Βασικά στοιχεία προγραμματισμού στη JAVA

Τελεστές, Επαναλήψεις και Παραδείγματα

Οι τελεστές της JAVA

ΕΠΛ233

- Η Java χρησιμοποιεί τους ίδιους τελεστές με την C και την C++, και κατά τον ίδιο γενικά τρόπο.
- Σχεδόν όλοι οι τελεστές εφαρμόζονται **μόνο** σε αρχέγονους τύπους.
- Εξαίρεση αποτελούν οι τελεστές: ‘=‘, ‘==‘ και ‘!=‘, οι οποίοι εφαρμόζονται και σε αντικείμενα.
- Επίσης, οι τελεστές ‘+’ και ‘+=‘ εφαρμόζονται και σε αντικείμενα της κλάσης **String.**
- **Κανόνες προτεραιότητας** (precedence): ο πολλαπλασιασμός και η διαίρεση έχουν προτεραιότητα έναντι της πρόσθεσης και της αφαίρεσης.

Ανάθεση

ΕΠΛ233

- Η ανάθεση σε μεταβλητές αρχέγονων τύπων είναι απλή και ακολουθεί γνωστούς κανόνες.
- Η ανάθεση ενός αντικειμένου **A** σε ένα άλλο **B**, σημαίνει την ανάθεση του χειριστηρίου **a**, του **A**, στο χειριστήριο **b** του **B**: **b = a;**
- Αποτέλεσμα της ανάθεσης αυτής είναι το b και το a να δείχνουν στο ίδιο αντικείμενο (**A**), ενώ η πρόσβαση στο αντικείμενο **B** έχει χαθεί.
- Το φαινόμενο αυτό λέγεται **aliasing** (ψευδωνυμία) και είναι αποτέλεσμα του πως η Java χειρίζεται τα αντικείμενα.
- Η ψευδωνυμία προκύπτει και όταν περνάμε αντικείμενα σαν παραμέτρους σε μια μέθοδο.

Ανάθεση και Ψευδωνυμία

ΕΠΛ233

```
class Number { int i; }
public class Assignment {
    public static void main(String[] args) {
        Number n1 = new Number();
        Number n2 = new Number();
        n1.i = 9;
        n2.i = 47;
        System.out.println("1: n1.i:" + n1.i + ", n2.i: " + n2.i);
        n1 = n2;
        System.out.println("1: n1.i:" + n1.i + ", n2.i: " + n2.i);
        n1.i = 27;
        System.out.println("3: n1.i:" + n1.i + ", n2.i: " + n2.i);
    } }
```

Κλήση Μεθόδων και Ψευδωνυμία

ΕΠΛ233

```
class Letter { char c; }
public class PassObject {
    static void f(Letter y) {
        y.c = 'z';
    }
    public static void main(String[] args) {
        Letter x = new Letter();
        x.c = 'a';
        System.out.println("1: x.c: " + x.c);
        f(x);
        System.out.println("2: x.c: " + x.c);
}}
```

- Σε ορισμένες γλώσσες η μέθοδος **f()** θα έκανε ένα αντίγραφο της παραμέτρου **Letter y,** μέσα στο πεδίο ισχύος (scope) της.
- Στην Java όμως περνάμε το χειριστήριο σαν παράμετρο, κι έτσι μέσα στην **f()** αλλάζουμε το ίδιο το αντικείμενο.

Τελεστές

ΕΠΛ233

- Μαθηματικοί τελεστές: όπως στην C

- **Συσχετιστικοί τελεστές-relational operators**
  - Δημιουργούν αποτελέσματα τύπου **boolean**.
  - Δέχονται κατηγορήματα αρχέγονου τύπου.
  - Οι τελεστές σύγκρισης (>,<,<=, >=) δεν δέχονται κατηγορήματα **boolean.**

Έλεγχος ισοδυναμίας αντικειμένων

ΕΠΛ233

- Για τον έλεγχο των περιεχομένων ενός αντικειμένου για ισοδυναμία με άλλο αντικείμενο, χρησιμοποιείται η μέθοδος **equals**, η οποία υπάρχει σε όλα τα αντικείμενα.

Έλεγχος ισοδυναμίας αντικειμένων

ΕΠΛ233

```
public class EqualsMethod {
    public static void main(String[] args) {
        Integer n1 = new Integer(47);
        Integer n2 = new Integer(47);
        System.out.println(n1.equals(n2));
    } }

class Value { int i; }
public class EqualsMethod2 {
    public static void main(String[] args) {
        Value v1 = new Value();
        Value v2 = new Value();
        System.out.println(v1.equals(v2));
    }}
```

Επιστρέφει true

Επιστρέφει false

Λογικοί Τελεστές

ΕΠΛ233

- Όπως στην C: AND(&&), OR(||), NOT(!)
- Επιστρέφουν **boolean** τιμές.
- Δέχονται σαν κατηγορήματα μόνο **boolean** μεταβλητές.
- **Δεν** επιτρέπεται η χρήση ακέραιων κατηγορημάτων με λογικούς τελεστές (όπως π.χ. στην C): i && j
- **Βραχυκύκλωμα (short circuiting):** κατά την αποτίμηση μιας λογικής έκφρασης (δηλ. έκφρασης με λογικούς τελεστές), η αποτίμηση συνεχίζεται μέχρι να γίνει σαφής η αλήθεια ή το σφάλμα της έκφρασης. Π.χ.:

```
if (test1(0) && test2(2) && test3(2)) ...
```

