Αρχικοποιήσεις Μεταβλητών και Αντικειμένων

Initializers

ΕΠΛ233

- Στην Java, πριν την χρήση μιας μεταβλητής αρχέγονου τύπου, η μεταβλητή αυτή θα πρέπει να έχει αρχικοποιηθεί .
- Διαφορετικά ο μεταφραστής διαμαρτύρεται και βγάζει μήνυμα λάθους. Π .χ.:

```
int i;
i = i++ ; // illegal
```

- Για την αποφυγή του προβλήματος αυτού , χρησιμοποιούμε μια ειδική ανάθεση *αρχικοποιητή* (initializer), με την οποία αρχικοποιούμε μια αρχέγονη μεταβλητή την στιγμή της δήλωσής της:

```
int i = 1;
```

- Εξαίρεση αποτελούν οι αρχέγονοι τύποι που είναι στοιχεία κάποιου αντικειμένου. Τα στοιχεία αυτά μπορούν να μην αρχικοποιηθούν από τον προγραμματιστή *(τι τιμές παίρνουν τότε* ;)

Παράδειγμα

ΕΠΛ233

```
class Measurement {
  boolean t; char c; byte b; short s;
  int i; long l; float f; double d;
  void print() {
   System.out.println("Data type Initial value\n" +
      "boolean " + t + "\n" +
      "char " + c + "\n" + "byte " + b + "\n" +
      "short " + s + "\n" +
      "int " + i + "\n" + "long " + l + "\n" +
      "float " + f + "\n" +"double " + d);
  }}

public class InitialValues {
   public static void main(String[] args){
      Measurement().print;
} }
```

Εκροή Παραδείγματος

ΕΠΛ233

```
C:\users\epl233\eckel-code\c04>java InitialValues
Data type Initial value
boolean false
char
byte 0
short 0
int 0
long 0
float 0.0
double 0.0
```

- Το **char** αρχικοποιείται σε **null.**
- Επίσης σε **null** αρχικοποιούνται και όποια χειριστήρια αντικειμένων περιέχονται ως στοιχεία σε αντικείμενα .

Αρχικοποιήσεις Αντικειμένων

ΕΠΛ233

```
class Measurement {
      boolean t = false;
      char c = 'z';
      byte b = 9;
      short s = 0;
      int i = f(s);
      long l = 12;
      float f = 2.3;
      double d = 0.9;
      void print() { }
}}
```

- Αρχικοποίηση μπορεί να γίνει με κλήση μεθόδου.
- Οι παράμετροι που δικαιούμαστε να περάσουμε στην μέθοδο, πρέπει να έχουν ήδη αρχικοποιηθεί (αλλοιώς λαμβάνουμε **exception**).

Κατασκευαστές (constructors)

Κατασκευαστές (constructors)

- Οι **constructors** είναι μέθοδοι που χρησιμοποιούνται για την αρχικοποίηση ενός αντικειμένου κατά την στιγμή της δημιουργίας του.

Κατασκευαστές (constructors)

- Οι **constructors** είναι μέθοδοι που χρησιμοποιούνται για την αρχικοποίηση ενός αντικειμένου κατά την στιγμή της δημιουργίας του.
- Δηλώνονται με το **ίδιο όνομα** με την κλάση στην οποία ανήκουν. Έτσι η ονομασία τους δεν χρειάζεται ιδιαίτερη διαχείριση, για αποφυγή τυχόν συγκρούσεων με άλλα ονόματα.

Κατασκευαστές (constructors)

ΕΠΛ233

- Οι **constructors** είναι μέθοδοι που χρησιμοποιούνται για την αρχικοποίηση ενός αντικειμένου κατά την στιγμή της δημιουργίας του.
- Δηλώνονται με το **ίδιο όνομα** με την κλάση στην οποία ανήκουν. Έτσι η ονομασία τους δεν χρειάζεται ιδιαίτερη διαχείριση, για αποφυγή τυχόν συγκρούσεων με άλλα ονόματα.
- Οι κατασκευαστές δεν επιστρέφουν τίποτε, χωρίς ωστόσο να δηλώνονται με τύπο επιστροφής **void.**

Κατασκευαστές (constructors)

ΕΠΛ233

- Οι **constructors** είναι μέθοδοι που χρησιμοποιούνται για την αρχικοποίηση ενός αντικειμένου κατά την στιγμή της δημιουργίας του.
- Δηλώνονται με το **ίδιο όνομα** με την κλάση στην οποία ανήκουν. Έτσι η ονομασία τους δεν χρειάζεται ιδιαίτερη διαχείριση, για αποφυγή τυχόν συγκρούσεων με άλλα ονόματα.
- Οι κατασκευαστές δεν επιστρέφουν τίποτε, χωρίς ωστόσο να δηλώνονται με τύπο επιστροφής **void.**
- Διευκολύνουν τον προγραμματισμό καθώς «ενοποιούν» ονοματολογικά την δήλωση και την αρχικοποίηση των κλάσεων και αντικειμένων.

