



Προγραμματισμός II (Java)

1. Εισαγωγή

Στόχοι

- Όσοι ολοκληρώνουν το μάθημα με επιτυχία να γνωρίζουν:
 - Τις βασικές αρχές του αντικειμενοστρεφούς προγραμματισμού: αντικείμενα, κλάσεις, γνωρίσματα, μεθόδους, αφαιρετικούς τύπους δεδομένων, ενθυλάκωση, κληρονομικότητα
 - Τα βασικά στοιχεία μιας γλώσσας προγραμματισμού με αντικείμενα όπως η Java: κλάσεις και διεπαφές αντικειμένων, εξαιρέσεις μεθόδων και χειρισμός συλλογών αντικειμένων
 - Πώς να καθορίζουν τη λογική για την επίλυση ενός προβλήματος σύμφωνα με το υπόδειγμα του αντικειμενοστρεφούς προγραμματισμού και κατόπιν να είναι σε θέση να την υλοποιούν σε ένα Java πρόγραμμα
 - Πώς να τεκμηριώνουν και να προετοιμάζουν ένα επαγγελματικό πρόγραμμα χρησιμοποιώντας τα εργαλεία της Java

Θεματολογία

- Εισαγωγή στην Java και στα αντικείμενα
- Σύνδεση με το δομημένο προγραμματισμό
 - έλεγχος ροής προγράμματος
 - τελεστές σύγκρισης
 - λογικοί τελεστές
- Αντικείμενα και κλάσεις, σχεδίαση αντικειμένων, αρχικοποίηση και κατασκευαστές
- Μέθοδοι, παράμετροι, δημιουργία και χρήση μεθόδων, τοπικές μεταβλητές, αποτελέσματα και τιμές επιστροφής, υπερφόρτωση και υπέρβαση μεθόδων,
- Προδιαγραφές κλάσης, στατικά μέλη, κατασκευαστές

Θεματολογία

- Κληρονομικότητα και σύνθεση
- Ενθυλάκωση των χαρακτηριστικών (private) και ελεγχόμενη πρόσβαση με μεθόδους (public)
- Υπερφόρτωση μεθόδων και κατασκευαστών
- Κληρονομικότητα μεταξύ κλάσεων και υπέρβαση μεθόδων
- Τερματισμός κληρονομικότητας με τη final
- this και super
- Αφηρημένες κλάσεις και μέθοδοι. Διεπαφές
- Πολυμορφισμός (late binding)

Θεματολογία

- Πίνακες και λίστες (ArrayLists)
 - Διάσχιση και εκτύπωση περιεχομένων
 - Εισαγωγή
 - Διαγραφή
 - Αναζήτηση
 - Τελεστές =, ==, μέθοδος equals
 - Το interface Comparable
- Καταχώρηση δεδομένων
- Διαχείριση αρχείων
- Γραφικά και χειρισμός γεγονότων

Διαδικαστικά

- Ώρες μαθήματος:
 - Δευτέρα 2-5 Αμφιθέατρο
 - Εργαστήριο: Μετά από συνεννόηση στο εργ. 2ου
- Διδάσκων: Ηρακλής Βαρλάμης
- Γραφείο: 5.1
- E-mail: varlamis@hua.gr
- Web: <http://eclass.hua.gr/courses/DIT130/index.php>
 - Σημειώσεις, ανακοινώσεις, ασκήσεις, κλπ.



Διδακτικές μέθοδοι

- Θεωρία:

- Θεωρία προγραμματισμού με αντικείμενα

- Εργαστήριο:

- Πρακτική εφαρμογή όσων διδάσκονται στη θεωρία
 - μέσα από το NetBeans για Java

- Εργασίες:

- Ανάπτυξη προγραμμάτων σε Java

Βιβλιογραφία

- Java προγραμματισμός : 8^η έκδοση, H. M. Deitel , P. J. Deitel (μτφρ. Χρυσούλα Απ. Κουτρούμπα). Εκδ. Γκιούρδας Μ. 2010. Αθήνα
- Πλήρες εγχειρίδιο της Java 6, Laura Lemay & Rogers Cadenhead. Εκδ. Γκιούρδας. 2007. Αθήνα.
- Εισαγωγή στην Java, 3^η έκδοση, Γ. Λιακέας. Εκδ.Κλειδάριθμος ΕΠΕ. 2008. Αθήνα
- Στο e-class: διαφάνειες και ασκήσεις που δίνονται στα εργαστήρια

Βαθμολογία

- Βαθμολογία (ισχύει για Ιούνιο και Σεπτέμβριο):
 - Τελική εξέταση: 60% [βαθμός ≥ 5]
 - Εργασίες (2-3): 40% [βαθμός ≥ 5 , **υποχρεωτική**]
 - Πιθανό bonus ως 10% σε καλές εργασίες
- Τελική εξέταση: *με ανοιχτό βιβλίο και σημειώσεις μαθήματος*
- Κατοχύρωση:
 - Εργασίες δίνετε **μόνο** μέσα στο εξάμηνο και ο βαθμός τους κατοχυρώνεται για το Σεπτέμβριο
 - Τίποτε δεν κατοχυρώνεται για επόμενη χρονιά!