Τελεστές Δυαδικών Ψηφίων

ΕΠΛ233

- Επιτρέπουν την επεξεργασία των δυαδικών ψηφίων μεταβλητών αρχέγονου **ακέραιου** τύπου (int, long, short, byte, char).
- Εκτελούν πράξεις άλγεβρας bool στα αντίστοιχα bits των κατηγορημάτων τους και δίνουν το αποτέλεσμα.
- Τελεστές: AND(&), OR (|), XOR(^), NOT(~).
- Επίσης: &=, |=, ^= (δεν επιτρέπεται όμως το ~=)
- Οι τύποι **boolean** σε συνδυασμό με τους δυαδικούς τελεστές, αντιμετωπίζονται σαν τιμές ενός bit.
- Η Java προσφέρει επίσης τελεστές διολίσθησης (shift operators), που δέχονται μεταβλητές αρχέγονου **ακέραιου** τύπου (int, long, short, byte, char)
- **Τριαδικός Τελεστής:**
  boolean-exp?value0:value1

Μετατροπή Τύπων - casting

ΕΠΛ233

- Η Java μετατρέπει ένα τύπο δεδομένων σε κάποιον άλλο αυτομάτως, όποτε αυτό χρειαστεί.
- Η μετατροπή τύπων (type casting) μας επιτρέπει να κάνουμε αυτόν τον μετασχηματισμό ρητά ή να τον εξαναγκάσουμε να γίνει, σε περιπτώσεις που δεν γίνεται αυτόματα. Π.χ.:

```
void casts() {
    int i = 200;
    long l = (long) i;  // γίνεται και αυτόματα
    long l2 = (long) 200; // γίνεται κ. αυτόματα
}
```

- Narrowing conversion: δεν γίνεται αυτόματα, και όταν γίνει υπάρχει ο κίνδυνος να χαθεί πληροφορία. Η Java επιτρέπει την μετατροπή από και προς οποιονδήποτε αρχέγονο τύπο εκτός **boolean.**
- Μετατροπή τύπων από μιά κλάση σε κάποια άλλη δεν επιτρέπεται.

sizeof

ΕΠΛ233

- Στην C και στην C++, η συνάρτηση **sizeof()** μας επιστρέφει τον αριθμό των χαρακτήρων που έχουν δεσμευθεί για κάποιο δεδομένο.
- Ο κυριότερος λόγος χρήσης της sizeof() είναι η φορητότητα (portability). Διάφοροι τύποι δεδομένων μπορεί να έχουν διαφορετικό μέγεθος σε διαφορετικές μηχανές. Συνεπώς, ο προγραμματιστής πρέπει να μπορεί να βρει το ακριβές μέγεθος ενός τύπου.
- Στην Java δεν υπάρχει **sizeof()** διότι σε όλα τα μηχανήματα οι τύποι δεδομένων έχουν το ίδιο μέγεθος.

Έλεγχος εκτέλεσης

ΕΠΛ233

- Οι εντολές της Java είναι παρόμοιες με αυτές της C και της C++.
- **if-else:**

```
if (boolean-expression)
        statement
else
        statement
```

- **return:**
  - χρησιμοποιείται για να καθορίσει την τιμή που επιστρέφει μια μέθοδος
  - επιβάλλει την επιστροφή της τιμής αυτής αμέσως

Επαναλήψεις - Iterations

ΕΠΛ233

- **while** (boolean-expression)
  statement
- **do**
  statement
  while (Boolean-expression);
- **for** (initialization;Boolean-expression; step)
  statement

```
public class ListCharacters {
    public static void main(String[] args) {
        for (char c =0; c<128; c++)
          if (c != 26) //ANSI clear screen
                System.out.println("value: " + (int) c + "character: " + c);
        }          }
```

Ο τελεστής «,»

ΕΠΛ233

```
for (int i = 0 , j = 1; i < 10 && j != 11; i++, j++ )
/* body of the loop */
```

- Ο τελεστής «,» μπορεί να χρησιμοποιηθεί μόνο σε εντολές αρχικοποίησης και βήματος τής εντολής **for.**
- Στην αρχικοποίηση τού **for** μπορούμε να ορίσουμε περισσότερες της μιας μεταβλητές, του ίδιου όμως τύπου.

break και continue

ΕΠΛ233

```
public class BreakAndContinue {
  public static void main(String[] args) {
     for (int i=0; i<100; i++) {
      if (I==74) break; // out of loop
      if (I % 9 != 0) continue; // next iteration
      System.out.println(I);
    }
    int I = 0;
    while (true) {
      I++;
      int j = I * 27;
      if (j == 1296) break; // out of loop
      if (I % 10 != 0) continue; // top of loop
      System.out.println(I);
  }}}
```

goto και labels

ΕΠΛ233

- Το **goto** στην Java είναι δεσμευμένη λέξη αλλά δεν χρησιμοποιείται ως εντολή.
- Ωστόσο η Java χρησιμοποιεί **labels**, σε συνδυασμό με εντολές **continue** ή **break** για να επιτρέπει την έξοδο από φωλιασμένους βρόχους.

```
label1:
outer-iteration {
   inner-iteration {
    //…
   break; //breaks out of inner iteration; end us in outer
   iteration
   //…
   continue; //moves back to beginning of inner iteration
   //…
   continue label1; //breaks all the way out to label1;
   reenters outer loop
   //…
   break label1; //breaks all the way out to label1; does not
   reenter loop
     }
}
```

- *Το μόνο σημείο στο οποίο έχει νόημα η χρήση label είναι ακριβώς πριν μια εντολή επανάληψης.*

switch

ΕΠΛ233

```
switch (integral-expression) {
  case integral-value1: statement; break;
  case integral-value2: statement; break;
  case integral-value3: statement; break;
  case integral-value4: statement; break;
     // …
  default: statement;
}
```

- Η χρήση του **break**.
- integral-expression: integer or char.

Μερικές Παρατηρήσεις

Μερικές Παρατηρήσεις

ΕΠΛ233

```
char c = (char) (Math.random() * 26 + 'a');
```

Μερικές Παρατηρήσεις

ΕΠΛ233

```
char c = (char) (Math.random() * 26 + 'a');
```

- Math.random() επιστρέφει **double**.