Παράδειγμα constructor

ΕΠΛ233

```
class Rock {
  Rock() { // This is the constructor
     System.out.println("Creating Rock");
  }
}

public class SimpleConstructor {
  public static void main(String[] args) {
     for(int i = 0; i < 10; i++)
          new Rock();
  }
}
```

Προκαθορισμένοι κατασκευαστές

ΕΠΛ233

- Σε πολλές περιπτώσεις μια κλάση ορίζεται χωρίς κατασκευαστή. Στην περίπτωση αυτή (και μόνο), ο μεταφραστής ορίζει και καλεί έναν προκαθορισμένο κατασκευαστή (default constructor), ο οποίος δεν δέχεται παραμέτρους:

```
class Bird {
  int i;
}
public class DefaultConstructor {
  public static void main(String[] args) {
      Bird nc = new Bird(); // default!
      Bird nc = new Bird(1); // error!
  }
}
```

- Η αρχικοποίηση των στοιχείων του δημιουργούμενου αντικειμένου, γίνεται από τον προκαθορισμένο κατασκευαστή με μηδενικά και null χαρακτήρες.

Κατασκευαστές με παραμέτρους

ΕΠΛ233

- Ένας constructor μπορεί να δεχεται παραμέτρους, οι οποίες καθορίζουν περαιτέρω το πώς θα αρχικοποιηθεί το αντίστοιχο αντικείμενο.

```
class Rock2 {
    Rock2(int i) {
        System.out.println("Creating Rock number " + i);
    }
}
public class SimpleConstructor2 {
    public static void main(String[] args) {
        for(int i = 0; i < 10; i++)
            new Rock2(i);
    }
}
```

Υπερφόρτωση Μεθόδων (overloading)

Υπερφόρτωση Μεθόδων (overloading)

- **Υπερφόρτωση** μεθόδων προκύπτει στην Java όταν *το ίδιο όνομα μεθόδου μπορεί να δεχτεί διαφορετικούς τύπους και/ή διαφορετικό αριθμό παραμέτρων.*

Υπερφόρτωση Μεθόδων (overloading)

- **Υπερφόρτωση** μεθόδων προκύπτει στην Java όταν *το ίδιο όνομα μεθόδου μπορεί να δεχτεί διαφορετικούς τύπους και/ή διαφορετικό αριθμό παραμέτρων.*
- Η υπερφόρτωση μεθόδων χρησιμοποιείται ευρύτατα στον Α/ΣΠ, αλλά και ιδιαίτερα στους κατασκευαστές-constructors.

Υπερφόρτωση Μεθόδων (overloading)

- **Υπερφόρτωση** μεθόδων προκύπτει στην Java όταν *το ίδιο όνομα μεθόδου μπορεί να δεχτεί διαφορετικούς τύπους και/ή διαφορετικό αριθμό παραμέτρων.*
- Η υπερφόρτωση μεθόδων χρησιμοποιείται ευρύτατα στον Α/ΣΠ, αλλά και ιδιαίτερα στους κατασκευαστές-constructors.
- Αν δύο μέθοδοι (κατασκευαστές) έχουν το ίδιο όνομα, τότε πως γνωρίζει η Java ποιά μέθοδο θέλουμε να επικαλεστούμε;

Υπερφόρτωση Μεθόδων (overloading)

- **Υπερφόρτωση** μεθόδων προκύπτει στην Java όταν *το ίδιο όνομα μεθόδου μπορεί να δεχτεί διαφορετικούς τύπους και/ή διαφορετικό αριθμό παραμέτρων.*
- Η υπερφόρτωση μεθόδων χρησιμοποιείται ευρύτατα στον Α/ΣΠ, αλλά και ιδιαίτερα στους κατασκευαστές-constructors.
- Αν δύο μέθοδοι (κατασκευαστές) έχουν το ίδιο όνομα, τότε πως γνωρίζει η Java ποιά μέθοδο θέλουμε να επικαλεστούμε;
  - Κάθε υπερφορτωμένη μέθοδος πρέπει να δέχεται μια διαφορετική λίστα τύπων κατηγορημάτων.

Υπερφόρτωση Μεθόδων (overloading)

- **Υπερφόρτωση** μεθόδων προκύπτει στην Java όταν *το ίδιο όνομα μεθόδου μπορεί να δεχτεί διαφορετικούς τύπους και/ή διαφορετικό αριθμό παραμέτρων.*
- Η υπερφόρτωση μεθόδων χρησιμοποιείται ευρύτατα στον Α/ΣΠ, αλλά και ιδιαίτερα στους κατασκευαστές-constructors.
- Αν δύο μέθοδοι (κατασκευαστές) έχουν το ίδιο όνομα, τότε πως γνωρίζει η Java ποιά μέθοδο θέλουμε να επικαλεστούμε;
  - Κάθε υπερφορτωμένη μέθοδος πρέπει να δέχεται μια διαφορετική λίστα τύπων κατηγορημάτων.
  - Ακόμη και διαφορές στην σειρά των κατηγορημάτων είναι αρκετές για να ξεχωρίσουν δυό μεθόδους , αλλά αυτό δεν είναι «καλή» προγραμματιστική τακτική .

Παράδειγμα Υπερφόρτωσης

ΕΠΛ233

```
import java.util.*;
class Tree {
  int height;
  Tree() {
     prt("Planting a seedling"); height = 0;
  }
  Tree(int i) {
     prt("Creating new Tree that is "+ i +
         " feet tall"); height = i;
     }
  void info() {
     prt("Tree is "+ height + " feet tall");
  }
  void info(String s) {
     prt(s + ": Tree is " + height + " feet tall");
  }
  static void prt(String s) {
      System.out.println(s); }}
```

Παράδειγμα Υπερφόρτωσης (συνέχεια)


```
public class Overloading {
  public static void main(String[] args) {
     for(int i = 0; i < 5; i++) {
          Tree t = new Tree(i);
          t.info();
          t.info("overloaded method");
     }
  // Overloaded constructor:
  new Tree();
  }
}
```

Υπερφόρτωση και Αρχέγονοι τύποι

Υπερφόρτωση και Αρχέγονοι τύποι

- Αν ορίσουμε μια υπορουτίνα : **void f (double x) { /* */}** και την καλέσουμε σαν **f(5**), η παράμετρος «5» θα προαχθεί αυτόματα (promotion) σε 5.0.

