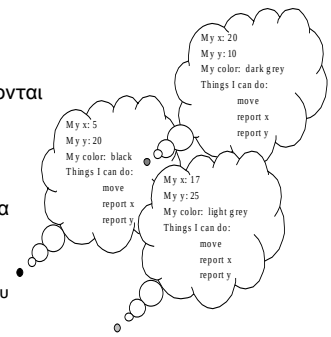
Εισαγωγή στη γλώσσα Java

Franz Marc, *Rehe im Walde (II), 1913-14*

Κωστής Σαγώνας <kostis@cs.ntua.gr>
Νίκος Παπασπύρου <nickie@softlab.ntua.gr>

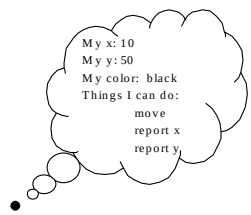
Παράδειγμα αντικειμενοστρεφούς τρόπου σκέψης

- Έγχρωμα σημεία στην οθόνη
- Τι δεδομένα αποθηκεύονται στο καθένα;
  - Οι συντεταγμένες του
  - Το χρώμα του
- Τι θέλουμε να μπορεί να κάνει το κάθε σημείο;
  - Να μετακινηθεί
  - Να αναφέρει τη θέση του



Η ορολογία της Java

- Κάθε σημείο είναι ένα **αντικείμενο (object)**
- Που περιλαμβάνει τρία **πεδία (fields)**
- Έχει τρεις **μεθόδους (methods)**
- Και κάθε αντικείμενο είναι ένα **στιγμιότυπο (instance)** της ίδιας **κλάσης (class)**



Αντικειμενοστρεφές στυλ προγραμματισμού

- Η επίλυση προβλημάτων γίνεται μέσω αντικειμένων:
  - μικρά δέματα από δεδομένα που ξέρουν πώς να κάνουν πράγματα στον εαυτό τους
- Δηλαδή η ιδέα δεν είναι ότι π.χ. *το πρόγραμμα ξέρει πώς να μετακινήσει ένα σημείο*, αλλά ότι *το σημείο ξέρει πώς να μετακινήσει τον εαυτό του*
- Οι γλώσσες αντικειμενοστρεφούς προγραμματισμού κάνουν πιο εύκολο το συγκεκριμένο τρόπο σκέψης και προγραμματισμού

Παράδειγμα ορισμού κλάσης στη Java

```
public class Point {
  private int x,y;
  private Color myColor;

  public int currentX() {
    return x;
  }

  public int currentY() {
    return y;
  }

  public void move(int newX, int newY) {
    x = newX;
    y = newY;
  }
}
```

field definitions

method definitions

Πρωτόγονοι τύποι της Java

- **`char`**: 0..2^{16}-1, γράφονται ως **`'a'`**, **`'\n'`**, …, με χρήση του συνόλου χαρακτήρων Unicode
- **`byte`**: -2^{7}..2^{7}-1
- **`short`**: -2^{15}..2^{15}-1
- **`int`**: -2^{31}..2^{31}-1, γράφονται με το συνηθισμένο τρόπο
- **`long`**: -2^{63}..2^{63}-1, γράφονται με χρήση ενός **`L`** στο τέλος
- **`float`**: IEEE 32-bit standard, γράφονται με χρήση ενός **`F`** στο τέλος
- **`double`**: IEEE 64-bit standard, γράφονται ως αριθμοί κινητής υποδιαστολής (π.χ., **`1.2`**, **`1.2e-5`**, ή **`1e3`**)
- **`boolean`**: **`true`** και **`false`**
- Εκκεντρικοί τύποι: **`void`** και **`null`**

Κατασκευαζόμενοι τύποι στη Java

- Όλοι οι κατασκευαζόμενοι τύποι είναι **τύποι αναφορών (reference types)**
- Με άλλα λόγια είναι αναφορές σε αντικείμενα
  - Ονόματα κλάσεων, όπως π.χ. **`Point`**
  - Ονόματα κάποιας διαπροσωπείας (interface)
  - Ονόματα τύπων πινάκων, όπως π.χ. **`Point[]`** ή **`int[]`**

Συμβολοσειρές (strings)

