500. Χρησιμοποιήστε φωλιασμένους βρόχους για να ελέγχετε για όλες τις πιθανότητες. Αυτό είναι ένα παράδειγμα αλγορίθμου εξαντλητικής αναζήτησης (brute force). Δείτε στην ενότητα 1.16 Πυθαγόρειοι αριθμοί.

**Α17.** Το τελευταίο θεώρημα του Fermat λέει ότι είναι αδύνατο να διαχωρίσεις οποιαδήποτε δύναμη μεγαλύτερη της δεύτερης σε δύο ίδιες δυνάμεις, ή πιο τυπικά: για κάθε $n>2$, η εξίσωση $a^n+b^n=c^n$ δεν έχει λύση για ακέραιους $a,b,c$. [Σημείωση: για $n=2$, υπάρχουν άπειρες λύσεις και ονομάζονται πυθαγόρειοι αριθμοί – δείτε άσκηση Α16.] 'Αποδείξτε' το θεώρημα, ελέγχοντας για κάθε $n\le n_{\max}=50$, και $a,b,c\le 100$. Προτείνετε βελτιώσεις που θα κάνουν το πρόγραμμά σας πιο γρήγορο μετρώντας κάθε φορά τον αριθμό των επαναλήψεων.

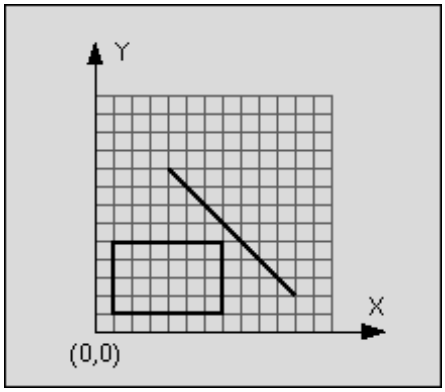
**Α18.** Μέχρι ποιο n η VB και η Java μπορούν να υπολογίσουν το n παραγοντικό (factorial, n!), που ορίζεται ως fac(n) = n * fac(n-1), με εκκίνηση fac(0) = 1, αν το n είναι απλός ακέραιος (Integer, **`int`**); Αν είναι μεγάλος (Long, **`long`**) ακέραιος; Κάντε το ίδιο για τους αριθμούς Fibonacci[11], που ορίζονται fib(n) = fib(n-1) + fib(n-2), με εκκίνηση fib(0)=0, fib(1)=1.

**Α19.** (Διεθνής Διαγωνισμός Προγραμματισμού ACM 1995, http://www.acm.inf.ethz.ch/ProblemSetArchive/B_EU_SWERC/1995/RegionalProblems.html) Γράψτε πρόγραμμα που αποφασίζει αν ένα συγκεκριμένο ευθύγραμμο τμήμα τέμνει συγκεκριμένο ορθογώνιο. Παράδειγμα στην Εικόνα 23:

αρχή γραμμής: (4,9)
τέλος γραμμής: (11,2)
πάνω αριστερά ορθογώνιου: (1,5)

[11] Εάν σας ενδιαφέρει να υπολογίσετε ακόμη μεγαλύτερα παραγοντικά ή αριθμούς Fibonacci, ερευνήστε τις κλάσεις `BigInteger` και `BigDecimal` για αριθμητική ακεραίων οσοδήποτε μεγάλου μεγέθους.

κάτω δεξιά ορθογώνιου: (7,1)
Η γραμμή τέμνει το ορθογώνιο αν η γραμμή και το ορθογώνιο έχουν τουλάχιστον ένα κοινό σημείο. Το ορθογώνιο αποτελείται από τέσσερεις ευθείες γραμμές και την περικλειόμενη περιοχή. Αν και οι τιμές εισόδου είναι ακέραιοι αριθμοί, τα νόμιμα σημεία τομής μπορεί να μην συμπίπτουν με το πλέγμα των ακέραιων συντεταγμένων.