Υπερφόρτωση και Αρχέγονοι τύποι

- Αν ορίσουμε μια υπορουτίνα : **void f (double x) { /* */}** και την καλέσουμε σαν **f(5**), η παράμετρος «5» θα προαχθεί αυτόματα (promotion) σε 5.0.
  - Τι θα συμβεί αν η f() είναι υπερφορτωμένη ώστε να δέχεται και ακέραιη παράμετρο;

Υπερφόρτωση και Αρχέγονοι τύποι

- Αν ορίσουμε μια υπορουτίνα : **void f (double x) { /* */}** και την καλέσουμε σαν **f(5**), η παράμετρος «5» θα προαχθεί αυτόματα (promotion) σε 5.0.
  - Τι θα συμβεί αν η f() είναι υπερφορτωμένη ώστε να δέχεται και ακέραιη παράμετρο;
  - *Θα εκτελεσθεί η έκδοση της f() που δέχεται ακέραιες παραμέτρους.*

Υπερφόρτωση και Αρχέγονοι τύποι

- Αν ορίσουμε μια υπορουτίνα : **void f (double x) { /* */}** και την καλέσουμε σαν **f(5**), η παράμετρος «5» θα προαχθεί αυτόματα (promotion) σε 5.0.
  - Τι θα συμβεί αν η f() είναι υπερφορτωμένη ώστε να δέχεται και ακέραιη παράμετρο;
  - *Θα εκτελεσθεί η έκδοση της f() που δέχεται ακέραιες παραμέτρους.*
- Γενικότερα, αν η τιμή που περνάμε σε μια μέθοδο έχει τύπο *μικρότερο* από τον τύπο της δηλωμένης παραμέτρου, τότε η τιμή *προάγεται* στον τύπο της παραμέτρου.

Υπερφόρτωση και Αρχέγονοι τύποι

ΕΠΛ233

- Αν ορίσουμε μια υπορουτίνα : **void f (double x) { /* */}** και την καλέσουμε σαν **f(5**), η παράμετρος «5» θα προαχθεί αυτόματα (promotion) σε 5.0.
  - Τι θα συμβεί αν η f() είναι υπερφορτωμένη ώστε να δέχεται και ακέραιη παράμετρο;
  - *Θα εκτελεσθεί η έκδοση της f() που δέχεται ακέραιες παραμέτρους.*
- Γενικότερα, αν η τιμή που περνάμε σε μια μέθοδο έχει τύπο *μικρότερο* από τον τύπο της δηλωμένης παραμέτρου, τότε η τιμή *προάγεται* στον τύπο της παραμέτρου.
- Αν υπάρχουν πολλοί τέτοιοι δυνατοί τύποι (πολλαπλών «υπερφορτώσεων»), τότε η προαγωγή γίνεται στον αμέσως μεγαλύτερο τύπο.

Υπερφόρτωση και Αρχέγονοι τύποι

ΕΠΛ233

- Αν ορίσουμε μια υπορουτίνα : **void f (double x) { /* */}** και την καλέσουμε σαν **f(5)**, η παράμετρος «5» θα προαχθεί αυτόματα (promotion) σε 5.0.
  - Τι θα συμβεί αν η f() είναι υπερφορτωμένη ώστε να δέχεται και ακέραιη παράμετρο;
  - *Θα εκτελεσθεί η έκδοση της f() που δέχεται ακέραιες παραμέτρους.*
- Γενικότερα, αν η τιμή που περνάμε σε μια μέθοδο έχει τύπο *μικρότερο* από τον τύπο της δηλωμένης παραμέτρου, τότε η τιμή *προάγεται* στον τύπο της παραμέτρου.
- Αν υπάρχουν πολλοί τέτοιοι δυνατοί τύποι (πολλαπλών «υπερφορτώσεων»), τότε η προαγωγή γίνεται στον αμέσως μεγαλύτερο τύπο.
- Εξαίρεση έχουμε στην περίπτωση που ο τύπος της δηλωμένης παραμέτρου είναι char κι εμείς περνάμε έναν ακέραιο, ο οποίος δεν αντιστοιχεί σε τιμή χαρακτήρα char, τότε η τιμή αυτή προάγεται σε ακέραιο (και όχι , π.χ. σε short).

Παράδειγμα

ΕΠΛ233

```
public class Foo {
  void f1(double x) {
     System.out.println("double f1 -->" + x); }

  void f1(int x) {
     System.out.println("int f1 -->" + x); }

  public static void main(String[] args) {

     Foo ff = new Foo();
     ff.f1(5);
  }
}
```

Η χρήση του this

ΕΠΛ233

```
class Banana {
   double param;

   Banana(int prm) {
         param = prm;
   }

   void f(int i) {
         System.out.println("Calc: " + i * param);
    }
Banana a = new Banana(5), b = new Banana(7);
a.f(1);
b.f(2);
```

Η χρήση του this (συνέχεια)

ΕΠΛ233

- Πως μπορεί η μέθοδος **f()** να γνωρίζει αν καλείται από το αντικείμενο **a** ή το αντικείμενο **b**;
  - a.f(1) ⇔ Banana.f(a,1)
  - b.f(2) ⇔ Banana.f(b,2)
  - Αν θέλουμε, μέσα από κάποιο αντικείμενο, να αποκτήσουμε πρόσβαση-χειριστήριο προς τον εαυτό του, μπορούμε να χρησιμοποιήσουμε την ειδική μεταβλητή **this**, η οποία είναι χειριστήριο για το αντικείμενο μας.
- Ανάθεση στην **this** *δεν επιτρέπεται*.
- Μέσω της **this** μπορούμε να περάσουμε το τρέχον αντικείμενο σαν παράμετρο σε μεθόδους άλλων αντικειμένων.