Μερικές Παρατηρήσεις


```
char c = (char) (Math.random() * 26 + 'a');
```

- Math.random() επιστρέφει **double**.
- Το 26 μετατρέπεται σε double για να γίνει ο πολλαπλασιασμός.

Μερικές Παρατηρήσεις


```
char c = (char) (Math.random() * 26 + 'a');
```

- Math.random() επιστρέφει **double**.
- Το 26 μετατρέπεται σε double για να γίνει ο πολλαπλασιασμός.
- Το 'a' μετατρέπεται σε double για να γίνει η πρόσθεση.

Μερικές Παρατηρήσεις


```
char c = (char) (Math.random() * 26 + 'a');
```

- Math.random() επιστρέφει **double**.
- Το 26 μετατρέπεται σε double για να γίνει ο πολλαπλασιασμός.
- Το 'a' μετατρέπεται σε double για να γίνει η πρόσθεση.
- Το αποτέλεσμα της πρόσθεσης μετασχηματίζεται σε char.

Μερικές Παρατηρήσεις


```
char c = (char) (Math.random() * 26 + 'a');
```

- Math.random() επιστρέφει **double**.
- Το 26 μετατρέπεται σε double για να γίνει ο πολλαπλασιασμός.
- Το 'a' μετατρέπεται σε double για να γίνει η πρόσθεση.
- Το αποτέλεσμα της πρόσθεσης μετασχηματίζεται σε char.
- Πώς;

Μερικές Παρατηρήσεις


```
char c = (char) (Math.random() * 26 + 'a');
```

- Math.random() επιστρέφει **double.**
- Το 26 μετατρέπεται σε double για να γίνει ο πολλαπλασιασμός.
- Το 'a' μετατρέπεται σε double για να γίνει η πρόσθεση.
- Το αποτέλεσμα της πρόσθεσης μετασχηματίζεται σε char.
- Πώς;
- Με αποκοπή.

Μερικές Παρατηρήσεις

Μερικές Παρατηρήσεις

ΕΠΛ233

```
public class CastingNumbers {
```

Μερικές Παρατηρήσεις

ΕΠΛ233

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

Μερικές Παρατηρήσεις

ΕΠΛ233

```
public class CastingNumbers {
  public static void main(String[] args) {
    double above = 0.7, below = 0.4;
```

Μερικές Παρατηρήσεις

ΕΠΛ233

```
public class CastingNumbers {
  public static void main(String[] args) {
    double above = 0.7, below = 0.4;
    System.out.println("above: " + above);
```

Μερικές Παρατηρήσεις

ΕΠΛ233

```
public class CastingNumbers {
  public static void main(String[] args) {
    double above = 0.7, below = 0.4;
    System.out.println("above: " + above);
    System.out.println("below: " + below);
```

Μερικές Παρατηρήσεις

ΕΠΛ233

```
public class CastingNumbers {
  public static void main(String[] args) {
    double above = 0.7, below = 0.4;
    System.out.println("above: " + above);
    System.out.println("below: " + below);
    System.out.println("(int) above: " + (int) above);
```

Μερικές Παρατηρήσεις

ΕΠΛ233

```
public class CastingNumbers {
  public static void main(String[] args) {
    double above = 0.7, below = 0.4;
    System.out.println("above: " + above);
    System.out.println("below: " + below);
    System.out.println("(int) above: " + (int) above);
    System.out.println("(int) below: " + (int) below);
```

Μερικές Παρατηρήσεις

ΕΠΛ233

```
public class CastingNumbers {
  public static void main(String[] args) {
    double above = 0.7, below = 0.4;
    System.out.println("above: " + above);
    System.out.println("below: " + below);
    System.out.println("(int) above: " + (int) above);
    System.out.println("(int) below: " + (int) below);
    System.out.println("(char) 'a' + above: " + (char) ('a' +
  above));
```

Μερικές Παρατηρήσεις

ΕΠΛ233

```
public class CastingNumbers {
  public static void main(String[] args) {
    double above = 0.7, below = 0.4;
    System.out.println("above: " + above);
    System.out.println("below: " + below);
    System.out.println("(int) above: " + (int) above);
    System.out.println("(int) below: " + (int) below);
    System.out.println("(char) 'a' + above: " + (char) ('a' +
  above));
    System.out.println("(char) 'a' + below: " + (char) ('a' +
  below)); } }
```

Μερικές Παρατηρήσεις

ΕΠΛ233

```
public class CastingNumbers {
  public static void main(String[] args) {
    double above = 0.7, below = 0.4;
    System.out.println("above: " + above);
    System.out.println("below: " + below);
    System.out.println("(int) above: " + (int) above);
    System.out.println("(int) below: " + (int) below);
    System.out.println("(char) 'a' + above: " + (char) ('a' +
  above));
    System.out.println("(char) 'a' + below: " + (char) ('a' +
  below)); } }
```

above: 0.7

Μερικές Παρατηρήσεις

ΕΠΛ233

```
public class CastingNumbers {
  public static void main(String[] args) {
    double above = 0.7, below = 0.4;
    System.out.println("above: " + above);
    System.out.println("below: " + below);
    System.out.println("(int) above: " + (int) above);
    System.out.println("(int) below: " + (int) below);
    System.out.println("(char) 'a' + above: " + (char) ('a' +
  above));
    System.out.println("(char) 'a' + below: " + (char) ('a' +
  below)); } }
```

above: 0.7
below: 0.4

Μερικές Παρατηρήσεις

ΕΠΛ233

```
public class CastingNumbers {
  public static void main(String[] args) {
    double above = 0.7, below = 0.4;
    System.out.println("above: " + above);
    System.out.println("below: " + below);
    System.out.println("(int) above: " + (int) above);
    System.out.println("(int) below: " + (int) below);
    System.out.println("(char) 'a' + above: " + (char) ('a' +
  above));
    System.out.println("(char) 'a' + below: " + (char) ('a' +
  below)); } }
```