**Εικόνα 23 Γραμμή δεν τέμνει ορθογώνιο**

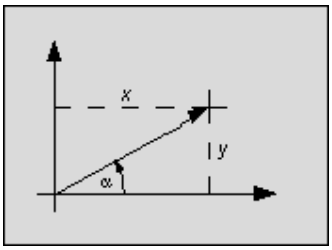
**Α20.** Γράψτε πρόγραμμα που δέχεται ως είσοδο το όνομα ενός ατόμου στη μορφή *Μικρό Μεσαίο Επώνυμο* και στη συνέχεια το τυπώνει στη μορφή *Επώνυμο, Μικρό, Μ.*, όπου "*Μ.*" είναι το αρχικό γράμμα του μεσαίου ονόματος. Για παράδειγμα, η είσοδος `Αναστάσιος Γεωργίου Μυλωνάς` θα πρέπει να παράγει την έξοδο `Μυλωνάς, Αναστάσιος Γ.`

**Α21.** Το τελευταίο ψηφίο κάθε αριθμού ISBN (International Standard Book Number) που συνοδεύει κάθε βιβλίο είναι ψηφίο ελέγχου .που υπολογίζεται από τα υπόλοιπα 9 ψηφία του αριθμού. Ένας ISBN είναι έγκυρος αν η παράσταση $d_1 + 2d_2 + \ldots + 10d_{10}$ είναι πολλαπλάσιο του 11. Φτιάξτε πρόγραμμα που να ελέγχει την εγκυρότητα ISBN. Φτιάξτε πρόγραμμα που δοθέντων των 9 πρώτων ψηφίων θα υπολογίζει το δέκατο.

**Α22.** Γράψτε πρόγραμμα που κάνει τα αρχικά γράμματα κάθε λέξης κεφαλαία και τα υπόλοιπα πεζά. Για παράδειγμα, η είσοδος `εΡΜΟυπολη Συρου` θα πρέπει να δίνει την έξοδο `Ερμουπολη Συρου`.

**Α23.** Ο Ιούλιος Καίσαρας χρησιμοποιούσε την εξής μέθοδο για την κρυπτογράφηση των μηνυμάτων που έστελνε: ωθούσε τα γράμματα n θέσεις προς τα πίσω στο αλφάβητο και όταν εξαντλούνταν τα γράμματα ξεκινούσε από την αρχή του αλφαβήτου, πχ για n=6, η λέξη `ave` γίνεται `gbk`. Φτιάξτε ένα πρόγραμμα που θα δέχεται ως είσοδο τον ακέραιο n και μια συμβολοσειρά και θα την τυπώνει κρυπτογραφημένη. Πώς μπορείτε να χρησιμοποιήσετε το πρόγραμμα που φτιάξατε για να αποκρυπτογραφήσετε; Δείτε στην ενότητα 1.17 Κρυπτογράφηση.

**Α24.** (Ως Α19) Συχνά χρειάζεται να υπολογίζουμε τη γωνία ενός διανύσματος, όπως στην Εικόνα 24. Ο συνηθισμένος τρόπος είναι να χρησιμοποιήσουμε τη μέθοδο `atan` του πακέτου `Math`. Δυστυχώς αυτό δεν λειτουργεί όπως θα περιμέναμε. Η `atan` επιστρέφει τιμές στο διάστημα (-π/2, π/2). Πιο συγκεκριμένα, επιστρέφει ίδια τιμή για τα διανύσματα (x, y) και (-x, -y). Θα προτιμούσαμε μια βελτιωμένη έκδοση που επιστρέφει τη γωνία στο διάστημα [0, 2*π) και διακρίνει τη διαφορά ανάμεσα στα (x, y) και (-x, -y).



**Εικόνα 24 Η γωνία κλίσης ενός διανύσματος**

**Α25.** (Ως Α19) Γράψτε πρόγραμμα που παράγει όλες τις δυνατές "λέξεις" (αναγράμματα) από δοθέν σύνολο γραμμάτων. Για παράδειγμα με είσοδο "`abc`" θα παράγει όλους τους πιθανούς συνδυασμούς τριών γραμμάτων, "`abc`", "`acb`", "`bac`", "`bca`", "`cab`" και "`cba`". Στην είσοδο κάποια γράμματα επιτρέπεται να εμφανίζονται περισσότερες από μία φορές, πχ αν είσοδος είναι "`acba`", έξοδο δίνει "`aabc`", "`aacb`", "`abac`", "`abca`", "`acab`", "`acba`", "`baac`", "`baca`", "`bcaa`", "`caab`", "`caba`" και "`cbaa`". Στην έξοδο κάθε λέξη θα εμφανίζεται μόνο μια φορά.