Παράδειγμα χρήσης this

ΕΠΛ233

```
// Simple use of the "this" keyword.
public class Leaf {
    private int i = 0;
    Leaf increment() {
            i++;
            return this;
    }
    void print() { System.out.println("i = " + i); }
    public static void main(String[] args) {
            Leaf x = new Leaf();
            x.increment().increment().increment().print();
    }
}
```

- Πολλαπλή κλήση της ίδιας μεθόδου πάνω στο ίδιο αντικείμενο.

Κλήση κατασκευαστών από κατασκευαστές

- Μέσα σε έναν **constructor,** η δεσμευμένη λέξη **this** μπορεί να χρησιμοποιηθεί και σαν κλήση μεθόδου, με πέρασμα σε αυτήν παραμέτρων.
- Σε μια τέτοια περίπτωση, έχουμε ρητή κλήση μιας διαφορετικής εκδοχής του constructor, η οποία έχει υποδηλωθεί με υπερφόρτωση.
- Η λέξη **this** μπορεί να χρησιμοποιηθεί σαν **constructor** μέσα σε κάποιον constructor, μόνο σαν *πρώτη εντολή* του *constructor* και μόνο για μιά φορά.

Παράδειγμα

ΕΠΛ233

```
// Calling constructors with "this"

public class Flower {
  private int petalCount = 0;
  private String s = new String("null");
  Flower(int petals) {
          petalCount = petals;
  }
  Flower(String ss) {
          s = ss;
  }
```

Παράδειγμα (συνέχεια)

ΕΠΛ233

```
Flower(String s, int petals) {

  this(petals);
  this(s);  // Δεν επιτρέπεται δεύτερη κλήση του this
  this.s = s;
}

Flower() {
  this("hi", 47);
}
```

Παράδειγμα (συνέχεια)


```
void print() {
    this(11);  // Δεν επιτρέπεται η κλήση του this εκτός constructor
    System.out.println("petalCount = " + petalCount + " s = "+
    s);
}

 public static void main(String[] args) {
    Flower x = new Flower();
    x.print();
 }
}
```

Στατικές Μέθοδοι και this

Στατικές Μέθοδοι και this

- Σε μια στατική μέθοδο **δεν** μπορεί να γίνει χρήση του **this** (γιατί;)

Στατικές Μέθοδοι και this

- Σε μια στατική μέθοδο **δεν** μπορεί να γίνει χρήση του **this** (γιατί;)
- Από μια μη–στατική μέθοδο μπορούν να κληθούν στατικές μέθοδοι.

Στατικές Μέθοδοι και this

ΕΠΛ233

- Σε μια στατική μέθοδο **δεν** μπορεί να γίνει χρήση του **this** (γιατί;)
- Από μια μη–στατική μέθοδο μπορούν να κληθούν στατικές μέθοδοι.
- Στατικές μέθοδοι μπορούν να καλέσουν μη-στατικές μεθόδους;
  - Όχι, διότι σε ποιό αντικείμενο θα σταλεί το μήνυμα για κλήση της μη-στατικής μεθόδου;
  - Εξαίρεση: η κλήση μη-στατικής μεθόδου μέσω χειριστηρίου που περνιέται σαν παράμετρος στην στατική μέθοδο.

```
public class Foo {
     static void test(Foo g) {
            g.f1(0.9);
     }
     void f1(double x) {
        System.out.println("double f1 -->" + x);
     }

     void f1() { test(this); }

     public static void main(String[] args) {
          Foo ff = new Foo();
          ff.f1();
     }
     }
```

Ανασκόπηση Αρχικοποιήσεων

Σειρά Αρχικοποιήσεων

ΕΠΛ233

- *Μέσα σε μια κλάση*, η σειρά των αρχικοποιήσεων καθορίζεται από τη σειρά δήλωσης των πεδίων δεδομένων της κλάσης.
- Ακόμη κι αν οι αρχικοποιήσεις είναι διεσπαρμένες ανάμεσα σε δηλώσεις μεθόδων, τα πεδία θα αρχικοποιηθούν πριν την κλήση οποιασδήποτε μεθόδου, **ακόμη και του constructor.**
- Πότε γίνεται η αρχικοποίηση **στατικών μεταβλητών;**
  - Μόνο όταν αυτό καταστεί αναγκαίο, είτε λόγω δημιουργίας του πρώτου σχετικού αντικειμένου, είτε λόγω κλήσης κάποιας στατικής μεθόδου της αντίστοιχης κλάσης.