above: 0.7
below: 0.4
(int) above:0

Μερικές Παρατηρήσεις

ΕΠΛ233

```
public class CastingNumbers {
  public static void main(String[] args) {
    double above = 0.7, below = 0.4;
    System.out.println("above: " + above);
    System.out.println("below: " + below);
    System.out.println("(int) above: " + (int) above);
    System.out.println("(int) below: " + (int) below);
    System.out.println("(char) 'a' + above: " + (char) ('a' +
  above));
    System.out.println("(char) 'a' + below: " + (char) ('a' +
  below)); } }
```

above: 0.7
below: 0.4
(int) above:0
(int) below:0

Μερικές Παρατηρήσεις

ΕΠΛ233

```
public class CastingNumbers {
  public static void main(String[] args) {
    double above = 0.7, below = 0.4;
    System.out.println("above: " + above);
    System.out.println("below: " + below);
    System.out.println("(int) above: " + (int) above);
    System.out.println("(int) below: " + (int) below);
    System.out.println("(char) 'a' + above: " + (char) ('a' +
  above));
    System.out.println("(char) 'a' + below: " + (char) ('a' +
  below)); } }
```

above: 0.7
below: 0.4
(int) above:0
(int) below:0
(char) ('a' + above): a

Μερικές Παρατηρήσεις

ΕΠΛ233

```
public class CastingNumbers {
  public static void main(String[] args) {
    double above = 0.7, below = 0.4;
    System.out.println("above: " + above);
    System.out.println("below: " + below);
    System.out.println("(int) above: " + (int) above);
    System.out.println("(int) below: " + (int) below);
    System.out.println("(char) 'a' + above: " + (char) ('a' +
  above));
    System.out.println("(char) 'a' + below: " + (char) ('a' +
  below)); } }
```

above: 0.7
below: 0.4
(int) above:0
(int) below:0
(char) ('a' + above): a
(char) ('a' + below): a

Πίνακες στη JAVA

Πίνακες στην JAVA

ΕΠΛ233

- Οι πίνακες στην JAVA έχουν σχεδιαστεί κατά τρόπον ώστε να ξεπερνιούνται οι δυσκολίες του προγραμματισμού πινάκων της C/C++.
- Ένας πίνακας JAVA είναι εξασφαλισμένο ότι θα αρχικοποιηθεί και ότι δεν θα γίνει πρόσβαση εκτός των ορίων του.
- Τα χαρακτηριστικά αυτά υλοποιούνται με κάποιο σχετικό κόστος μνήμης και χρόνου εκτέλεσης (κατά την εκτέλεση γίνεται έλεγχος κατά πόσον δεν γίνεται υπέρβαση των ορίων του πίνακα).
- Κατά τη δημιουργία ενός πίνακα, κατ' ουσίαν δημιουργείται ένας πίνακας χειριστηρίων (Handles), τα οποία αρχικοποιούνται σε null.
- Είναι ευθύνη του προγραμματιστή να αρχικοποιήσει σωστά τα χειριστήρια, ώστε να παραπέμπουν σε αντικείμενα.

Πίνακες

ΕΠΛ233

- Πίνακες στη Java: ακολουθίες από αντικείμενα ή μεταβλητές αρχέγονου τύπου, οι οποίες ομαδοποιούνται κάτω από το ίδιο όνομα.
- Δήλωση πινάκων:

```
int[] a ;
int a[];
```

- Παρατηρείστε ότι ο μεταφραστής δεν επιτρέπει στον προγραμματιστή να δηλώσει το μέγεθος του πίνακα .
- Κατ ’ ουσίαν, στο σημείο της δήλωσης δημιουργείται μόνο το χειριστήριο του πίνακα.
- Για την κράτηση μνήμης για τον πίνακα, χρειάζεται να γράψουμε μια έκφραση *αρχικοποίησης*, η οποία μπορεί να βρίσκεται οπουδήποτε στον κώδικα (πριν το σημείο στο οποίο γίνεται χρήση του πίνακα).

Αρχικοποίηση Πινάκων (συνέχεια)

- Αρχικοποίηση ενός πίνακα μπορεί να γίνει και στο σημείο της δήλωσής του, ως εξής: **int[] a = { 1 , 2, 3, 4, 5 };**

```
public class Arrays {
    public static void main(String[] args) {
        int[] a1 = { 1, 2, 3, 4, 5 };
        int[] a2;
        a2 = a1;
        for(int i=0; i<a2.length; i++)  a2[i]++;
        }
}
```

- Το στοιχείο length υπάρχει σε όλους τους πίνακες και δίνει τον αριθμό των στοιχείων τους (δεν επιτρέπεται ανάθεση σε αυτό).
- Σε περίπτωση που προσπαθήσουμε να ξεπεράσουμε το μήκος ενός πίνακα, λαμβάνουμε εξαίρεση την στιγμή της εκτέλεσης (**exception**).

Αρχικοποίηση Πινάκων (συνέχεια)

- Εαν δεν γνωρίζουμε το μήκος ενός πίνακα πριν την εκτέλεση του προγράμματός μας, θα πρέπει να χρησιμοποιήσουμε τη **new** για την κράτηση τής μνήμης του πίνακα, κατά την εκτέλεση του προγράμματος:

```
import java.util.*;
public class ArrayNew {
   static Random rand=new Random();
   static int pRand(int mod) {
         return Math.abs(rand.nextInt())%mod+1;
   }
   public static void main(String[] args) {
         int[] a = new int[pRand(20)];
         for(int i=0;i<a.length; i++)
               System.out.println(a[i]);
   }
}
```

Αρχικοποίηση Πινάκων (συνέχεια)

ΕΠΛ233

- Σε περίπτωση πινάκων με στοιχεία *μή-αρχέγονου τύπου*, πρέπει να χρησιμοποιείται η **new** για την αρχικοποίηση των στοιχείων του πίνακα.

```
import java.util.*;
public class ArrayClassObj {
        static Random rand = new Random();
        static int pRand(int mod) {
                return Math.abs(rand.nextInt()) % mod + 1;
        }
public static void main(...)  {
        int q, p = pRand(20);
        Integer[] a=new Integer[p];
        for(int i=0;i<a.length;i++){
                        q = pRand(500);
                        a[i] = new Integer(q);
        }
}
```