- Προκαθορισμένος τύπος αλλά όχι πρωτόγονος: η κλάση **`String`**
- Μια σειρά από χαρακτήρες που περικλείονται από διπλές αποστρόφους και συμπεριφέρονται σα μια σταθερή συμβολοσειρά
- Αλλά στην πραγματικότητα είναι ένα στιγμιότυπο της κλάσης **`String`**, δηλαδή ένα αντικείμενο που περιέχει τη συγκεκριμένη σειρά χαρακτήρων

`"Hello there"`

Κλήσεις μεθόδου στιγμιότυπου (instance method)

Έκφραση Java	Τιμή
`s.length()`	το μήκος του **`String s`**
`s.equals(r)`	**`true`** εάν **`s`** και **`r`** είναι ίδια και **`false`** εάν όχι
`r.equals(s)`	το ίδιο με το παραπάνω
`s.toUpperCase()`	ένα αντικείμενο **`String`** που είναι το **`String s`** αλλά με κεφαλαία
`s.charAt(3)`	η τιμή του χαρακτήρα στη θέση 3 στο **`String s`** (δηλαδή, ο τέταρτός του χαρακτήρας)
`s.toUpperCase().charAt(3)`	η τιμή του χαρακτήρα στη θέση 3 στο **`String s`** με όλα κεφαλαία

Κλήσεις μεθόδου κλάσης (class method calls)

- Οι **μέθοδοι μιας κλάσης (class methods)** ορίζουν λειτουργίες που η κλάση ξέρει πώς να κάνει – δεν ορίζουν αντικείμενα της κλάσης
- Οι κλάσεις χρησιμεύουν ως τόποι ονομάτων
- Οι κλήσεις μεθόδων είναι κάτι σαν τις συνήθεις κλήσεις συναρτήσεων στις μη αντικειμενοστρεφείς γλώσσες

Έκφραση Java	Τιμή
`String.valueOf(1==2)`	**`"false"`**
`String.valueOf(6*7)`	**`"42"`**
`String.valueOf(1.0/3.0)`	**`"0.3333333333333333"`**

Εκφράσεις δημιουργίας αντικειμένων

- Δημιουργία ενός νέου αντικειμένου το οποίο είναι στιγμιότυπο κάποιας συγκεκριμένης κλάσης

  <creation-expression> **`::=`** **`new`** *<class-name>* **`(`***<parameter-list>***`)`**
- Οι παράμετροι περνιούνται σε έναν **κατασκευαστή (constructor)**–κάτι σαν μια ειδική μέθοδο της κλάσης

Έκφραση Java	Τιμή
`new String()`	ένα νέο **`String`** με μήκος μηδέν
`new String(s)`	ένα νέο **`String`** που περιλαμβάνει ένα αντίγραφο του **`String s`**
`new String(chars)`	ένα νέο **`String`** που περιλαμβάνει τους χαρακτήρες από τον πίνακα **`chars`**

Δεν υπάρχει τρόπος να καταστρέψουμε αντικείμενα

- Τα αντικείμενα δημιουργούνται με κλήση της **`new`**
- Όμως δεν υπάρχει άμεσος τρόπος να τα καταστρέψουμε ή να αποδεσμεύσουμε τη μνήμη που καταλαμβάνουν
- Αυτό γίνεται αυτόματα μέσω συλλογής σκουπιδιών (garbage collection)

Ανακύκλωση στην Αθήνα: Πού είναι ο κάδος, ο-έ-ο;

Γενικές πληροφορίες για τελεστές στη Java

- Όλοι οι τελεστές είναι αριστερά προσεταιριστικοί, εκτός από την ανάθεση
- Υπάρχουν 15 επίπεδα προτεραιότητας
  - Κάποια επίπεδα είναι προφανή: π.χ. ο τελεστής `*` έχει μεγαλύτερη προτεραιότητα από τον τελεστή `+`
  - Άλλα επίπεδα είναι λιγότερο προφανή: π.χ. ο τελεστής `<` έχει μεγαλύτερη προτεραιότητα από τον `!=`
- Επιτρέπονται πολλοί εξαναγκασμοί μετατροπής τύπου
  - Από `null` σε κάθε τύπο αναφοράς
  - Κάθε τιμή μπορεί να μετατραπεί σε `String` σε κάποια συνένωση
  - Ένας τύπος αναφοράς σε έναν άλλον (κάποιες φορές)