Ανασκόπηση Δημιουργίας Αντικειμένων

- Έστω ότι έχουμε ορίσει μια κλάση Dog.
- Την πρώτη φορά που δημιουργείται ένα αντικείμενο Dog, ή την πρώτη φορά που καλείται μια **στατική** μέθοδος της Dog ή γίνεται πρόσβαση σε ένα **στατικό** πεδίο της κλάσης Dog, ο διερμηνέας της Java πρέπει να βρεί την κλάση **Dog.class**, την οποία αναζητεί με την βοήθεια του **classpath.**
- Καθώς η **Dog.class** φορτώνεται (οπότε και δημιουργείται ένα αντικείμενο **Class**), εκτελούνται όλοι οι **στατικοί αρχικοποιητές** (static initializers). Επομένως η στατική αρχικοποίηση συμβαίνει μόνο μια φορά, όταν το **Class Object** φορτώνεται για πρώτη φορά.
- Όταν δημιουργηθεί αντικείμενο Dog με την **new Dog(),** η διεργασία δημιουργίας κρατάει πρώτα αρκετό χώρο στον σωρό.
- Ο χώρος που κρατήθηκε αρχικοποιείται με μηδενικά.

Ανασκόπηση Δημιουργίας Αντικειμένων

- Οποιεσδήποτε αρχικοποιήσεις έχουν δηλωθεί εκτελούνται.
- Εκτελούνται οι **constructors** τού αντικειμένου. Στο σημείο αυτό μπορούμε να έχουμε αρκετή δραστηριότητα, ιδιαίτερα όταν η κλάση μας κληρονομεί χαρακτηριστικά άλλων κλάσεων.

Ρητή στατική αρχικοποίηση

ΕΠΛ233

- Μπορούμε να συγκεντρώσουμε τις στατικές αρχικοποιήσεις (static initializations) μέσα σε ένα ειδικό "στατικό πλαίσιο" (static block), στο εσωτερικό μιας κλάσης.
- Οι ρητές στατικές αρχικοποιήσεις εκτελούνται μαζί με τους στατικούς αρχικοποιητές.

```
public class Spoon {
        static int i;
        static {
                i = 47;
        }
```

```
class Cups {
        static Cup cup1;
        static Cup cup2;
        static {
                cup1 = new Cup(1);
                cup2 = new Cup(2);
        }
}
```

Αρχικοποίηση στιγμιοτύπου

ΕΠΛ233

- Non-static instance initialization: παρόμοια σύνταξη με τους ρητούς στατικούς αρχικοποιητές:
- Π.χ.:

```
public class Mugs {
    Mug mug1;
    Mug mug2;
    {
            mug1 = new Mug(1);
            mug2 = new Mug(2);
            System.out.println("Instance Initialization");
    }
...
}
```

Άλλα θέματα

Variable Argument Lists

ΕΠΛ233

- Στη JAVA 5.0, οι μέθοδοι μπορούν να δηλωθούν ώστε κατά την κλήση τους να μπορούν να λάβουν μεταβλητό αριθμό παραμέτρων (vararg methods)
  - Π.χ. η System.out.printf()
  - System.out.printf("%d %d 5d\n",1,2,3);
- Οι μέθοδοι vararg τυγχάνουν διαχείρισης από τον μεταγλωττιστή, ο οποίος μετατρέπει μια vararg δήλωση σε κανονική δήλωση μεθόδου που αναμένει σαν παράμετρο ένα πίνακα παραμέτρων. Π.χ.:
  - public static int max(int first, **int...** rest) { ... }
  - public static int max(int first, int[] rest) { ... }

Enumerated types

ΕΠΛ233

- Στη JAVA 5.0, εκτός από τους τύπους αναφοράς (reference types) Class και Interface, έχει προστεθεί και ο τύπος Απαρίθμησης (Enumerated type ή enum).
- Ένας τύπος Απαρίθμησης προσδιορίζει ένα πεπερασμένο (απαριθμήσιμο) σύνολο τιμών και εξασφαλίζει ασφάλεια τύπου (type safety):
  - μια μεταβλητή τύπου απαρίθμησης μπορεί να αποθηκεύσει μόνο τιμές αυτού του τύπου ή το null
- public enum Seasons {WINTER, SPRING, SUMMER, AUTUMN}
- Οι τιμές του τύπου Απαρίθμησης (enumerated values or enum constants) είναι σαν στατικά και τελικά πεδία δεδομένων (static, final data fields) και χρησιμοποιούνται ως εξής: Seasons.WINTER

Boxing, Unboxing, Autoboxing


```
Integer i = 0; // boxing
Number n = 0.0f; // boxing
Integer i = 1;
int j = i;
i++;
Integer k = i+2; // i is unboxed and boxed up again
i=null;
j = i; // unboxing throws NullPointerException
```

Αποκομιδή Σκυβάλων

Garbage Collection

Αποκομιδή Σκυβάλων

ΕΠΛ233

- Πως δημιουργούνται τα «σκουπίδια» στην JAVA και που είναι αποθηκευμένα;
- Ποιός είναι ο ρόλος του αποκομιστή σκυβάλων (σκουπιδιάρη);
  - Να απελευθερώνει μνήμη, η οποία έχει δεσμευθεί με τη **new** και να την επιστρέφει στο σωρό.
  - Είναι αυτό αρκετό για την «εκκαθάριση» αχρείαστων αντικειμένων;
    - Αρκετές φορές, η «εκκαθάριση» των αντικειμένων στη Java δεν είναι πολύ απλή υπόθεση, που μπορεί να αφεθεί στον σκουπιδιάρη.
    - Υπάρχει η πιθανότητα, με τη δημιουργία του αντικειμένου αυτού να έχουν κληθεί βιβλιοθήκες οι οποίες δημιουργούν άλλα αντικείμενα, γραφικά, δεσμεύουν μνήμη χωρίς κλήση της new (με επίκληση ιθαγενών-native μεθόδων) κοκ.