Αρχικοποίηση Πινάκων (συνέχεια)

ΕΠΛ233

```
public class ArrayInit {
    public static void main(String[] args) {
        Integer[] a = {
            new Integer(1), new Integer(2), new Integer(3),
        };
        Integer[] b = new Integer[]  {
            new Integer(1), new Integer(2), new Integer(3),
        }; }}
```

- Το τελικό ‘,’ ΄στην αρχικοποίηση των πινάκων είναι προαιρετικό.
- Η δεύτερη συντακτική μορφή αρχικοποίησης επιτρέπει την υλοποίηση και κλήση μεθόδων που δέχονται άγνωστο αριθμό παραμέτρων, αγνώστου τύπου (λειτουργικότητας αντίστοιχης με τα «variable argument lists» varargs - της C.) Π.χ.:

```
f(new Object[] {new Integer(23), “two”, “three”, });
```

Παράδειγμα

ΕΠΛ233

```
class A { int i; }
public class VarArgs {
  static void print(Object[] x) {
     for(int i = 0; i < x.length; i++)
                          System.out.println(x[i]);
  }
  public static void main(String[] args) {
     print(new Object[] { new Integer(47),
                          new VarArgs(),
                          new Float(3.14),
                          new Double(1.1)});
     print(new Object[] {"one", "two", "three" });
     print(new Object[] {new A(), new A(), new A()});
  }
}
```

Πολυδιάστατοι Πίνακες

ΕΠΛ233

```
int[ ][ ]  a1 = {{ 1, 2, 3, },{ 4, 5, 6, },};

// 3-D array with fixed length:
int[ ][ ][ ] a2 =new int[2][2][4];
for(int i = 0; i<a2.length; i++)
   for(int j=0;j<a2[i].length;j++)
         for(int k=0;k<a2[i][j].length;k++) {   }
```

Πολυδιάστατοι Πίνακες

ΕΠΛ233

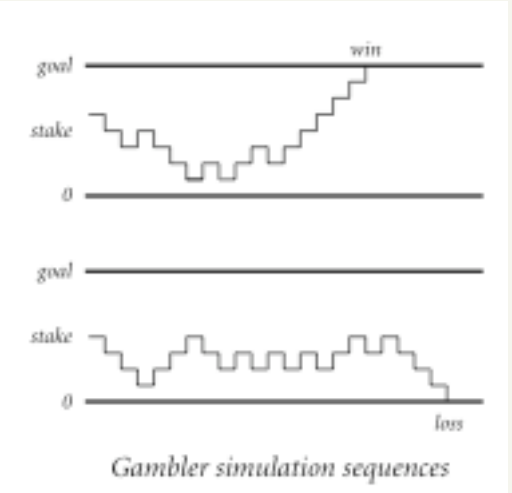
```
// 3-D array with varied-length // vectors:
   int[ ][ ][ ]   a3 = new int[q][ ][ ];
        for(int i = 0;i<a3.length; i++) {
                a3[i] = new int[p][];
                for(int j=0;j<a3[i].length; j++)
                        a3[i][j] = new int[r];
        }
   Integer[ ][ ] a4 = {
                { new Integer(1),new Integer(2)},
                { new Integer(3),new Integer(4)},
                { new Integer(5),new Integer(6)},
   };
```

Μερικά απλά υπολογιστικά προβλήματα

Η καταστροφή του χαρτοπαίκτη

ΕΠΛ233

- Suppose a gambler makes a series of fair $1 bets, starting with $50, and continue to play until she either goes broke or has $250.
- What are the chances that she will go home with $250, and how many bets might she expect to make before winning or losing?



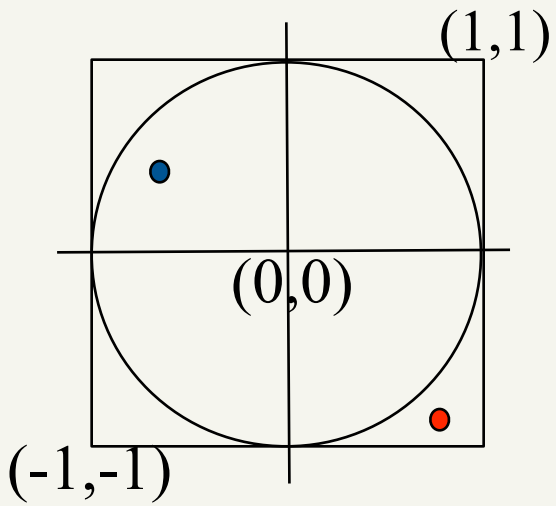
Gambler simulation sequences

```
public class Gambler {

public static void main(String[] args) {
    int stake = Integer.parseInt(args[0]);    // gambler's stating bankroll
    int goal  = Integer.parseInt(args[1]);    // gambler's desired bankroll
    int N     = Integer.parseInt(args[2]);    // number of trials to perform

    int bets = 0;         // total number of bets made
    int wins = 0;         // total number of games won

    // repeat N times to take average
    for (int i = 0; i < N; i++) {

        // do one gambler's ruin simulation
        int t = stake;
        while (t > 0 && t < goal) {
            bets++;
            if (Math.random() < 0.5) t++;     // win $1
            else                     t--;     // lose $1
        }
        if (t == goal) wins++;         // did gambler go achieve desired goal?
    }

    // print results
    System.out.println(wins + " wins of " + N);
    System.out.println("Percent of games won = " + 100.0 * wins / N);
    System.out.println("Avg # bets           = " + 1.0 * bets / N);
}}
```

Τυχαία σημεία εντός κύκλου

ΕΠΛ233

- Υπολογίστε ένα τυχαίο σημείο που βρίσκεται εντός κύκλου με ακτίνα ίση με τη μονάδα.

