

Εισαγωγή στη γλώσσα Java

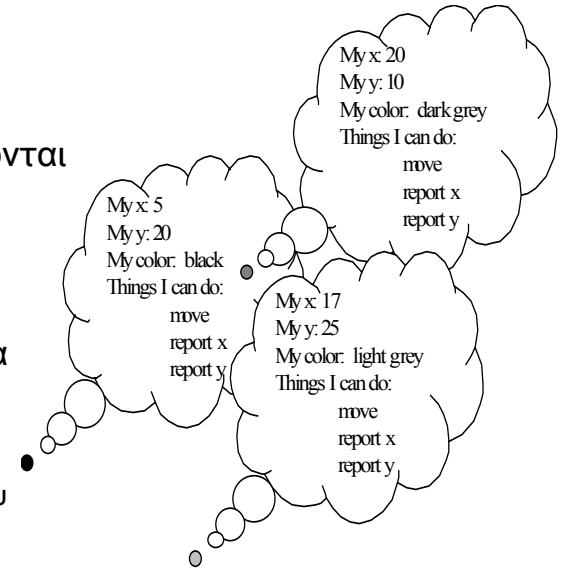


Franz Marc, *Rehe im Walde (II)*, 1913-14

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



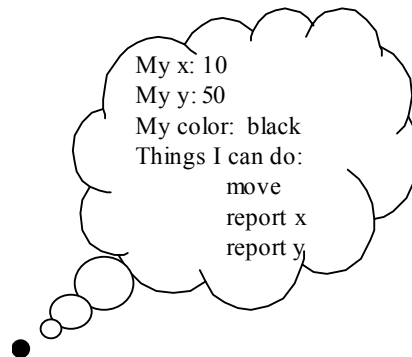
Εισαγωγή στη γλώσσα Java

2

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

Εισαγωγή στη γλώσσα Java

5

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**

Εισαγωγή στη γλώσσα Java

6

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[]**

Εισαγωγή στη γλώσσα Java

7

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



Εισαγωγή στη γλώσσα Java

8

Αριθμητικοί τελεστές

- Για **int**: +, -, *, /, %, μοναδιαίος -

Έκφραση Java	Τιμή
1+2*3	7
15/7	2
15%7	1
-(6*7)	-42

- Για **double**: +, -, *, /, μοναδιαίος -

Έκφραση Java	Τιμή
6.0*7.0	42.0
15.0/7.0	2.142857142857143

Τελεστής συνένωσης

- Ο τελεστής + έχει ειδική υπερφόρτωση και εξαναγκασμό μετατροπής τύπου για την κλάση **String**

Έκφραση Java	Τιμή
"123"+"456"	"123456"
"The answer is " + 42	"The answer is 42"
"" + (1.0/3.0)	"0.3333333333333333"
1 + "2"	"12"
"1" + 2 + 3	"123"
2 + 2 + "2"	"42"

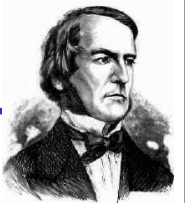
Τελεστές σύγκρισης

- Υπάρχουν οι συνήθεις τελεστές σύγκρισης: <, <=, >=, και >, για αριθμητικούς τύπους
- Επίσης ορίζεται η ισότητα == και η ανισότητα != για κάθε τύπο, συμπεριλαμβανομένου του τύπου **double** (σε αντίθεση με την ML)

Έκφραση Java	Τιμή
1 <= 2	true
1.0 == 2.1	false
true != false	true

Boolean τελεστές

- && και ||, που βραχυκυκλώνουν (όπως οι τελεστές **andalso** και **orelse** της ML)
- !, όπως ο τελεστής **not** της ML
- a?b:c, όπως το **if a then b else c** της ML



Έκφραση Java	Τιμή
1 <= 2 && 2 <= 3	true
1 < 2 1 > 2	true
1 < 2 ? 3 : 4	3

Τελεστές με παρενέργειες

- Ένας τελεστής έχει μια **παρενέργεια (side effect)** εάν αλλάζει κάτι στο περιβάλλον του προγράμματος, όπως για παράδειγμα την τιμή μιας μεταβλητής ή ένα στοιχείο ενός πίνακα
- Στην ML είδαμε μόνο **αγνούς (pure) τελεστές** – δηλαδή τελεστές χωρίς παρενέργειες
- Στη Java υπάρχουν και τελεστές με παρενέργειες