Γενικές Αρχές για την Αποκομιδή Σκυβάλων

- Τα αντικείμενά σας μπορεί να μην συλλεχθούν ποτέ από τον αποκομιστή, ακόμη κι αν καταστούν σκουπίδια.
- Η αποκομιδή σκυβάλων *δεν* ισοδυναμεί με *καταστροφή* των αντικειμένων-σκουπιδιών.
- Η αποκομιδή σκυβάλων αφορά μόνο στην απελευθέρωση μνήμης.

Αποτελείωμα και Αποκομιδή Σκυβάλων

- Ο αποκομιστής σκυβάλων δεν επιλύει το πρόβλημα της **εκκαθάρισης** αντικειμένων στη Java, διότι:
  - **«Απορρίματα» της Java μπορεί να μη συλλεχθούν από τον αποκομιστή σκυβάλων.** Ο λόγος είναι ότι συχνά τα προγράμματα δεν ξεμένουν από μνήμη, οπότε δεν καλείται ο GC στο χρόνο ζωής τους.
  - Ο αποκομιστής σκυβάλων γνωρίζει πως να αποδεσμεύσει μνήμη που έχει κρατηθεί με την **new,** όχι όμως και τι θα κάνει με ιδιάζουσες περιπτώσεις μνήμης που έχουν κρατηθεί από κάποιο αντικείμενο.
  - Στη Java, η αποκομιδή σκυβάλων **δεν ισοδυναμεί με καταστροφή των αντικειμένων** (όπως στην C++). Αν υπάρχει κάποια δραστηριότητα που πρέπει να εκτελεσθεί πριν την ολοκλήρωση της χρήσης ενός αντικειμένου, τη δραστηριότητα αυτή πρέπει να την καθορίσει σαφώς ο προγραμματιστής.

Finalize

ΕΠΛ233

- Για κάθε κλάση της Java μπορούμε να ορίσουμε μια μέθοδο **finalize(),** με την οποία μπορούμε να κάνουμε αναγκαίες «εκκαθαρίσεις» που προηγούνται της αποδέσμευσης των αντικειμένων της κλάσης.
- Στην περίπτωση που έχει οριστεί η **finalize**, όταν κληθεί ο αποκομιστής σκυβάλων (GC) και επιχειρήσει να απελευθερώσει τη μνήμη αντικειμένου της αντίστοιχης κλάσης ο GC:
  - θα καλέσει πρώτα την finalize
  - στο επόμενο πέρασμά του θα απελευθερώσει τη μνήμη του αντικειμένου
- Ο GC στοχεύει **στην απελευθέρωση μνήμης**. Αυτή θα πρέπει να είναι και η δραστηριότητα της finalize, όποτε χρησιμοποιείται.
- Η χρησιμότητα της finalize περιορίζεται κυρίως σε ειδικές περιπτώσεις («ιθαγενείς» μέθοδοι – native methods).

Ρητή Εκκαθάριση Αντικειμένων

ΕΠΛ233

- Στη Java τα αντικείμενα δημιουργούνται μόνο με χρήση της new.
- Δεν δημιουργούνται «τοπικά» αντικείμενα (στη στοίβα) και δεν υπάρχει μέθοδος delete για καταστροφή αντικειμένων.
- Αν ωστόσο θέλουμε να «εξαναγκάσουμε» την απαλοιφή μη χρησιμοποιούμενων αντικειμένων, μπορούμε να καλέσουμε από το πρόγραμμά μας τον Αποκομιστή Σκυβάλων (GC), ακολουθούμενο από την μέθοδο runFinalization:

  System.gc();
  System.runFinalization();
- Πως λειτουργεί η **System.runFinalization();**

Παράδειγμα


```
// Demonstration of the garbage
// collector and finalization

class Chair {
    static boolean gcrun = false;
    static boolean f = false;
    static int created = 0;
    static int finalized = 0;
    int i;
    Chair() {
                i = ++created;
                if  (created == 47)
                        System.out.println("Created 47");
    }
```

Παράδειγμα (συνέχεια)

ΕΠΛ233

```
public void finalize() {
  if (!gcrun) { // The first time finalize() is called:
       gcrun = true;
       System.out.println("Beginning to finalize after "
         + created  +" Chairs have been created");
  }
  if (i == 47) {
     System.out.println("Finalizing Chair #47, " +
                  "Setting flag to stop Chair creation");
     f = true;
  }
  finalized++;
  if  (finalized  >=  created)
     System.out.println("All " + finalized +
                                 "finalized");
 }
}
```

Παράδειγμα (συνέχεια)

ΕΠΛ233

```
public class Garbage {
   public static void main(String[] args){
            while  (!Chair.f) {
                    new Chair();
                    new String("To take up space");
            }
   System.gc(); // forces execution of GC
   System.runFinalization(); // finalizes all unfinalized objects
   }
}
```

Τεχνικές Αποκομιδής Σκυβάλων

ΕΠΛ233

Τεχνικές Αποκομιδής Σκυβάλων

- Reference counting (μέτρηση αναφορών) – τι γίνεται με αυτοαναφερόμενες κυκλικές δομές σκυβάλων;

