

# Εισαγωγή στη γλώσσα Java

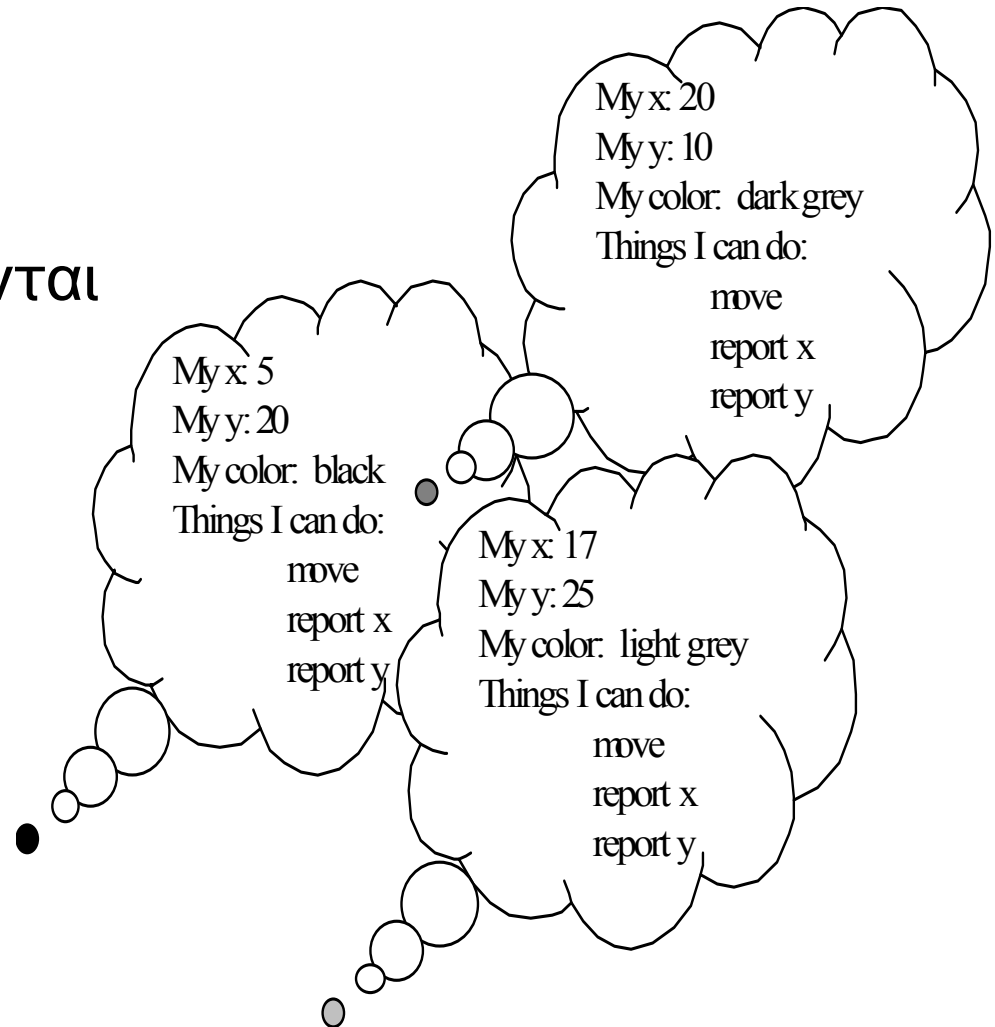


Franz Marc, *Rehe im Walde (II)*, 1913-14

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

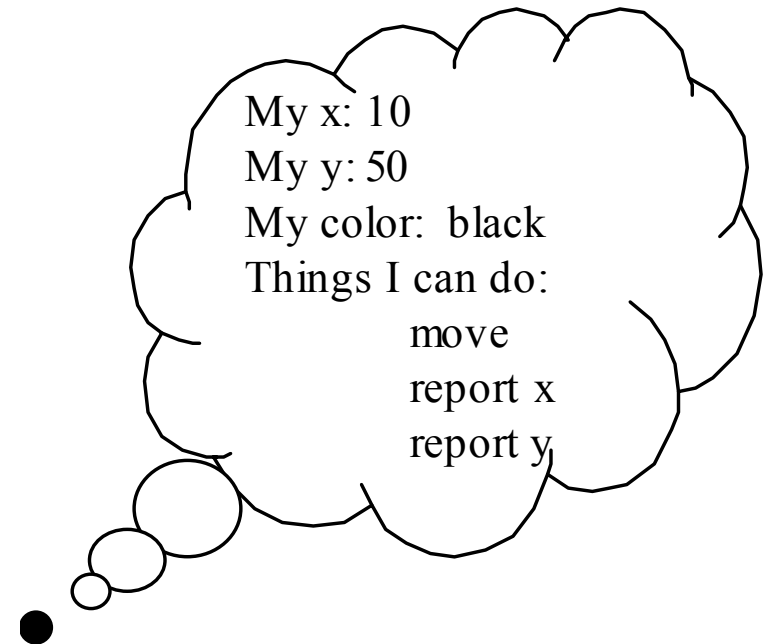
# Παράδειγμα αντικειμενοστρεφούς τρόπου σκέψης

- 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;  
}
```

*field definitions*

```
    public int currentX() {  
        return x;  
    }
```

```
    public int currentY() {  
        return y;  
    }
```

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

*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





# Αριθμητικοί τελεστές

---

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

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

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

Έκφραση Java	Τιμή
<code>6.0*7.0</code>	42.0
<code>15.0/7.0</code>	2.142857142857143

# Τελεστής συνένωσης

---

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

Έκφραση Java	Τιμή
<code>"123" + "456"</code>	<code>"123456"</code>
<code>"The answer is " + 42</code>	<code>"The answer is 42"</code>
<code>"" + (1.0/3.0)</code>	<code>"0.33333333333333333333"</code>
<code>1 + "2"</code>	<code>"12"</code>
<code>"1" + 2 + 3</code>	<code>"123"</code>
<code>2 + 2 + "2"</code>	<code>"42"</code>

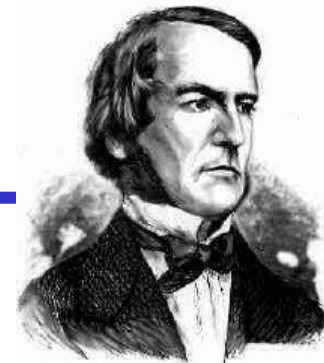
# Τελεστές σύγκρισης

---

- 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
<code>1 &lt;= 2</code>	<code>true</code>
<code>1.0 == 2.1</code>	<code>false</code>
<code>true != false</code>	<code>true</code>

# Boolean τελεστές



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

Έκφραση Java	Τιμή