Παράδειγμα ορισμού κλάσεων: μία απλά συνδεδεμένη λίστα

Παράδειγμα κλάσης: `ConsCell`

```
/**
 * A ConsCell is an element in a linked list of ints.
 */
public class ConsCell {
  private int head;      // the first item in the list
  private ConsCell tail; // rest of the list, or null

  /**
   * Construct a new ConsCell given its head and tail.
   * @param h the int contents of this cell
   * @param t the next ConsCell in the list, or null
   */
  public ConsCell(int h, ConsCell t) {
    head = h;
    tail = t;
  }
```



```
  /**
   * Accessor for the head of this ConsCell.
   * @return the int contents of this cell
   */
  public int getHead() {
    return head;
  }

  /**
   * Accessor for the tail of this ConsCell.
   * @return the next ConsCell in the list, or null
   */
  public ConsCell getTail() {
    return tail;
  }
}
```


Χρήση της κλάσης `ConsCell`

- Είναι αντίστοιχης λειτουργίας με το cons της ML
- Θέλουμε οι λίστες στη Java να είναι αντικειμενοστρεφείς: όπου η ML εφαρμόζει `::` σε μια λίστα, το αντικείμενο-λίστα σε Java πρέπει να είναι σε θέση να εφαρμόσει τη μέθοδο `ConsCell` στον εαυτό του
- Η ML εφαρμόζει `length` σε μια λίστα. Οι λίστες σε Java πρέπει να είναι σε θέση να υπολογίσουν το μήκος τους
- Κατά συνέπεια, δε μπορούμε να χρησιμοποιήσουμε `null` για την κενή λίστα


```
/**
 * An IntList is a list of ints.
 */
public class IntList {
  private ConsCell start; // list head, or null

  /**
   * Construct a new IntList given its first ConsCell.
   * @param s the first ConsCell in the list, or null
   */
  public IntList(ConsCell s) {
    start = s;
  }

  /**
   * Cons the given element h onto us and return the
   * resulting IntList.
   * @param h the head int for the new list
   * @return the IntList with head h, and us as tail
   */
  public IntList cons (int h) {
    return new IntList(new ConsCell(h, start));
  }
```

```
  /**
   * Get our length.
   * @return our int length
   */
  public int length() {
    int len = 0;
    ConsCell cell = start;
    while (cell != null) { // while not at end of list
      len++;
      cell = cell.getTail();
    }
    return len;
  }
}
```


Χρήση της `IntList`

ML:
```
val a = nil;
val b = 2::a;
val c = 1::b;
val x = (length a) + (length b) + (length c);
```

Java:
```
IntList a = new IntList(null);
IntList b = a.cons(2);
IntList c = b.cons(1);
int x = a.length() + b.length() + c.length();
```


Τι είναι μια αναφορά;

- Μια **αναφορά (reference)** είναι μια τιμή που προσδιορίζει μονοσήμαντα κάποιο συγκεκριμένο αντικείμενο

```
public IntList(ConsCell s) {
  start = s;
}
```

- Αυτό που περνάμε ως όρισμα στον κατασκευαστή `IntList` δεν είναι ένα αντικείμενο–είναι μια αναφορά σε ένα αντικείμενο
- Αυτό που αποθηκεύεται στη μεταβλητή `start` δεν είναι ένα αντίγραφο του αντικειμένου αλλά μια αναφορά στο συγκεκριμένο αντικείμενο (το οποίο δεν αντιγράφεται)

Δείκτες

- Σε μια γλώσσα όπως η C ή η C++, υπάρχει ένας εύκολος τρόπος να σκεφτόμαστε τις αναφορές: μια αναφορά είναι ένας δείκτης (pointer)
- Με άλλα λόγια, μια αναφορά είναι η διεύθυνση ενός αντικειμένου στη μνήμη
- Τα συστήματα Java μπορούν, αν θέλουν, να υλοποιήσουν τις αναφορές με αυτόν τον τρόπο