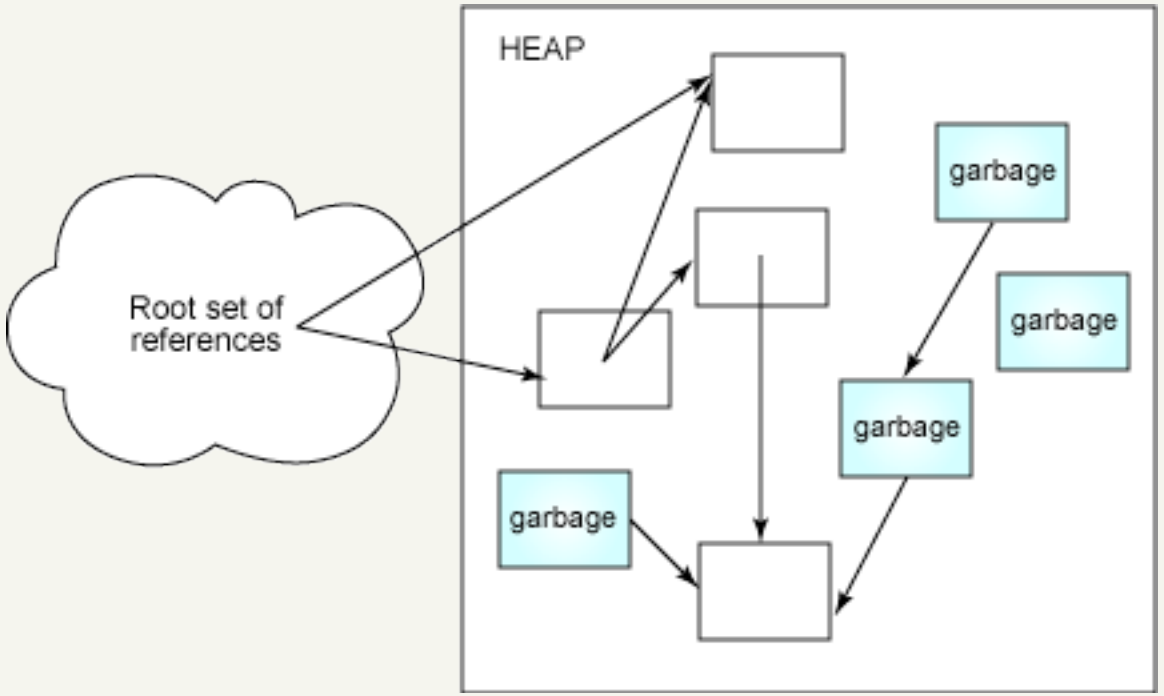
Τεχνικές Αποκομιδής Σκυβάλων

- Reference counting (μέτρηση αναφορών) – τι γίνεται με αυτοαναφερόμενες κυκλικές δομές σκυβάλων;
- Tracing live objects back to references living on the stack or static memory: ιχνηλασία σε ζώντα χειριστήρια.

Τεχνικές Αποκομιδής Σκυβάλων

ΕΠΛ233

- Reference counting (μέτρηση αναφορών) – τι γίνεται με αυτοαναφερόμενες κυκλικές δομές σκυβάλων;
- Tracing live objects back to references living on the stack or static memory: ιχνηλασία σε ζώντα χειριστήρια.



Τεχνικές Αποκομιδής Σκυβάλων

- Reference counting (μέτρηση αναφορών) – τι γίνεται με αυτοαναφερόμενες κυκλικές δομές σκυβάλων;
- Tracing live objects back to references living on the stack or static memory: ιχνηλασία σε ζώντα χειριστήρια.
- Προσέγγιση τού JVM: προσαρμοζόμενος ΑΣ (adaptive)

Τεχνικές Αποκομιδής Σκυβάλων

ΕΠΛ233

- Reference counting (μέτρηση αναφορών) – τι γίνεται με αυτοαναφερόμενες κυκλικές δομές σκυβάλων;
- Tracing live objects back to references living on the stack or static memory: ιχνηλασία σε ζώντα χειριστήρια.
- Προσέγγιση τού JVM: προσαρμοζόμενος ΑΣ (adaptive)
  - Stop-and-copy: μεταφορά των ζώντων αντικειμένων από έναν σωρό σε κάποιον άλλο, αφού πρώτα σταματήσει η εκτέλεση του προγράμματος.

Τεχνικές Αποκομιδής Σκυβάλων

ΕΠΛ233

- Reference counting (μέτρηση αναφορών) – τι γίνεται με αυτοαναφερόμενες κυκλικές δομές σκυβάλων;
- Tracing live objects back to references living on the stack or static memory: ιχνηλασία σε ζώντα χειριστήρια.
- Προσέγγιση τού JVM: προσαρμοζόμενος ΑΣ (adaptive)
  - Stop-and-copy: μεταφορά των ζώντων αντικειμένων από έναν σωρό σε κάποιον άλλο, αφού πρώτα σταματήσει η εκτέλεση του προγράμματος.
  - Mark-and-sweep: εντοπισμός και υποσημείωση των ζώντων αντικειμένων, ξεκινώντας από την στοίβα και την στατική μνήμη. Μετά το πέρας της υποσημείωσης, σάρωση του σωρού και εκκαθάριση των σκυβάλων.