<code>1 &lt;= 2 &amp;&amp; 2 &lt;= 3</code>	<code>true</code>
<code>1 &lt; 2    1 &gt; 2</code>	<code>true</code>
<code>1 &lt; 2 ? 3 : 4</code>	<code>3</code>

# 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  $a = 42$  έχει νόημα, αλλά όχι η  $42 = a$ ;
- Οι εκφράσεις στα δεξιά του  $=$  πρέπει να έχουν μια τιμή: π.χ.  $a$ ,  $42$ ,  $a+42$ ,  $f()$  (εκτός αν είναι `void`), κ.λπ.
- Οι εκφράσεις στα αριστερά του  $=$  πρέπει να είναι θέσεις μνήμης: π.χ.  $a$  ή  $d[42]$ , αλλά όχι  $42$  ή  $a+42$
- Τα δύο αυτά χαρακτηριστικά των εκφράσεων πολλές φορές αναφέρονται ως **rvalue** και **lvalue**

# Τελεστές με παρενέργειες στη Java

---

- Σύνθετες αναθέσεις

<b>Έκφραση Java</b>	<b>Σύντομη Έκφραση Java</b>
<code>a = a+b</code>	<code>a += b</code>
<code>a = a-b</code>	<code>a -= b</code>
<code>a = a*b</code>	<code>a *= b</code>

- Αύξηση και μείωση κατά ένα

<b>Έκφραση Java</b>	<b>Σύντομη Έκφραση Java</b>
<code>a = a+1</code>	<code>a++</code>
<code>a = a-1</code>	<code>a--</code>

# Τιμές και παρενέργειες

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

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



# Εκφράσεις αύξησης και μείωσης κατά ένα

---

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

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

# Κλήσεις μεθόδου στιγμιότυπου (instance method)

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

<b>Έκφραση Java</b>	<b>Τιμή</b>
<code>String.valueOf(1==2)</code>	"false"
<code>String.valueOf(6*7)</code>	"42"
<code>String.valueOf(1.0/3.0)</code>	"0.33333333333333333333"

# 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> )
```