**Α26.** Γράψτε πρόγραμμα που δέχεται ως στοίχημα έναν αριθμό και ρίχνει ζάρι (τυχαίο αριθμό 1..6). Κερδίζουν τα 3, 5, 6 και χάνουν τα υπόλοιπα. Τυπώνει «Έβαλες στοίχημα 7 φλουριά, έφερες ζάρι 3, ΚΕΡΔΙΣΕΣ, έχεις 14 φλουριά.». Αντίστοιχα, όταν χάνεις. Επεικτείνετε το πρόγραμμα, ώστε με το ξεκίνημα του παιχνιδιού να δέχεται αρχικό ποσό που διαθέτεις. Κατόπιν, βάζεις στοιχήματα, όσο έχεις χρήματα. Το παιχνίδι τελειώνει άμα τα χάσεις όλα, ή βάλεις «στοίχημα» -1. Τότε τυπώνει «Ξεκίνησες με 77 φλουριά, έπαιξες 10 φορές, κέρδισες 60% και έχασες 40% από αυτές. Στο τέλος έχεις 17 φλουριά.».
Επέκταση 1: Το πραγματικό παιχνίδι παίζεται με δυο ζάρια. Ρωτήστε ποιοι συνδυασμοί κερδίζουν και χάνουν και εφαρμόστε τους στο πρόγραμμα. Εμπλουτίστε τα στατιστικά, πχ ποιο ήταν το μεγαλύτερο στοίχημα που μπήκε, ποιο το μεγαλύτερο / μικρότερο ποσό που φτάσατε να κερδίζετε κατά τη διάρκεια του παιχνιδιού, και ό,τι άλλο σας φαίνεται χρήσιμο.
Επέκταση 2: Μια στρατηγική που «πάντα κερδίζει» είναι: ξεκίνα στοιχηματίζοντας, πχ 10 φλουριά. Όταν χάνεις ένα στοίχημα, στο επόμενο στοιχημάτισε τα διπλάσια. Όταν κερδίζεις, στο επόμενο στοιχημάτισε 10 φλουριά. Μελέτησέ την.

## 1.14 Είσοδος / Έξοδος στην τυπική ροή

```
package stdio;
import java.io.*;

/* είσοδος από την τυπική είσοδο System.in (πληκτρολόγιο) και
 * έξοδος στην τυπική έξοδο System.out
 * χωρίς παράθυρα διαλόγου του στιλ swing (JOptionPane.showXDialog)
 */
public class StandardIO {
  public static void main(String[] args) throws IOException {

    BufferedReader stdin = new BufferedReader (new InputStreamReader(System.in));
    String sName;
```

```
    System.out.print("Πώς σε λένε; "); // δεν αλλάζει γραμμή
    System.out.flush(); // για να εμφανιστεί ο δρομέας
    sName = stdin.readLine(); // διαβάζει μια γραμμή

    System.out.println("Γεια σου " + sName + "!");

    System.exit(0);
  }
}
```