Απαιτούμενα και καλές πρακτικές

■ Μαθήματα

- "Εισαγωγή στον Προγραμματισμό"
- "Προγραμματισμός Ι"
- «Αντικειμενοστραφής Προγραμματισμός»

■ Καλές πρακτικές

- Παρακολούθηση διαλέξεων και τήρηση σημειώσεων.
- Μελέτη διαφανειών και σημειώσεων μετά από κάθε διάλεξη.
- Εξάσκηση στον υπολογιστή κατά το διάβασμα.
- Ενεργή συμμετοχή στα εργαστήρια.
- Παρακολούθηση των περιοχών συζητήσεων.
- Σοβαρή ενασχόληση με τις εργασίες.
- Επανάληψη μέσω των διαφανειών πριν τις εξετάσεις.



Ας ξεκινήσουμε

Τι θα πρέπει να ξεχάσουμε

- Τη λογική του δομημένου προγραμματισμού
- Μια εφαρμογή Java χρησιμοποιεί τις 4 δομές:
 - ☐ Ακολουθία
 - ☐ Επιλογή
 - ☐ Επανάληψη
 - ☐ Αναδρομή

αλλά δεν στηρίζεται σε αυτές

- Η εφαρμογή μας **δεν** πρέπει να είναι μια σειρά από **μεθόδους** που η μια καλεί την άλλη και όλες μαζί διαχειρίζονται **δεδομένα**.

Γιατί δεν αρκεί ο δομημένος προγραμματισμός

- Τα δεδομένα δεν παριστούν πάντοτε οντότητες του πραγματικού κόσμου
- Τα δεδομένα είναι πολύπλοκα και έχουν εξαρτήσεις
- Όταν αλλάξουν τα δεδομένα χρειάζονται σημαντικές αλλαγές σε όλο το πρόγραμμα
- Είναι προτιμότερο να σχεδιάσουμε την εφαρμογή γύρω από οντότητες που αντιλαμβανόμαστε:
Αντικειμενοστρεφής προγραμματισμός

Τι θα πρέπει να θυμόμαστε

- Τα πάντα είναι αντικείμενα. Τα αντικείμενα περιέχουν:
 - Κατάσταση: δεδομένα, χαρακτηριστικά, μεταβλητές
 - Συμπεριφορά: λειτουργίες, μεθόδους
 - Ταυτότητα: είναι μοναδικά
- Κάθε αντικείμενο έχει ένα τύπο (κλάση). Ο τύπος περιγράφει τα δεδομένα που μπορεί να αποθηκεύει (όχι όμως τις τιμές των μεταβλητών) και τις λειτουργίες που μπορεί να επιτελεί το αντικείμενο.
- Όλα τα αντικείμενα ίδιου τύπου μπορούν να λαμβάνουν τα ίδια μηνύματα.
- Κάθε αντικείμενο μπορεί να αποτελείται από άλλα αντικείμενα.
- Μια εφαρμογή σε Java είναι μια συλλογή αντικειμένων που επικοινωνούν μεταξύ τους με μηνύματα.

κλάσεις και αντικείμενα

- Αφηρημένοι τύποι δεδομένων (κλάσεις)
- κλάση: Ένα σύνολο αντικειμένων με κοινά χαρακτηριστικά (data elements) και λειτουργικότητα (functionality)
- Οι κλάση ορίζεται μία φορά και μπορούμε να δημιουργούμε όσα στιγμιότυπα θέλουμε.
- Οι αφηρημένοι τύποι λειτουργούν όπως οι συνήθεις τύποι δεδομένων
- Μεταβλητές κάποιου τύπου \Leftrightarrow αντικείμενα (objects) ή στιγμιότυπα (instances) της κλάσης

Ορισμός κλάσης

```
class Human {  
    boolean alive;  
    int age;  
    String name;
```

Δεδομένα

```
    void born()  
    {
```

```
        alive = true;  
    }
```

```
    void speak()  
    {
```

```
        System.out.println("Hi!");  
    }
```

```
    void incr_age()  
    {
```

```
        age = age + 1;  
    }
```

```
}
```

Λειτουργίες

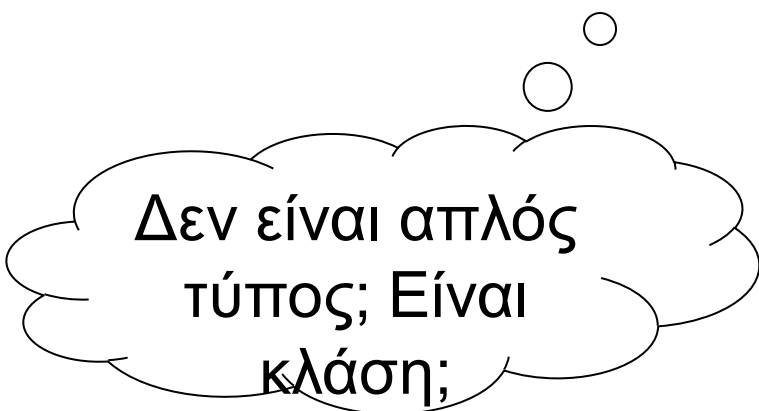
Μέθοδοι

Στόχο έχουν να χειρίζονται τα δεδομένα, να τα τροποποιούν ή να επιστρέφουν τις τιμές αυτών, σε όποιον τις ζητήσει