Τεχνικές Αποκομιδής Σκυβάλων

ΕΠΛ233

- Reference counting (μέτρηση αναφορών) – τι γίνεται με αυτοαναφερόμενες κυκλικές δομές σκυβάλων;
- Tracing live objects back to references living on the stack or static memory: ιχνηλασία σε ζώντα χειριστήρια.
- Προσέγγιση τού JVM: προσαρμοζόμενος ΑΣ (adaptive)
  - Stop-and-copy: μεταφορά των ζώντων αντικειμένων από έναν σωρό σε κάποιον άλλο, αφού πρώτα σταματήσει η εκτέλεση του προγράμματος.
  - Mark-and-sweep: εντοπισμός και υποσημείωση των ζώντων αντικειμένων, ξεκινώντας από την στοίβα και την στατική μνήμη. Μετά το πέρας της υποσημείωσης, σάρωση του σωρού και εκκαθάριση των σκυβάλων.
    - δουλεύει ικανοποιητικά όταν δεν υπάρχουν πολλά σκουπίδια

Αποκομιδή Σκυβάλων στη JAVA

Αποκομιδή Σκυβάλων στη JAVA

- Στις αρχικές εκδόσεις της JAVA, η δέσμευση μνήμης για κατασκευή αντικειμένων ήταν εξίσου “ακριβή” με τη δυναμική δέσμευση μνήμης στη C και στη C++

Αποκομιδή Σκυβάλων στη JAVA

- Στις αρχικές εκδόσεις της JAVA, η δέσμευση μνήμης για κατασκευή αντικειμένων ήταν εξίσου “ακριβή” με τη δυναμική δέσμευση μνήμης στη C και στη C++
  - “first-fit” ή “best-fit” αλγόριθμοι για τη διαχείριση του ελεύθερου χώρου στον σωρό

Αποκομιδή Σκυβάλων στη JAVA

ΕΠΛ233

- Στις αρχικές εκδόσεις της JAVA, η δέσμευση μνήμης για κατασκευή αντικειμένων ήταν εξίσου “ακριβή” με τη δυναμική δέσμευση μνήμης στη C και στη C++
  - “first-fit” ή “best-fit” αλγόριθμοι για τη διαχείριση του ελεύθερου χώρου στον σωρό
- Η αποδέσμευση ήταν εξίσου ακριβή:

Αποκομιδή Σκυβάλων στη JAVA

- Στις αρχικές εκδόσεις της JAVA, η δέσμευση μνήμης για κατασκευή αντικειμένων ήταν εξίσου “ακριβή” με τη δυναμική δέσμευση μνήμης στη C και στη C++
  - “first-fit” ή “best-fit” αλγόριθμοι για τη διαχείριση του ελεύθερου χώρου στον σωρό
- Η αποδέσμευση ήταν εξίσου ακριβή:
  - mark-sweep ολόκληρου του σωρού σε κάθε σκούπισμα από τον αποκομιστή σκυβάλων

Αποκομιδή Σκυβάλων στη JAVA

ΕΠΛ233

- Στις αρχικές εκδόσεις της JAVA, η δέσμευση μνήμης για κατασκευή αντικειμένων ήταν εξίσου “ακριβή” με τη δυναμική δέσμευση μνήμης στη C και στη C++
  - “first-fit” ή “best-fit” αλγόριθμοι για τη διαχείριση του ελεύθερου χώρου στον σωρό
- Η αποδέσμευση ήταν εξίσου ακριβή:
  - mark-sweep ολόκληρου του σωρού σε κάθε σκούπισμα από τον αποκομιστή σκυβάλων
- Σε μεταγενέστερες εκδόσεις της JAVA (HotSpot JVM), ο σωρός λειτουργεί σαν ιμάντας (conveyor belt).

Αποκομιδή Σκυβάλων στη JAVA

ΕΠΛ233

- Στις αρχικές εκδόσεις της JAVA, η δέσμευση μνήμης για κατασκευή αντικειμένων ήταν εξίσου “ακριβή” με τη δυναμική δέσμευση μνήμης στη C και στη C++
  - “first-fit” ή “best-fit” αλγόριθμοι για τη διαχείριση του ελεύθερου χώρου στον σωρό
- Η αποδέσμευση ήταν εξίσου ακριβή:
  - mark-sweep ολόκληρου του σωρού σε κάθε σκούπισμα από τον αποκομιστή σκυβάλων
- Σε μεταγενέστερες εκδόσεις της JAVA (HotSpot JVM), ο σωρός λειτουργεί σαν ιμάντας (conveyor belt).
  - Η δημιουργία (μικρών) αντικειμένων γίνεται πάρα πολύ γρήγορα στη JAVA (10 εντολές μηχανής) και πιο “φθηνά” από την C/C++.

Αποκομιδή Σκυβάλων στη JAVA

- Στις αρχικές εκδόσεις της JAVA, η δέσμευση μνήμης για κατασκευή αντικειμένων ήταν εξίσου “ακριβή” με τη δυναμική δέσμευση μνήμης στη C και στη C++
  - “first-fit” ή “best-fit” αλγόριθμοι για τη διαχείριση του ελεύθερου χώρου στον σωρό
- Η αποδέσμευση ήταν εξίσου ακριβή:
  - mark-sweep ολόκληρου του σωρού σε κάθε σκούπισμα από τον αποκομιστή σκυβάλων
- Σε μεταγενέστερες εκδόσεις της JAVA (HotSpot JVM), ο σωρός λειτουργεί σαν ιμάντας (conveyor belt).
  - Η δημιουργία (μικρών) αντικειμένων γίνεται πάρα πολύ γρήγορα στη JAVA (10 εντολές μηχανής) και πιο “φθηνά” από την C/C++.
  - Επίσης, η αποκομιδή των σκυβάλων είναι πολύ γρήγορη.