Ναι, αλλά νόμιζα ότι…

- Έχω ακούσει από κάποιους ότι η Java είναι σαν τη C++ αλλά *χωρίς δείκτες…*
- Το παραπάνω είναι αληθές από μια οπτική γωνία
- Η C και η C++ **κάνουν προφανή την πολύ στενή σχέση μεταξύ διευθύνσεων και δεικτών** (π.χ. επιτρέπουν αριθμητική σε δείκτες)
- Τα προγράμματα σε Java **δε μπορούν να καταλάβουν πώς υλοποιούνται οι αναφορές**: οι αναφορές είναι απλά τιμές που προσδιορίζουν μοναδικά κάθε αντικείμενο

Σύγκριση μεταξύ Java και C++

- Μια μεταβλητή στη C++ μπορεί να έχει ως τιμή ένα αντικείμενο ή ένα δείκτη σε ένα αντικείμενο
- Υπάρχουν δύο επιλογείς:
  - `a->x` επιλέγει μια μέθοδο ή ένα πεδίο `x` όταν το `a` είναι ένας δείκτης σε ένα αντικείμενο
  - `a.x` επιλέγει το `x` όταν το `a` είναι ένα αντικείμενο
- Μια μεταβλητή στη Java δε μπορεί να έχει ως τιμή ένα αντικείμενο, μόνο μια αναφορά σε ένα αντικείμενο
- Δηλαδή υπάρχει μόνο ένας επιλογέας:
  - `a.x` επιλέγει το `x` όταν το `a` είναι μια αναφορά σε ένα αντικείμενο

Σύγκριση C++ και Java

Πρόγραμμα σε C++	Αντίστοιχο στη Java

`IntList* p;`
`p = new IntList(0);`
`p->length();`
`p = q;`
	`IntList p;`
	`p = new IntList(null);`
	`p.length();`
	`p = q;`

`IntList p(0);`
`p.length();`
`p = q;`
	Δεν υπάρχει αντίστοιχο

Σύντομες οδηγίες χρήσης για τη Java

Εκτύπωση κειμένου εξόδου

- Υπάρχει το προκαθορισμένο αντικείμενο: **`System.out`**
- Το οποίο έχει δύο μεθόδους:
  - **`print(x)`** που τυπώνει το **`x`**, και
  - **`println(x)`** που τυπώνει το **`x`** και ένα χαρακτήρα νέας γραμμής
- Οι μέθοδοι αυτοί είναι υπερφορτωμένες για όλους τους τύπους παραμέτρων

Εκτύπωση μιας `IntList`

```
/**
 * Print ourself to System.out.
 */
public void print() {
  System.out.print("[");
  ConsCell a = start;
  while (a != null) {
    System.out.print(a.getHead());
    a = a.getTail();
    if (a != null) System.out.print(",");
  }
  System.out.println("]");
}
```


Η μέθοδος `main`

- Μια κλάση μπορεί να έχει μια μέθοδο **`main`** ως εξής:

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

- Η μέθοδος αυτή χρησιμοποιείται ως το σημείο έναρξης της κλάσης όταν αυτή τρέξει ως εφαρμογή
- Η λέξη κλειδί **`static`** την κάνει μια μέθοδο της κλάσης (class method). Πρέπει να χρησιμοποιείται με φειδώ!

Η κλάση TestList

```
public class TestList {
  public static void main(String[] args) {
    IntList a = new IntList(null);
    IntList b = a.cons(2);
    IntList c = b.cons(1);
    int x = a.length() + b.length() + c.length();
    a.print();
    b.print();
    c.print();
    System.out.println(x);
  }
}
```

Μετάφραση και τρέξιμο του προγράμματος

- Τρεις κλάσεις προς μετάφραση, σε τρία αρχεία:
  `ConsCell.java`, **`IntList.java`** και **`TestList.java`**
- (Όνομα αρχείου = όνομα κλάσης + **`.java`**)
- Μεταφράζουμε τα αρχεία με χρήση της εντολής **`javac`**
  - Μπορούν να μεταγλωττιστούν ένα προς ένα
  - Ή με χρήση της εντολής **`javac TestList.java`** όλα μαζί

- Ο compiler παράγει **`.class`** αρχεία
- Χρησιμοποιούμε τον Java launcher (εντολή **`java`**) για να τρέξουμε τη μέθοδο **`main`** ενός **`.class`** αρχείου