Κλήση μεθόδου = Αποστολή Μηνύματος

Γνωρίσματα - Δεδομένα

- Τα δεδομένα μπορεί να είναι απλού τύπου: int, boolean, double, float, long κλπ.
- Μπορεί να είναι σύνθετου τύπου. Δηλαδή αντικείμενα μιας άλλης κλάσης: String, Date, Human




Δεν είναι απλός
τύπος; Είναι
κλάση;

Λειτουργίες - Μέθοδοι

- Οι μέθοδοι περιγράφουν τις λειτουργίες της κλάσης πάνω στα δεδομένα. Είναι ο τρόπος με τον οποίο επικοινωνούν τα αντικείμενα διαφόρων κλάσεων μεταξύ τους.
- Μία μέθοδος στη Java έχει:
 - Όνομα
 - Τύπο επιστροφής
 - Λίστα ορισμάτων
- Το σώμα της μεθόδου πρέπει στο τέλος να επιστρέφει τιμή με χρήση της εντολής **return**
- Αν ο τύπος επιστροφής είναι **void** δεν χρειάζεται return.

```
returnType methodName( /* λίστα ορισμάτων */ ) {  
    /* σώμα μεθόδου */  
    return ...;  
}
```

Δημιουργία κλάσης Human

- Τα πάντα στην Java είναι αντικείμενα κάποιας κλάσης. Για το λόγο αυτό σε κάθε μας πρόγραμμα χρειάζεται να ορίσουμε τουλάχιστον μία κλάση. Η κλάση αυτή θα έχει το ίδιο όνομα με το αρχείο που θα την αποθηκεύσουμε. π.χ. Η κλάση Human θα πρέπει να αποθηκευθεί στο αρχείο Human.java
- Κλήση του Java compiler
 - `javac Human.java`
 - παράγει  ένα αρχείο Human.class

Στο
εργαστήριο

1. Αρκεί μόνο η κλάση Human;

- Θεωρητικά αρκεί. Αλλά είναι προτιμότερο:
- Η εφαρμογή που θα διαχειρίζεται αντικείμενα της κλάσης Human θα είναι ένα java application σε μια δεύτερη κλάση με το όνομα Demo στο αρχείο Demo.java
- Για να μπορέσει η εφαρμογή Demo να εκτελεστεί πρέπει να έχει μια μέθοδο με το όνομα **main**.
- Η μέθοδος αυτή θα φτιάχνει αντικείμενα της κλάσης Human και θα καλεί τις μεθόδους τους.

Αντικείμενο = Χαρακτηριστικά + Μέθοδοι

- Για να χρησιμοποιήσουμε ένα αντικείμενο πρέπει πρώτα να το δημιουργήσουμε
- Για να στείλουμε μήνυμα σε ένα αντικείμενο, δίνουμε το όνομα του αντικειμένου μία τελεία και το όνομα της διαθέσιμης λειτουργίας
- Παρόμοια δίνουμε τιμή στα χαρακτηριστικά ενός αντικειμένου
- Το σύνολο των διαθέσιμων λειτουργιών ενός αντικειμένου αποτελεί τη διεπαφή του με τα υπόλοιπα αντικείμενα

```
Human john = new Human();  
// Δημιουργείται ένας human
```

```
john.speak();  
john.age=0;
```



Human
age alive
born() speak() incr_age()

Τα αντικείμενα και η μνήμη

- Όταν δημιουργούμε ένα αντικείμενο στη Java, δεσμεύουμε γι' αυτό μνήμη και χρησιμοποιούμε πάντα μια αναφορά σ' αυτό.
- Η εντολή: *Human John*;
ορίζει την αναφορά John σε ένα Human αντικείμενο, αλλά δεν δεσμεύει μνήμη γι' αυτό.
- Η εντολή: *Human John = **new** Human()*;
δεσμεύει μνήμη για ένα αντικείμενο Human και ορίζει μια αναφορά John σ' αυτό.

Κλήση μεθόδων

- Μια μέθοδος καλείται από κάποιο αντικείμενο (συνήθως) π.χ.

```
Human John= new Human();
```

```
John.born();
```

```
John.speak();
```

```
John.setAge(10);
```

- Μπορούμε να αναφερθούμε απευθείας στα δεδομένα του αντικειμένου.

```
John.age=15;
```

```
int z=John.getAge();
```

Η μέθοδος main

- Η μέθοδος αυτή πρέπει να είναι διαθέσιμη σε όλες τις κλάσεις άρα να δηλωθεί **public**.
- Η μέθοδος πρέπει να μπορεί να κληθεί χωρίς να έχει δημιουργηθεί αντικείμενο της συγκεκριμένης κλάσης, άρα να δηλωθεί **static**.
- Τέλος πρέπει να παίρνει ορίσματα (παραμέτρους) από τη γραμμή εντολών και δεν είναι απαραίτητο να επιστρέφει κάποια τιμή.
- Έτσι προκύπτει η ανάγκη για μια μέθοδο στην κύρια κλάση της εφαρμογής μας που ορίζεται ως εξής:

```
class Demo{  
    public static void main(String args[]){  
        /* σώμα της main*/  
        Human john=new Human();  
        john.born();  
        john.speak();  
    }  
}
```