## 1.15 Γεννήτρια τυχαίων αριθμών

Λύση της άσκησης Α10

```
public class Zari {

  public static void main(String[] args) {

    System.out.println("ΖΑΡΙ : Έλεγχος γεννήτριας τυχαίων αριθμών");
    if (args.length != 1)
      System.out.println("Δώστε αριθμό επαναλήψεων.");
    else {
      //Τμήμα 1 : είσοδος, αρχικοποίηση
      int iEpanal = Integer.parseInt(args[0]);
      int iApotel[] = new int[7]; //θα χρησιμοποιήσουμε τις θέσεις 1..6
      for (int i = 0; i < iApotel.length; i++) iApotel[i] = 0; //αρχικοποίηση
      int iTemp; //αποτέλεσμα ζαριού

      //Τμήμα 2 : υπολογισμός
      for (int i = 1; i <= iEpanal; i++) {
        iTemp = 1 + (int) (6 * Math.random()); //ρίξιμο του ζαριού
        //System.out.println(iTemp); //εμφάνιση του αποτελέσματος
        iApotel[iTemp]++; //αύξηση του μετρητή κατά 1
      } //end for i

      //Τμήμα 3 : εκτύπωση αποτελεσμάτων
      System.out.println("Αριθμός επαναλήψεων: " + iEpanal);
      String sLine = "";
      for (int i = 1; i <= 6; i++) {
        sLine = i + " ";
        for (int j = 1; j <= iApotel[i]; j++) sLine += "*";
        sLine += " " + iApotel[i];
        System.out.println(sLine);
      } //end for i
    }  //end else if

    System.exit(0);
  } //end main
} //end class
```

## 1.16 Πυθαγόρειοι αριθμοί

Λύση της άσκησης Α16

```
package pythag;
public class Pythag {

  public static void main(String[] args) { // παράδειγμα για χρήση παραμέτρων
    if (args.length != 1)                  // από την γραμμή διαταγών
      System.out.println("Δεν δώσατε το όριο της αναζήτησης.");
    else { //έδωσες σωστά την παράμετρο
      int n = Integer.parseInt(args[0]);
      System.out.println("Έναρξη αναζήτησης πυθαγορείων αριθμών μέχρι " + n);
      System.out.println("Πρώτη λύση");      findPythag1(n);
      System.out.println("Δεύτερη λύση");    findPythag2(n);
      System.out.println("Τρίτη λύση");      findPythag3(n);
      System.out.println("Τέταρτη λύση");    findPythag4(n);
    }
    System.exit(0);
  } //end main

  // οι μεταβλητές a, b αντιστοιχούν πάντα στις δύο κάθετες πλευρές.
  // η μεταβλητή c αντιστοιχεί στην υποτείνουσα

  private static void findPythag1(int n) { //a, b, c <= n
```

```
  // η πιο χαζή (εξαντλητική) λύση
    for (int a=1; a<n; a++) {       // η μια κάθετη πλευρά
      for (int b=1; b<n; b++) {     // η άλλη κάθετη πλευρά
        for (int c=1; c<=n; c++) { // η υποτείνουσα
          if (a*a+b*b == c*c) {
            System.out.println(a + ", " + b + ", " + c);
          } //end if
        } //end for c
      } //end for b
    } //end for a
  } //end findPythag1

  private static void findPythag2(int n) { //a, b <= n
  /* πιο έξυπνη λύση: δεν έχει τριπλό φώλιασμα βρόχων.
   * Απλώς ελέγχει αν κάθε ζευγάρι καθέτων δίνει ακέραια υποτείνουσα.
   * Επίσης θεωρεί οτι η κάθετη b είναι μεγαλύτερη από την a,
   * οπότε αποφεύγει τις διπλές λύσεις 3, 4, 5 και 4, 3, 5
   * Παρατήρηση: η υποτείνουσα μπορεί να είναι μεγαλύτερη του n */
    double dTemp; int c;
    for (int a=1; a<n; a++) {       // η μια κάθετη πλευρά
      for (int b=a; b<n; b++) {     // η άλλη κάθετη πλευρά
        dTemp = Math.sqrt(a*a+b*b);
        c = (int) dTemp;
        if (dTemp == c) {
          System.out.println(a + ", " + b + " , " + c);
        } //end if
      } //end for b
    } //end for a
  } //end findPythag2

  private static void findPythag3(int n) { //a, b <= n
  /* βελτίωση της δεύτερης λύσης: όταν βρει λύση για ένα ζευγάρι i, j
   * σταματάει τον βρόχο του b και συνεχίζει με το επόμενο a */
    double dTemp; int c;
    for (int a=1; a<n; a++) {       // η μια κάθετη πλευρά
      for (int b=a; b<n; b++) { // η άλλη κάθετη πλευρά
        dTemp = Math.sqrt(a*a+b*b);
        c = (int) dTemp;
        if (dTemp == c) {
          System.out.println(a + ", " + b + " , " + c);
          break; //βγες έξω από το εσωτερικό βρόχο b
        } //end if
      } //end for b
    } //end for a
  } //end findPythag3

  private static void findPythag4(int n) { // a, b, c <=n
  /* Μια εναλλακτική προσέγγιση (όχι καλύτερη).
   * Βρίσκει τις υποτείνουσες με ακέραια τετραγωνική ρίζα.
   * Έπειτα ψάχνει για κάθετες */
    double dTemp; int b;
    for (int c=1; c<=n; c++) {    // η υποτείνουσα
      for (int a=1; a<n; a++) {
        dTemp=Math.sqrt(c*c-a*a);
        b = (int) dTemp;
        if (dTemp == b) {
          System.out.println(a + ", " + b + " , " + c);
        } //end if
      } //end for a
    } //end for c
  } //end findPythag4

  /* Συγκριτικά απόδοσης με βάση των αριθμό των επαναλήψεων
   *           n          200           500
   * findPythag1    7.920.200    124.500.500
   * findPythag2       19.701        124.251
   * findPythag3       11.009         68.528 <- ο αποδοτικότερος
   * findPythag4       39.800        249.500
   */

} //end class
```