# 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 προσηταιριστικοί, 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)

# 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
<code>'a'+'b'</code>	195
<code>1/3</code>	0
<code>1/3.0</code>	0.3333333333333333
<code>1/2+0.0</code>	0.0
<code>1/(2+0.0)</code>	0.5



# 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
<code>speed = 42;</code>	Αποθήκευσε την τιμή 42 στη <code>speed</code> .
<code>a++;</code>	Αύξησε την τιμή του <code>a</code> κατά 1.
<code>inTheRed = cost &gt; balance;</code>	Εάν η τιμή του <code>cost</code> είναι μεγαλύτερη από <code>balance</code> , θέσε την τιμή της <code>inTheRed</code> σε <code>true</code> , αλλιώς σε <code>false</code> .

# 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
<pre>{     a = 0;     b = 1; }</pre>	Αποθήκευσε μηδέν στο <b>a</b> , μετά αποθήκευσε ένα στο <b>b</b> .
<pre>{     a++;     b++;     c++; }</pre>	Αύξησε το <b>a</b> , μετά αύξησε το <b>b</b> , μετά αύξησε το <b>c</b> .
<pre>{ }</pre>	Μην κάνεις τίποτα.

# Εντολές δηλώσεων

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

- Ορισμός μεταβλητών με εμφάνιση μπλοκ

<pre><b>boolean done = false;</b></pre>	Ορίζει μια νέα μεταβλητή με όνομα <b>done</b> τύπου <b>boolean</b> , και την αρχικοποιεί σε <b>false</b> .
<pre><b>Point p;</b></pre>	Ορίζει μια νέα μεταβλητή με όνομα <b>p</b> τύπου <b>Point</b> . (και δεν την αρχικοποιεί.)
<pre>{   <b>int temp = a;</b>   <b>a = b;</b>   <b>b = temp;</b> }</pre>	Ανταλλάζει τις τιμές των ακέραιων μεταβλητών <b>a</b> και <b>b</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
<pre><b>if</b> (i &gt; 0) i--;</pre>	Μείωσε το <b>i</b> , αλλά μόνο εάν είναι μεγαλύτερο από το μηδέν.
<pre><b>if</b> (a &lt; b) b -= a; <b>else</b> a -= b;</pre>	Αφαίρεσε το μικρότερο από τα <b>a</b> και <b>b</b> από το μεγαλύτερο.
<pre><b>if</b> (reset) {     a = b = 0;     reset = <b>false</b>; }</pre>	Εάν η τιμή της <b>reset</b> είναι <b>true</b> , μηδένισε τα <b>a</b> και <b>b</b> και θέσε την τιμή της <b>reset</b> σε <b>false</b> .

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

# 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

---

<b>Πρόγραμμα σε C++</b>	<b>Αντίστοιχο στη Java</b>
<pre>IntList* p; p = new IntList(0); p-&gt;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
  - Ή 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