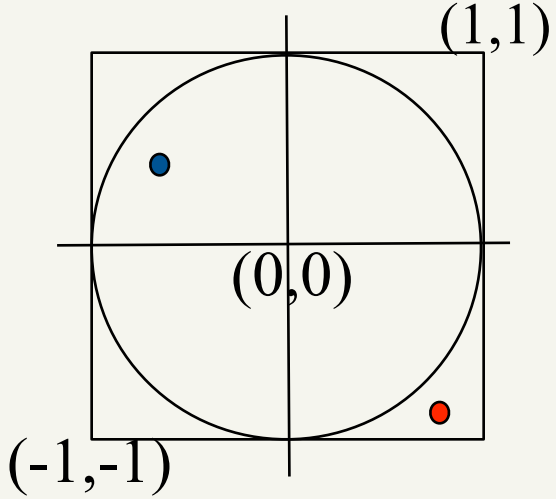


`static double`	**`random()`** Returns a `double` value with a positive sign, greater than or equal to `0.0` and less than `1.0`.

Τυχαία σημεία εντός κύκλου

ΕΠΛ233

- Υπολογίστε ένα τυχαίο σημείο που βρίσκεται εντός κύκλου με ακτίνα ίση με τη μονάδα.




```
double x, y, r;
do {
   x = 2.0 * Math.random() - 1.0;
   y = 2.0 * Math.random() - 1.0;
   r = x*x + y*y;
} while (r > 1);
```

```
static double | random()
```
Returns a `double` value with a positive sign, greater than or equal to `0.0` and less than `1.0`.

Μέθοδος των Newton/Raphson

Μέθοδος των Newton/Raphson

ΕΠΛ233

- Πώς υπολογίζουμε την τετραγωνική ρίζα αριθμών;

Μέθοδος των Newton/Raphson

ΕΠΛ233

- Πώς υπολογίζουμε την τετραγωνική ρίζα αριθμών;
- Παρατήρηση: βρίσκοντας τη λύση της εξίσωσης:
  - $f(x) = x^2 - c$

Μέθοδος των Newton/Raphson

ΕΠΛ233

- Πώς υπολογίζουμε την τετραγωνική ρίζα αριθμών;
- Παρατήρηση: βρίσκοντας τη λύση της εξίσωσης:
  - $f(x) = x^2 - c$
- Το πρόβλημα ανάγεται στον υπολογισμό των ριζών συναρτήσεων.

Μέθοδος των Newton/Raphson

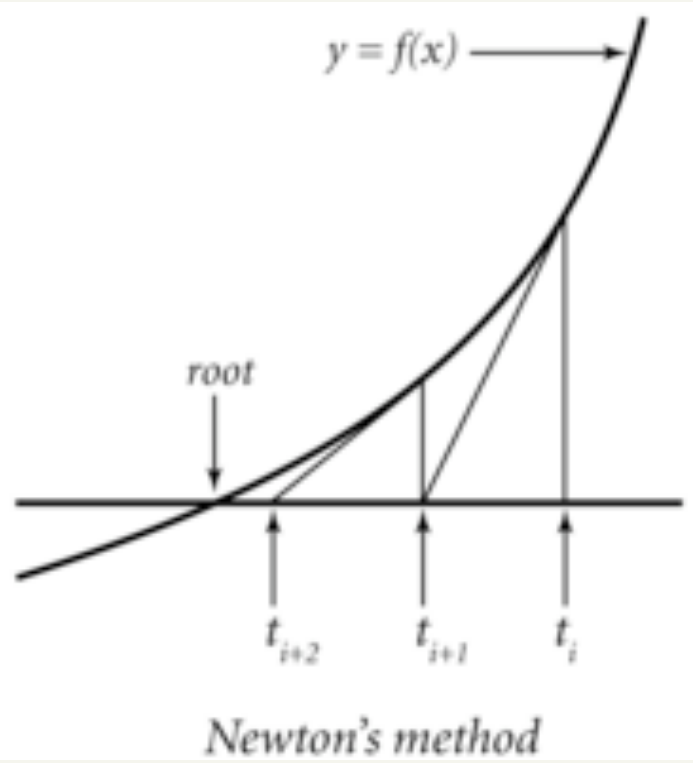
ΕΠΛ233

- Πώς υπολογίζουμε την τετραγωνική ρίζα αριθμών;
- Παρατήρηση: βρίσκοντας τη λύση της εξίσωσης:
  - $f(x) = x^2 - c$
- Το πρόβλημα ανάγεται στον υπολογισμό των ριζών συναρτήσεων.
- Με ποιόν αλγόριθμο υπολογίζουμε την ρίζα μιας συνάρτησης;

Μέθοδος των Newton/Raphson

ΕΠΛ233

- Ξεκινάμε από μια εκτίμηση της ρίζας της συνάρτησης: a
- Μια καλύτερη εκτίμηση της ρίζας είναι η τιμή:
  - x = a - f(a)/f'(a)



Newton's method

```
public class Newton {

// return the square root of c, computed using Newton's method
public static double sqrt(double c) {
    if (c < 0) return Double.NaN;
    double EPS = 1E-15;
    double t = c;
    while (Math.abs(t - c/t) > EPS*t)
        t = (c/t + t) / 2.0;
    return t;
}

// test client
public static void main(String[] args) {
    // parse command-line parameters
    double[] a = new double[args.length];
    for (int i = 0; i < args.length; i++) {
        a[i] = Double.parseDouble(args[i]);
    }

    // compute square root for each command line parameter
    for (int i = 0; i < a.length; i++) {
        double x = sqrt(a[i]);
        StdOut.println(x);
    }   } }
```

Εισαγωγή στην χαρτοπαιξία!