Αναθέσεις, Rvalues και Lvalues

- **a = b**: αλλάζει την τιμή του **a** και την κάνει ίση με τη **b**
- Η ύπαρξη αναθέσεων είναι ένα σημαντικό συστατικό και χαρακτηριστικό των προστακτικών γλωσσών
- Γιατί η ανάθεση **a = 1** έχει νόημα, αλλά όχι η **1 = a** ;
- Οι εκφράσεις στα δεξιά του = πρέπει να έχουν μια τιμή: π.χ. **a**, **1**, **a+1**, **f()** (εκτός αν είναι **void**), κ.λπ.
- Οι εκφράσεις στα αριστερά του = πρέπει να είναι θέσεις μνήμης: π.χ. **a** ή **d[2]**, αλλά όχι **1** ή **a+1**
- Τα δύο αυτά χαρακτηριστικά των εκφράσεων πολλές φορές αναφέρονται ως **rvalue** και **lvalue**

Τελεστές με παρενέργειες στη Java

- Σύνθετες αναθέσεις

Έκφραση Java	Σύντομη Έκφραση Java
a = a+b	a += b
a = a-b	a -= b
a = a*b	a *= b

- Αύξηση και μείωση κατά ένα

Έκφραση Java	Σύντομη Έκφραση Java
a = a+1	a++
a = a-1	a--

Τιμές και παρενέργειες

- Οι εκφράσεις με παρενέργειες έχουν τόσο τιμή όσο και κάποια παρενέργεια
- Η τιμή της ανάθεσης **x = y** είναι η τιμή του **y** και η παρενέργειά της είναι να αλλάξει την τιμή του **x** σε αυτήν την τιμή

Έκφραση Java	Τιμή	Παρενέργεια
a+(x=b)+c	το άθροισμα των a , b και c	αλλάζει την τιμή του x , και την κάνει ίση με b
(a=d)+(b=d)+(c=d)	τρεις φορές η τιμή του d	αλλάζει τις τιμές των a , b και c , και τις κάνει όλες ίσες με το d
a=b=c	η τιμή του c	αλλάζει τις τιμές των a και b , και τις κάνει ίσες με το c

Εκφράσεις αύξησης και μείωσης κατά ένα

- Οι τιμές χρήσης των τελεστών αύξησης και μείωσης κατά ένα εξαρτώνται από την τοποθέτησή τους

Έκφραση Java	Τιμή	Παρενέργεια
a++	η παλιά τιμή του a	προσθέτει ένα στο a
++a	η νέα τιμή του a	προσθέτει ένα στο a
a--	η παλιά τιμή του a	αφαιρεί ένα από το a
--a	η νέα τιμή του a	αφαιρεί ένα από το a

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
r.toUpperCase()	ένα αντικείμενο String που είναι το String r αλλά με κεφαλαία
r.charAt(3)	η τιμή του χαρακτήρα στη θέση 3 στο String r (δηλαδή, ο τέταρτος του χαρακτήρας)
r.toUpperCase().charAt(3)	η τιμή του χαρακτήρα στη θέση 3 στο String r 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"

Σύνταξη των κλήσεων μεθόδων

- Κλήση μεθόδου στιγμιότυπου:

```
<method-call> ::= <reference-expression>.<method-name>  
(<parameter-list>)
```

- Κλήση μεθόδου κλάσης:

```
<method-call> ::= <class-name>.<method-name>  
(<parameter-list>)
```

- Οποιαδήποτε από τις παραπάνω κλήσεις, αλλά από μια μέθοδο της ίδιας κλάσης:

```
<method-call> ::= <method-name>(<parameter-list>)
```

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

Εισαγωγή στη γλώσσα Java

21

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;

Εισαγωγή στη γλώσσα Java

22

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)

Εισαγωγή στη γλώσσα Java

23

Κάποιες αριθμητικές μετατροπές τύπων

- Από **char** σε **int** πριν εφαρμοστεί κάποιος αριθμητικός τελεστής (εκτός της συνένωσης συμβολοσειρών)
- Από **int** σε **double** για δυαδικούς τελεστές που περιλαμβάνουν και τους δύο τύπους

Έκφραση Java	Τιμή
'a' + 'b'	195
1/3	0
1/3.0	0.3333333333333333
1/2+0.0	0.0
1/(2+0.0)	0.5

Εισαγωγή στη γλώσσα Java

24

Εντολές εκφράσεων (expression statements)

- Όπως σε όλες τις προστακτικές γλώσσες, οι εντολές εκτελούνται για τις παρενέργειές τους

<expression-statement> ::= <expression> ;

- Η τιμή της έκφρασης, εάν υπάρχει, απορρίπτεται
- Η Java δεν επιτρέπει στην έκφραση να είναι κάτι χωρίς παρενέργειες, π.χ. **x == y**

Εντολή Java	Επεξήγηση
speed = 0;	Αποθήκευσε την τιμή 0 στη speed .
a++;	Αύξησε την τιμή του a κατά 1.
inTheRed = cost > balance;	Εάν η τιμή του cost είναι μεγαλύτερη από balance , θέσε την τιμή της inTheRed σε true , αλλιώς σε false .