## 1.17 Κρυπτογράφηση

Λύση της άσκησης Α23

```
package julius;
public class Julius {

  public static void main(String[] args) {
    System.out.println(encrypt(Integer.parseInt(args[0]), // μετατροπή σε ακέραιο
                               args[1].toUpperCase()));   // μετατροπή σε κεφαλαία
    System.exit(0);
  } //end main

  public static String encrypt(int encryptionKey, String plainText) {
    // κρυπτογραφεί κεφαλαία αγγλικά κατά τον αλγόριθμο του Καίσαρα
    // μεταθέτοντας τα ψηφία τόσες θέσεις, όσες λέει το encryptionKey

    char cPlain;    // ο χαρακτήρας που πρόκειται να κωδικοποιηθεί
    int iPlain;     //   και ο κώδικάς του
    int iCipher;    // ο κωδικοποιημένος χαρακτήρας
    char cCipher;   //   και ο κώδικάς του
    String cipherText = ""; //εδώ σωρεύονται χαρακτήρες του κωδικοποιημένου κειμένου

    // εάν το κλειδί είναι εκτός των -25 .. +25, φέρε το μέσα στο εύρος
    encryptionKey = encryptionKey % 26;  // 26 γράμματα έχει το αγγλικό αλφάβητο
    // εάν το κλειδί είναι αρνητικό -26 .. -1 , γίνεται θετικό 0..25
    if (encryptionKey < 0) encryptionKey += 26;
    System.out.println("Κλειδί: " + encryptionKey);

    for (int i=0; i<plainText.length(); i++) { // για κάθε χαρακτήρα της λέξης
      cPlain = plainText.charAt(i);
      if ((cPlain >= 'A') && (cPlain <= 'Z')) { // νόμιμος χαρακτήρας, εντός ορίων
        iPlain = (int) cPlain;
        iCipher = iPlain + encryptionKey;
        if (iCipher > (int) 'Z') { // αν έχεις βγει μετά το Z (τελευταίο γράμμα)
          iCipher = iCipher - 26;  // κατέβα από την αρχή του αλφαβήτου
        } //endif
        cCipher = (char) iCipher;
      } // νόμιμος χαρακτήρας
      else { // εάν βρεθεί κάποιος χαρακτήρας εκτός των A..Z
        cCipher = '?'; // τύπωσε στο κωδικοποιημένο κείμενο ?
        iCipher = (int) cCipher;
      } // end else
      // για έλεγχο και αποσφαλμάτωση
      //System.out.println(cPlain + " " + iPlain + " " + iCipher + " " + cCipher);
      cipherText += cCipher;
    } // end for i
    return cipherText;
  } // end encrypt
} // end class
```