Μεταγλώττιση και εκτέλεση

- Κλήση του Java compiler
 - `javac Demo.java`
Παράγει ένα αρχείο `Demo.class`
- **Πρέπει** το `Human.class` να είναι στον ίδιο φάκελο
- Κλήση του Java Virtual Machine
 - `java Demo`
τυπώνει το μήνυμα στην οθόνη

2. Έλεγχος πρόσβασης - Ενθυλάκωση

- Είναι καλή πρακτική να μην επιτρέπουμε πρόσβαση σε όλα τα χαρακτηριστικά ή τις μεθόδους μιας κλάσης.
 - Για να αποκρύψουμε από τους προγραμματιστές που θα χρησιμοποιήσουν την κλάση, τα μέρη της κλάσης που δεν χρειάζεται και δεν πρέπει να μεταβάλλουν
 - Για να μπορούμε να αλλάζουμε την εσωτερική δομή μιας κλάσης χωρίς να επηρεάζεται ο τρόπος επικοινωνίας της με τις υπόλοιπες.
- Με τον τρόπο αυτό η διεπαφή της κλάσης απλουστεύεται και περιορίζεται μόνο στα απαραίτητα

Τα όρια μιας κλάσης

- Υπάρχουν τρεις τύποι πρόσβασης στα μέρη μιας κλάσης: ***public***, ***private***, και ***protected***.
- Δημόσια (***public***), τα μέρη είναι διαθέσιμα σε όλους
- Ιδιωτική (***private***), τα μέρη είναι διαθέσιμα μόνο στην ίδια την κλάση και τις λειτουργίες της
- Προστατευμένη (***protected***), τα μέρη είναι διαθέσιμα στην κλάση και σε όλες κλάσεις την κληρονομούν.
- Δεδομένη (***default***) πρόσβαση, όταν δεν δηλώνεται κάποια από τις προηγούμενες, τα μέρη είναι διαθέσιμα στην κλάση και σε όλες τις υπόλοιπες κλάσεις της ίδιας εφαρμογής (του ίδιου package)

Παράδειγμα

- Αν δε δηλώσουμε κάτι διαφορετικό, η κλάση Demo έχει πρόσβαση και στις μεθόδους και στα χαρακτηριστικά των αντικειμένων τύπου Human.
`John.age=15;` ⇔ `John.setAge(15);`
- Αν όμως κάνουμε το age private

```
class Human{  
    private int age;  
    ...  
}
```
- Τότε η Demo δεν μπορεί να αναφερθεί στο age μέσα από αντικείμενα τύπου Human.
`John.setAge(15);`
- Αντίθετα εντός της Human, μπορούμε να αναφερθούμε στο age.
π.χ. Στο σώμα των setAge και getAge
- Το ίδιο μπορεί να γίνει και για τις μεθόδους μιας κλάσης

Μέθοδοι

- Οι μέθοδοι μιας κλάσης μπορεί να είναι:
 - Μέθοδοι πρόσβασης (get) που διαβάζουν τις τιμές των χαρακτηριστικών
 - Μέθοδοι τροποποίησης (set) που τροποποιούν τις τιμές των χαρακτηριστικών
 - Μέθοδοι κατασκευής (constructor) που αρχικοποιούν τις τιμές των χαρακτηριστικών κατά τη δημιουργία των αντικειμένων.
 - Άλλες βοηθητικές μέθοδοι.

Νέος ορισμός κλάσης

```
class Human {  
    private boolean alive;  
    private int age;  
    void Human()  
    {  
        alive = true;  
        age = 0  
    }  
    int getAge()  
    {  
        return age;  
    }  
    void setAge(int a)  
    {  
        age=a;  
    }  
    void speak()  
    {  
        System.out.println("Hi!");  
    }  
    void incr_age()  
    {  
        age = age +1;  
    }  
}
```

Συμπερασματικά

- Σε ένα πρόβλημα:
 - Προσπαθώ να δω: ποια αντικείμενα εμπλέκονται, τι δεδομένα έχει το καθένα από αυτά και ποιες λειτουργίες επιτελεί και στη συνέχεια ορίζω τις αντίστοιχες κλάσεις
- Σε μια κλάση
 - Ορίζω χαρακτηριστικά: name, surname, age
 - Ορίζω μεθόδους πρόσβασης: getName(), setName() ..
 - Ορίζω άλλες βοηθητικές μεθόδους
- Στο τέλος
 - Ορίζω μια κλάση με μια μέθοδο main()
 - Η κλάση αυτή εκτελεί το πρόγραμμά μου. Μόνο που δεν περιέχει όλη τη λύση. Κάποια πράγματα τα αναλαμβάνουν τα αντικείμενα

Βασικοί τύποι δεδομένων

- Για τους βασικούς τύπους (primitive types) η Java δημιουργεί μεταβλητές και όχι αναφορές στη μνήμη.