Σύνθετες εντολές (compound statements)

<compound-statement> ::= { <statement-list> }
<statement-list> ::= <statement> <statement-list> | <empty>

- Οι εντολές εκτελούνται με τη σειρά που εμφανίζονται
- Οι σύνθετες εντολές χρησιμοποιούνται και ως μπλοκ που δηλώνουν την εμβέλεια των μεταβλητών

Εντολή Java	Επεξήγηση
{ a = 0; b = 1; }	Αποθήκευσε μηδέν στο a , μετά αποθήκευσε ένα στο b .
{ a++; b++; c++; }	Αύξησε το a , μετά αύξησε το b , μετά αύξησε το c .
{ }	Μην κάνεις τίποτα.

Εντολές δηλώσεων

<declaration-statement> ::= <declaration> ;
<declaration> ::= <type> <variable-name>
| <type> <variable-name> = <expression>

- Ορισμός μεταβλητών με εμβέλεια μπλοκ

boolean done = false;	Ορίζει μια νέα μεταβλητή με όνομα done τύπου boolean , και την αρχικοποιεί σε false .
Point p;	Ορίζει μια νέα μεταβλητή με όνομα p τύπου Point . (και δεν την αρχικοποιεί.)
{ int temp = a; a = b; b = temp; }	Ανταλλάζει τις τιμές των ακέραιων μεταβλητών a και b .

Η εντολή if

<if-statement> ::= if (<expression>) <statement>
| if (<expression>) <statement> else <statement>

- Το ξεκρέμαστο **else** επιλύεται με το συνήθη τρόπο

Εντολή Java	Επεξήγηση
if (i > 0) i--;	Μείωσε το i , αλλά μόνο εάν είναι μεγαλύτερο από το μηδέν.
if (a < b) b -= a; else a -= b;	Αφαίρεσε το μικρότερο από τα a και b από το μεγαλύτερο.
if (reset) { a = b = 0; reset = false; }	Εάν η τιμή της reset είναι true , μηδένισε τα a και b και θέσε την τιμή της reset σε false .

Η εντολή `while`

```
<while-statement> ::= while (<expression>) <statement>
```

- Αποτίμησε την έκφραση *expression* – εάν είναι **false** μην κάνεις τίποτε
- Αλλιώς εκτέλεσε το *statement* και επανάλαβε
- Η επανάληψη είναι άλλο ένα χαρακτηριστικό των προστακτικών γλωσσών προγραμματισμού
- (Παρατηρήστε ότι επανάληψη χωρίς παρενέργειες δεν έχει νόημα, διότι η τιμή της έκφρασης πρέπει να αλλάζει)
- Εκτός από **while** η Java έχει επίσης **do** και **for** loops

Εντολή Java	Επεξήγηση
<code>while (a < 100) a += 5;</code>	Όσο το a είναι μικρότερο του 100, εξακολουθήσε να προσθέτεις 5 στο a .
<code>while (a != b) if (a < b) b -= a; else a -= b;</code>	Αφαίρεσε το μικρότερο των a και b από το μεγαλύτερο, ξανά και ξανά μέχρι να γίνουν ίσοι. (Αλγόριθμος του Ευκλείδη.)
<code>while (time > 0) { simulate(); time--; }</code>	Όσο η μεταβλητή time είναι μεγαλύτερη του μηδενός, κάλεσε τη μέθοδο simulate της τρέχουσας κλάσης και στη συνέχεια μείωσε κατά ένα την time .
<code>while (true) work();</code>	Κάλεσε τη μέθοδο work της τρέχουσας κλάσης ξανά και ξανά, για πάντα.

Η εντολή `return`

```
<return-statement> ::= return <expression>;  
                      | return;
```

- Οι μέθοδοι που επιστρέφουν κάποια τιμή πρέπει να εκτελέσουν μια εντολή **return** της πρώτης μορφής
- Οι μέθοδοι που δεν επιστρέφουν κάποια τιμή (δηλαδή οι μέθοδοι που έχουν ορισθεί ως **void**) μπορούν να εκτελέσουν μια εντολή **return** της δεύτερης μορφής

Ορισμοί κλάσεων

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
<pre>IntList* p; p = new IntList(0); p->length(); p = q;</pre>	<pre>IntList p; p = new IntList(null); p.length(); p = q;</pre>
<pre>IntList p(0); p.length(); p = q;</pre>	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 Driver

```
class Driver {
    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 `Driver.java`
- (Όνομα αρχείου = όνομα κλάσης + `.java`)
- Μεταφράζουμε τα αρχεία με χρήση της εντολής `javac`
 - Μπορούν να μεταγλωττιστούν ένα προς ένα
 - Η με χρήση της εντολής `javac Driver.java` όλα μαζί
- Ο compiler παράγει `.class` αρχεία
- Χρησιμοποιούμε τον Java launcher (εντολή `java`) για να τρέξουμε τη μέθοδο `main` ενός `.class` αρχείου