ΕΠΛ233

- Μια τράπουλα αποτελείται από:
  - 52 χαρτιά
  - Τέσσερεις οικογένειες “χρώματα” χαρτιών:
    - σπαθί (clubs) ♣
    - καρρώ (diamonds) ♦
    - καρδιές (hearts) ♥
    - πίκα (spades)♠
  - Σε κάθε οικογένεια, υπάρχουν 13 χαρτιά, με την ακόλουθη διαβάθμιση: 2, 3, 4, 5, 6, 7, 8, 9, 10, Αξιωματικός (Jack), Βασίλισσα (Queen), Βασιλιάς (King), Άσσος (Ace)

Η τράπουλα σε πίνακες..

ΕΠΛ233

```
String[] suit = { “Clubs”, “Diamonds”, “Hearts”, “Spades”};
String[] rank = { “2”, “3”, “4”, “5”, “6”, “7”, “8”, “9”, “10”,
                  “Jack”, “Queen”, “King”, “Ace” };

String[] deck = new String[52];
for (int i = 0; i<13; i++)
    for (int j = 0; j<4; j++)
        deck[4*i+j] = rank[i] + “ of ” + suit[j];
```

Ανακάτεμα της τράπουλας

ΕΠΛ233

```
int N = deck.length;

for (int i=0; i<N; i++) {
    int r = i + (int) (Math.random() * (N-i));
    String t = deck[i];
    deck[i] = deck[r];
    deck[r] = t;
 }
```

Ανακάτεμα της τράπουλας

ΕΠΛ233

```
int N = deck.length;

for (int i=0; i<N; i++) {
    int r = i + (int) (Math.random() * (N-i));
    String t = deck[i];
    deck[i] = deck[r];
    deck[r] = t;
 }
```

επιστρέφει τυχαίο αριθμό μεταξύ i και N

Δειγματοληψία χωρίς αντικατάσταση

ΕΠΛ233

- Θέλουμε να πάρουμε ένα τυχαίο δείγμα από ένα σύνολο αριθμών, έτσι ώστε κάθε στοιχείο του συνόλου να εμφανίζεται το πολύ μια φορά στο δείγμα. Π.χ.:
  - Για το σύνολο: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15
  - Μια δειγματοληψία 5 αριθμών χωρίς αντικατάσταση, είναι η: 2, 5, 10, 4, 14

- Γράψτε ένα πρόγραμμα που να διαβάζει δύο ακεραίους M και N από την είσοδο και επιστρέφει δημιουργεί τη μετάθεση ενός δείγματος μεγέθους M από το σύνολο των αριθμών 0..N-1

```
public class Sample {
   public static void main(String[] args) {
      int M = Integer.parseInt(args[0]);    // choose this many elements
      int N = Integer.parseInt(args[1]);    // from 0, 1, ..., N-1

      // create permutation 0, 1, ..., N-1
      int[] perm = new int[N];
      for (int i = 0; i < N; i++)
         perm[i] = i;

      // create random sample in perm[0], perm[1], ..., perm[M-1]
      for (int i = 0; i < M; i++)  {

         // random integer between i and N-1
         int r = i + (int) (Math.random() * (N-i));

         // swap elements at indices i and r
         int t = perm[r];
         perm[r] = perm[i];
         perm[i] = t;
      }

      // print results
      for (int i = 0; i < M; i++)
         System.out.print(perm[i] + " ");
      System.out.println();
   }
}
```

Τυχαίος περίπατος αυτοαποφυγής

ΕΠΛ233

- Γράψτε ένα πρόγραμμα που:
  - Δέχεται στην είσοδο δύο ακεραίους N και T
  - Υπολογίζει T τυχαίους περίπατους αυτοαποφυγής (self-avoiding random walks)
  - Εκτυπώνει το ποσοστό των τυχαίων περιπάτων που καταλήγουν σε αδιέξοδο
- Τι είναι τυχαίος περίπατος;
  - Ξεκινάμε από ένα σημείο στο επίπεδο και κινούμαστε πάνω ή κάτω ή αριστερά ή δεξιά, ένα βήμα τη φορά
  - Σε κάθε βήμα, η επιλογή της κατεύθυνσης είναι τυχαία
- Τ.Π. αυτοαποφυγής:
  - Αποφεύγουμε σημεία που τα έχουμε ήδη επισκεφθεί (ο τυχαίος περίπατος δεν τέμνει τον εαυτό του)

Επεξεργασία Χρωμάτων

ΕΠΛ233

- Βιβλιοθήκη της JAVA η οποία περιλαμβάνει τον ορισμό της κλάσης Color: java.awt.Color
- Οι τιμές των χρωμάτων αναπαριστώνται στο σύστημα RGB (red, green blue):
  - Κάθε χρώμα ορίζεται από τρεις ακεραίους αριθμούς (μεταξύ 0 και 255), οι οποίοι αντιστοιχούν στην ένταση του κόκκινου, πράσινου και μπλε χρώματος
- Η Color διαθέτει κατασκευαστή ο οποίος δέχεται ως παραμέτρους τρεις ακεραίους (τιμές RGB):
  - Color red = new Color(255,0,0);

red	*green*	*blue*	
255	0	0	*red*
0	255	0	*green*
0	0	255	*blue*
0	0	0	*black*
100	100	100	*dark gray*
255	255	255	*white*
255	255	0	*yellow*
255	0	255	*magenta*
160	82	45	*sienna*