- Δήλωση μεταβλητής

```
int age;  
//(τύπος) (Όνομα Μεταβλητής)
```

- Υπάρχουν αντίστοιχες τάξεις (wrapper types) για καθέναν από αυτούς, π.χ.

```
char c = 'x';  
Character C = new Character(c);  
Character C = new Character('x');
```

Primitive type	Size
boolean	—
char	16-bit
byte	8-bit
short	16-bit
int	32-bit
long	64-bit
float	32-bit
double	64-bit
void	—

Wrapper type
Boolean
Character
Byte
Short
Integer
Long
Float
Double
Void

Διαφορές

Απλός τύπος

- τιμή

```
double d= 10.50;
```

- ένας μόνο τρόπος αρχικοποίησης

- δεν έχει μεθόδους

- σύγκριση τιμών

```
if (d==10.50) ←true
```

Wrapper type

- αντικείμενο

```
Double d1 = new Double(10.50);
```

- πολλοί τρόποι αρχικοποίησης
- ```
Double d2=new Double("10.60");
```

```
Double d3=new Double(10.50);
```

- έχει μεθόδους

```
int x=d.intValue(); // x = 10
```

```
int a = d1.compareTo(d2); // a<0
```

- σύγκριση αντικειμένων

```
if (d1==d3) ←false
```

```
if (d1.equals(d3) ←true
```

# Δηλώσεις μεταβλητών παντού

- Οι δηλώσεις των μεταβλητών γίνονται παντού, όχι μόνο στην αρχή ενός block

```
{
 for (int i=0; i< 20; i++) {
 System.out.println("Hi!");
 Char ch='A';
 }
 double pi = 3.14159;
}
```

- Παραμένουν σε ισχύ μέχρι την έξοδο από το block που τις περιβάλλει.
- Οι μεταβλητές πρέπει να αρχικοποιούνται πριν την χρήση τους (Ο compiler της Java επιβάλλει κάτι τέτοιο).

# Εμβέλεια μεταβλητών/αντικειμένων

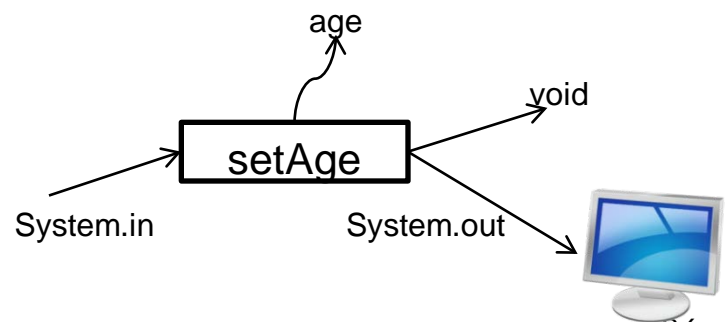
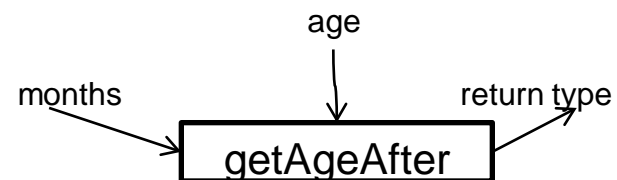
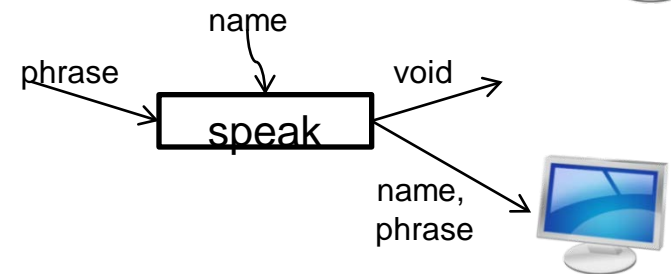
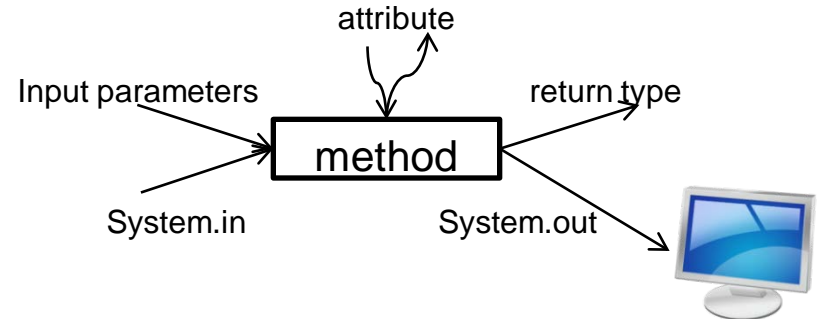
- Κάθε μεταβλητή έχει ισχύ εντός του μπλοκ στο οποίο ορίζεται και σε κάθε μπλοκ που υπάρχει μέσα σε αυτό.

```
{
 int x = 12;
 /* μόνο το x υπάρχει */
 {
 int q = 96;
 /* υπάρχουν τα x & q */
 }
 q=150 /* δεν επιτρέπεται
γιατί μόνο το x υπάρχει */
 /* εκτός της εμβέλειας του q */
}
```

```
{
 int x = 12;
 {
 int x = 96; /* Δεν επιτρέπεται
γιατί έχει ήδη οριστεί το x*/
 }
}
```

# Σε μια κλάση

```
class Human {
 boolean alive;
 int age;
 String name;
 void speak(String phrase)
 {
 System.out.println("Hi!");
 System.out.println("My name is "+name);
 System.out.print("You asked me to say:");
 System.out.println(phrase);
 }
 int getAgeAfter(int months) {
 int years=months/12;
 int ageafter=age+years;
 return ageafter;
 }
 void setAge(){
 Scanner keyboard=new Scanner(System.in);
 System.out.println("How old are you?");
 age=keyboard.nextInt();
 }
}
```



# Προαγωγή τύπου

- Σε αριθμητικές εκφράσεις που περιλαμβάνουν διάφορους τύπους, όλοι οι τύποι προάγονται στον πιο ευρύ από όλους.

```
short s; int i; double d;
```

```
System.out.println(s+i+d); //result is double
```

- Η διαίρεση ακεραίων δίνει μόνο το ακέραιο μέρος;

```
int years=42/12; → years=3
```

```
double years= 42/12; → years=3.0
```

```
double years=(double)42/12; → years=3.5
```

- Υπολογισμοί που αφορούν τους τύπους byte, short, char προάγονται σε υπολογισμούς με int:

```
short s = 4; int t = 6;
```

```
s = (short) (s + t); //cast required
```

# Στένεμα και διεύρυνση τύπων

- Η C ελεύθερα κάνει αναθέσεις μεταξύ αριθμητικών τύπων, αλλά η Java απαιτεί casting για την ανάθεση ενός τύπου σε έναν άλλο μικρότερο.
- Το στένεμα γίνεται από περισσότερο σε λιγότερο πλούσιο τύπο  
Πιθανή απώλεια πληροφορίας, γι'αυτό χρειάζεται το casting  
π.χ. ανάθεση short σε byte, double σε float
- Το πέρασμα παραμέτρων είναι το ίδιο σαν την ανάθεση.
- Δεν μπορούμε να κάνουμε τίποτα cast σε boolean.  
Λάθος: `boolean b = (boolean) num;`  
Σωστό: `boolean b = (num != 0 );`

# Θεμελιώδεις περικλείουσες κλάσεις

- Byte, Boolean, Short, Integer, Long, Float, Double
  - Μπορούν να υπάρχουν επειδή η Java είναι case-sensitive
  - Η κάθε μια τους είναι wrapper κλάση για έναν απλό τύπο
  - Δεν μπορεί να αλλάξει η τιμή τους (τίθεται μόνο στον constructor και δεν υπάρχουν μέθοδοι που να την αλλάζουν)
- Παράδειγμα

```
Integer i = new Integer(15);
Float f = new Float("3.15169");
int k = i.intValue();
double d = f.floatValue();
```

# String type

- Οι σταθερές String περικλείονται με διπλά εισαγωγικά

String s = "a string literal";

String s = new String("A string literal");

- Οι τελεστές + και += υπερφορτώνονται για την ένωση δύο ή περισσοτέρων Strings.

s = "Welcome to" + 1998 + "!";

- Οτιδήποτε μπορεί να μετατραπεί σε αναπαράσταση string (όλοι οι primitive τύποι και τα αντικείμενα)

- Μεγάλη λειτουργικότητα

εύρεση μήκους, ανάκτηση συγκεκριμένων χαρακτήρων  
αναζήτηση για χαρακτήρες και substrings

ένωση strings, εύρεση/αντικατάσταση, σύγκριση μεταξύ strings



# String type

- Τα Strings είναι αμετάβλητα

Οι χαρακτήρες που τα αποτελούν θέτονται στον constructor και δεν αλλάζουν

- Αντιπροσωπευτικές μέθοδοι

- ☐ int **length** ()
- ☐ char **charAt** (int index)
- ☐ int **indexOf** (char ch)
- ☐ int **indexOf** (String s)
- ☐ boolean **equals** (String s)
- ☐ String **toLowerCase** ()
- ☐ String **replace** (char oldChar, char newChar)
- ☐ String **concat** (String s)
- ☐ String **substring** (int offset, int endIndex)
- ☐ String[] **split** (String regex)



# Τελεστές

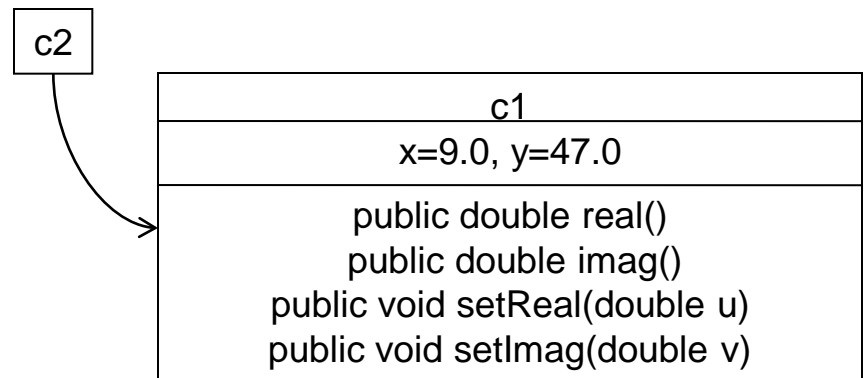
# Τελεστές

- πρόσθεση (+)
- αφαίρεση(-)
- πολλαπλασιασμός (\*)
- διαίρεση(/) (για ακεραίους επιστρέφει το ακέραιο μέρος του αποτελέσματος)
- υπόλοιπο διαίρεσης (%)
- ανάθεση τιμής σε μεταβλητή(=)
- $x+=4;$
- $x-=a;$
- Τελεστές προσαύξησης και μείωσης:  
 $x=a++;$        $x=++a;$   $x=a--;$        $x=--a;$