Some color values

Color API

ΕΠΛ233

public class java.awt.Color (Java color data type)		
	Color(int r, int g, int b)	
int	getRed()	*red intensity*
int	getGreen()	*green intensity*
int	getBlue()	*blue intensity*
Color	brighter()	*brighter version of this color*
Color	darker()	*darker version of this color*
String	toString()	*string representation of this color*
boolean	equals(Color c)	*is this color's value the same as c's?*

http://java.sun.com/j2se/1.5.0/docs/api/java/awt/Color.html

Φωτεινότητα - luminance

ΕΠΛ233

- Η ποιότητα μιας ψηφιακής εικόνας που εμφανίζεται σε οθόνη εξαρτάται από την μονοχρωματική φωτεινότητα (monochromatic luminance).
- Η φωτεινότητα προκύπτει από την ευαισθησία του ματιού στα χρώματα R, G, B και υπολογίζεται από την ακόλουθη σχέση:
  - Y = 0.299 r + 0.587 g + 0.114 b, όπου r, g, b είναι η ένταση των χρωμάτων κόκκινο, πράσινο, μπλε
- Η φωτεινότητα προσδιορίζει και την συμβατότητα μεταξύ χρωμάτων προσκηνίου και παρασκηνίου:
  - αν η διαφορά της φωτεινότητας προσκηνίου και παρασκηνίου είναι λιγότερη από 128, τότε θεωρείται ότι τα χρώματα είναι ασύμβατα μεταξύ τους

Φωτεινότητα

ΕΠΛ233

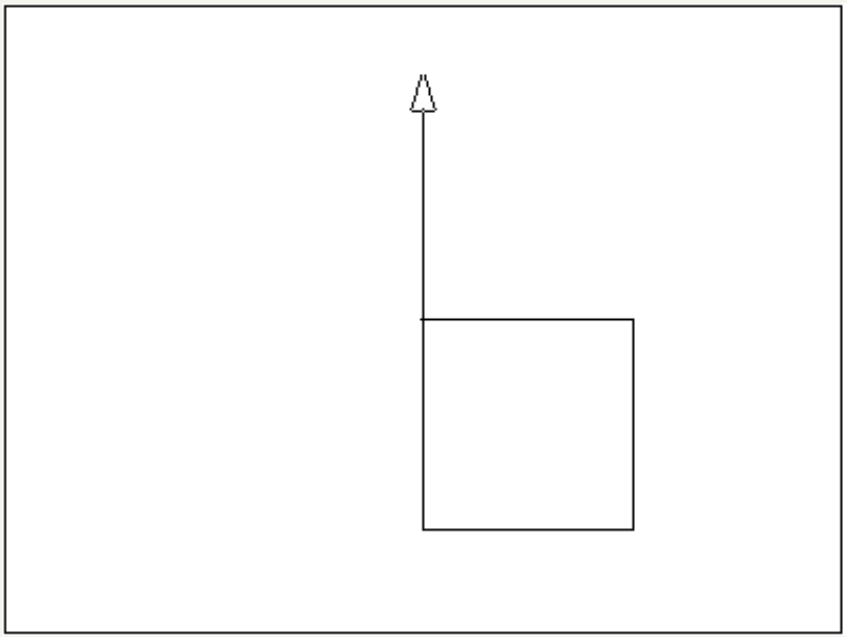
- Η τιμή της φωτεινότητας μπορεί να χρησιμοποιηθεί και για την μετατροπή μιας έγχρωμης φωτογραφίας σε ασπρόμαυρη
  - με μετατροπή του χρώματος κάθε εικονοστοιχείου σε ένα χρώμα της κλίμακας του γκρι ίσου με τη φωτεινότητα του έγχρωμου εικονοστοιχείου

```
public class Luminance {

// return the monochrome luminance of given color
public static double lum(Color color) {
    int r = color.getRed();
    int g = color.getGreen();
    int b = color.getBlue();
    return .299*r + .587*g + .114*b;
}

// return a gray version of this Color
public static Color toGray(Color color) {
    int y = (int) (Math.round(lum(color)));
    Color gray = new Color(y, y, y);
    return gray;
}

// are the two colors compatible?
public static boolean compatible(Color a, Color b) {
    return Math.abs(lum(a) - lum(b)) >= 128.0;
}

}
```

Turtle graphics

ΕΠΛ233

- Turtle graphics was originally developed by Seymour Papert at MIT in the 1960s as part of an educational programming language, Logo.
- Turtle graphics also has numerous commercial applications. For example, it is the basis for PostScript, a programming language for creating printed pages that is used for most newspapers, magazines, and books.



```
Welcome to Berkeley Logo version 5.5
? right 90
? forward 100
? right 90
? forward 100
? right 90
? forward 100
? right 90
? forward 100
? forward 100
?
```

Ψηφιακή επεξεργασία εικόνων

ΕΠΛ233

- Ψηφιακή εικόνα:
  - ένα ορθογώνιο πλέγμα εικονοστοιχείων (pixels)
  - κάθε εικονοστοιχείου έχει το δικό του χρώμα
  - αποθηκεύεται σαν ένας δυδιάστατος πίνακας τιμών
  - αναφέρεται συνήθως σαν raster ή bitmapped image (σε αντίθεση με τις εικόνες που δημιουργούνται μέσω κώδικα όπως της StdDraw, οι οποίες αποκαλούνται vector images)
- Η σύμβαση λέει ότι το πάνω αριστερά εικονοστοιχείο μιας εικόνας είναι το (0,0)

API της κλάσης Picture:

public class Picture (our data type for image processing)	
`Picture(String s)`	*create a picture from a file*
`Picture(int w, int h)`	*create a blank w-by-h picture*
`int width()`	*return the width of the picture*
`int height()`	*return the height of the picture*
`Color get(int i, int j)`	*return the color of pixel (i, j)*
`void set(int i, int j, Color c)`	*set the color of pixel (i, j) to c*
`void show()`	*display the image in a window*
`void save(String s)`	*save the image to a file*

```
import java.awt.Color;

public class Grayscale {

   public static void main(String[] args) {
      Picture pic = new Picture(args[0]);
      int width  = pic.width();
      int height = pic.height();

      // convert to grayscale
      for (int i = 0; i < width; i++) {
         for (int j = 0; j < height; j++) {
            Color color = pic.get(i, j);
            Color gray = Luminance.toGray(color);
            pic.set(i, j, gray);
         }
      }
      pic.show();
   }

}
```