# Τελεστής ανάθεσης

- Δημιουργώντας ένα αντικείμενο, ουσιαστικά δεσμεύουμε μνήμη και δημιουργούμε μια αναφορά σε αυτή.  
Complex c1 = new Complex();  
Complex c2 = new Complex();  
c1.setReal(9);  
c1.setImag(47);  
c2=c1;
- Όταν αναθέσουμε ένα αντικείμενο (c1) σε ένα άλλο (c2), ουσιαστικά έχουμε δύο αναφορές στην ίδια θέση μνήμης (αυτή που δεσμεύτηκε για το c1)  
c2 = c1;
- Το φαινόμενο αυτό ονομάζεται aliasing (συνωνυμία)

```
public class Complex {
 private double x,y;
 public double real() { return x; }
 public double imag() { return y; }
 public void setReal(double u) {x=u; }
 public void setImag(double v) {y=v; }
}
```



# Σχεσιακοί τελεστές

- Επιστρέφουν true ή false
  - Ισότητα (==)
  - Ανισότητα (!=)
  - Μικρότερο/Μεγαλύτερο (>, <, <=, >=)
- Για objects οι τελεστές συγκρίνουν αναφορές και όχι τα ίδια τα αντικείμενα.
- Για να συγκρίνουμε αντικείμενα χρησιμοποιούμε τη μέθοδο ***equals()***.

```
Integer n1 = new Integer(47);
Integer n2 = new Integer(47);
System.out.println(n1 == n2);
//τυπώνει false
System.out.println(n1 != n2);
//τυπώνει true
System.out.println(n1.equals(n2));
//τυπώνει true
```

# Λογικοί τελεστές

- Επιστρέφουν true ή false

- ☐ AND (&&)
- ☐ OR (||)
- ☐ NOT (!)

- Bitwise operators

- ☐ AND (&) π.χ.  $3 \& 5 \rightarrow 00000011 \& 00000101 \rightarrow 00000001 \rightarrow 1$
- ☐ OR (|)
- ☐ XOR (^)
- ☐ NOT (~)
- ☐ right-shift (>>), left-shift (<<), unsigned right shift (>>>)

# Άλλοι τελεστές

- Τριαδικός (ternary) τελεστής  
boolean-exp ? valueIfTrue : valueIfFalse  
π.χ.  
int x=55;  
String z= (x%2==1?"Περιττός":"Άρτιος");
- Τελεστής ; (comma)  
Χρησιμοποιείται μόνο στη λίστα παραμέτρων του for
- Τελεστής πρόσθεσης String (+)  
int x = 0, y = 1, z = 2;  
String sString = "x, y, z ";  
System.out.println(sString + x + y + z);
- Τελεστές μετατροπής (casting)  
int i = 200;  
long l = (long)i;  
long l2 = (long)200;

# Προτεραιότητες

| Operator type          | Operators                           |
|------------------------|-------------------------------------|
| Unary                  | + - ++--                            |
| Arithmetic (and shift) | * / % + - << >>                     |
| Relational             | > < >= <= == !=                     |
| Logical (and bitwise)  | &&    &   ^                         |
| Conditional (ternary)  | <b>A &gt; B ? X : Y</b>             |
| Assignment             | = (and compound assignment like *=) |



# Η equals στις κλάσεις χρήστη

- Ορίζοντας τη μέθοδο equals καθορίζουμε τον τρόπο με τον οποίο θα συγκρίνονται τα αντικείμενα της κλάσης μας

```
public class Complex {
 private double x,y;

 ...

 public boolean equals(Complex c) {
 if (x==c.x && y==c.y)
 return true;
 else return false;
 }
}
```



# Έλεγχος ροής προγράμματος

# if - else

**if**(συνθήκη εισόδου)

εντολές

**else if**(συνθήκη εισόδου)

εντολές

**else**

εντολές

```
public class IfElse {
 public int test(int testval, int target) {
 int result = 0;
 if(testval > target)
 result = +1;
 else if(testval < target)
 result = -1;
 else
 result = 0; // Match
 return result;
 }
}
```

# return

## ■ To return

- καθορίζει την τιμή που θα επιστρέψει μια μέθοδος
- διακόπτει τη ροή εκτέλεσης της μεθόδου και επιστρέφει την τιμή

```
public class IfElse2 {
 public int test(int testval, int target){
 int result = 0;
 if(testval > target)
 return +1;
 else if(testval < target)
 return -1;
 else
 return 0; // Match
 }
}
```

# while & do-while

**while**(συνθήκη εισόδου)  
    εντολές

**do**  
    εντολές  
**while**(συνθήκη συνέχισης);

Το statement εκτελείται  
τουλάχιστον μία φορά

```
public class WhileTest {
 public static void main(String[] args) {
 double r = Math.random();
 while(r < 0.5)
 {
 System.out.println(r);
 r = Math.random();
 }
 do
 {
 System.out.println(r);
 r = Math.random();
 } while(r < 0.5);
 }
}
```

# for

- **for(αρχικοποίηση; συνθήκη τερματισμού; βήμα)**

**{... ΕΝΤΟΛΕΣ ...}**

- ```
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);  
    }  
}
```
- ```
public class CommaOperator {
 public static void main(String[] args) {
 for(int i = 1, j = i + 10; i < 5; i++, j = i * 2) {
 System.out.println("i= " + i + " j= " + j);
 }
 }
}
```

# break / continue

- Το break διακόπτει την εκτέλεση και βγαίνει από το βρόχο
- Το continue διακόπτει την εκτέλεση της συγκεκριμένης επανάληψης και συνεχίζει από την αρχή του βρόχου
- Προσοχή στους βρόχους που δεν τερματίζουν

```
public class BreakAndContinue {
 public static void main(String[] args)
 {
 for(int i = 0; i < 100; i++) {
 if(i == 74) break;
 // Out of for loop
 if(i % 9 != 0) continue;
 // Next iteration
 System.out.println(i);
 }
 }
}
```

# Επανάληψη και αναδρομή

- Συχνά το ίδιο αποτέλεσμα μπορεί να επιτευχθεί
  - με επαναλαμβανόμενη κλήση μιας μεθόδου για συγκεκριμένο αριθμό επαναλήψεων
  - με αναδρομική κλήση της ίδιας της μεθόδου μέσα στο σώμα της.



# Παράδειγμα με παραγοντικό

- $N! = N * (N-1) * (N-2) * \dots * 1$

- Επαναληπτικό

```
long factorial (int n){
 long result = 1;
 for (int i =n; i>1; i--)
 result *=i;
 return result;
}
```

- $N! = N * (N-1)!$

- Αναδρομικό

```
long factorial (int n){
 if (n<=1)
 return 1;
 return n * factorial(n-1);
}
```



# Γνωρίσματα κλάσης

# static γνώρισμα- μέθοδοι

- Αν ένα γνώρισμα είναι **κοινό** για όλα τα αντικείμενα μιας κλάσης τότε δεσμεύουμε μια φορά χώρο για όλα τα αντικείμενα.
- Κατά τη δήλωση της κλάσης, τη δηλώνουμε static. Το γνώρισμα αυτό θα δημιουργηθεί μία φορά μόλις δηλωθεί το πρώτο αντικείμενο αυτής της κλάσης και θα μπορεί να χρησιμοποιείται από κάθε αντικείμενο της ίδιας κλάσης.  
π.χ. `static float bonus;`
- Αν θέλουμε να αναφερθούμε στο γνώρισμα αυτό, μπορούμε απ' ευθείας μέσω της κλάσης:  
π.χ. `Human.bonus=100,00;`
- Παρόμοια ισχύουν και για τις μεθόδους.  
π.χ. `static setBonus(float b)`  
`Human.setBonus(200,00);`
- Τα static μέλη μπορούμε να τα καλούμε είτε απ'ευθείας μέσω της κλάσης είτε μέσω των αντικειμένων που δημιουργούμε. Οι static μέθοδοι έχουν πρόσβαση στα static μέλη της κλάσης.

# Final γνωρίσματα

- Πρακτικά: `final` = μόνο για ανάγνωση
- Αν ένα γνώρισμα δηλωθεί `final`, τότε λειτουργεί ως σταθερά που
  - πρέπει να αρχικοποιηθεί
  - και δεν αλλάζει τιμή
- Αν η παράμετρος εισόδου μιας μεθόδου δηλωθεί `final` τότε δεν μπορεί να αλλάξει τιμή στο σώμα της μεθόδου.
  - `public void setHour(final int hrs)`
- Αν ένα αντικείμενο δηλωθεί `final` τότε το όνομα του αντικειμένου θα αναφέρεται πάντα στο ίδιο αντικείμενο.
- Αν μια κλάση δηλωθεί `final` τότε δεν μπορεί να κληρονομηθεί από άλλη κλάση.

# Γνωρίσματα static και final

- Ο πιο αποτελεσματικός τρόπος για να υλοποιηθούν σταθερά γνωρίσματα μίας κλάσης είναι ως static final:

```
public class Time {
 private static final int NumHours = 24;
 public static final int NumMinutes = 60;
}
```

- Μπορούν να είναι private αν πρόκειται να χρησιμοποιηθούν μόνο εσωτερικά
- Έχουν μόνο μία θέση αποθήκευσης όσα και αν είναι τα στιγμιότυπα μιας κλάσης και αυτή είναι αμετάβλητη.

```
double totalminutes=280;
double hours=totalminutes/Time.NumMinutes;
double minutes=totalminutes-hours*Time.NumMinutes;
```



# Μέθοδοι

# Δομή μεθόδου

```
πρόσβαση τύποςΕπιστροφής όνομαΜεθόδου (τύπος1 παράμετρος1, ...)
{
 ...
}
```

```
public double addDouble (double num1, double num2)
{
 return num1+num2;
}
```

# Βασικές μέθοδοι

- Μέθοδοι πρόσβασης  
getAttribute, setAttribute  
για ανάγνωση, ενημέρωση  
αντίστοιχα
- Μέθοδος toString  
Επιστρέφει μια String  
περιγραφή του αντικειμένου
- Μέθοδος αρχικοποίησης:  
Κατασκευαστής - Constructor

```
class Human {
 private boolean alive;
 private int age;
 private String name;
 public void setAlive(boolean aAlive)
 {
 alive=aAlive;
 }
 public boolean getAlive()
 {
 return alive;
 }
 ...
 public String toString(){
 return name+" "+age +" "+alive;
 }
 ...
 public Human(boolean aAlive, int aAge, String aName){
 alive=aAlive;
 age=aAge;
 name=aName;
 }
}
```