ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΑΦΗΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ

Κωδικός Θ: ΤΠ4003, Κωδικός Ε: ΤΠ4103 (ΜΕΥ/Υ)
Ώρες (Θ - ΑΠ - Ε): 2 - 1 - 3
Προαπαιτούμενα: ΤΠ2003,2103

Δρ. ΒΙΔΑΚΗΣ ΝΙΚΟΣ

Περιεχόμενα

Κλάσεις (Class)

Κεντρική ιδέα στον αντικειμενοστραφή προγραμματισμό είναι η κλάση (class), μία αυτοτελής και αφαιρετική αναπαράσταση κάποιας κατηγορίας αντικειμένων,

- είτε φυσικών αντικειμένων του πραγματικού κόσμου
- είτε νοητών, εννοιολογικών αντικειμένων, σε ένα περιβάλλον προγραμματισμού.

Πρακτικά είναι ένας τύπος δεδομένων, ή αλλιώς το προσχέδιο μίας δομής δεδομένων με δικά της περιεχόμενα, τόσο μεταβλητές όσο και διαδικασίες.

Τα περιεχόμενα αυτά δηλώνονται είτε ως δημόσια (public) είτε ως ιδιωτικά (private), με τα ιδιωτικά να μην είναι προσπελάσιμα από κώδικα εκτός της κλάσης.

Κλάσεις (Class)

Οι διαδικασίες των κλάσεων συνήθως καλούνται μέθοδοι (methods) και οι μεταβλητές τους γνωρίσματα (attributes) ή πεδία (fields).

Μία κλάση πρέπει ιδανικά να είναι:
εννοιολογικά αυτοτελής, να περιέχει δηλαδή μόνο πεδία τα οποία περιγράφουν μία κατηγορία αντικειμένων και δημόσιες μεθόδους οι οποίες επενεργούν σε αυτά όταν καλούνται από το εξωτερικό πρόγραμμα, χωρίς να εξαρτώνται από άλλα δεδομένα ή κώδικα εκτός της κλάσης,
και επαναχρησιμοποιήσιμη, να αποτελεί δηλαδή μαύρο κουτί δυνάμενο να λειτουργήσει χωρίς τροποποιήσεις ως τμήμα διαφορετικών προγραμμάτων

Κλάσεις (Class)

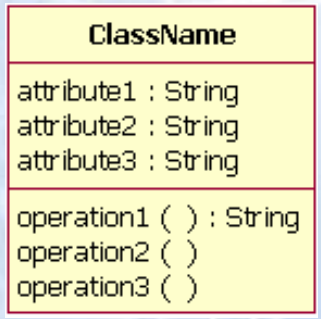
Η κλάση θεωρείται το καλούπι (blueprint) όλων των object που ανήκουν σε αυτήν.

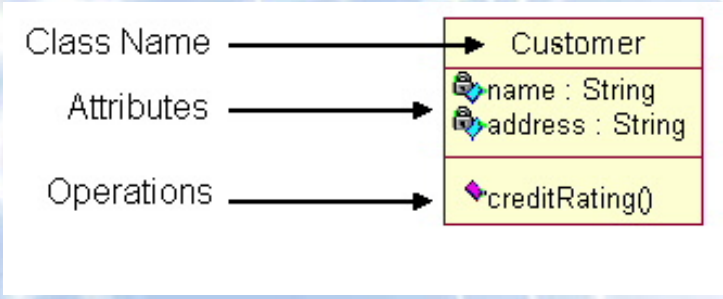
Μία κλάση ή τάξη είναι ένα σχέδιο ή πρότυπο που ορίζει τις κοινές μεταβλητές και μεθόδους όλων των ομοειδών αντικειμένων.

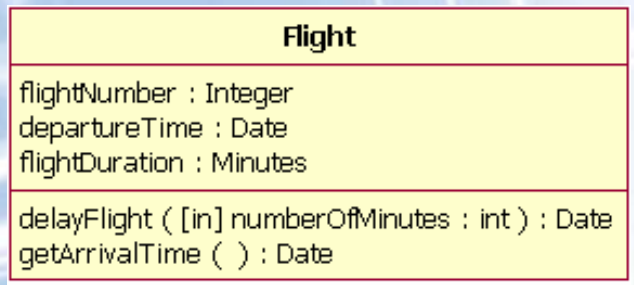
Κάθε αντικείμενο είναι στιγμιότυπο μίας κλάσης.

Κλάσεις (Class)

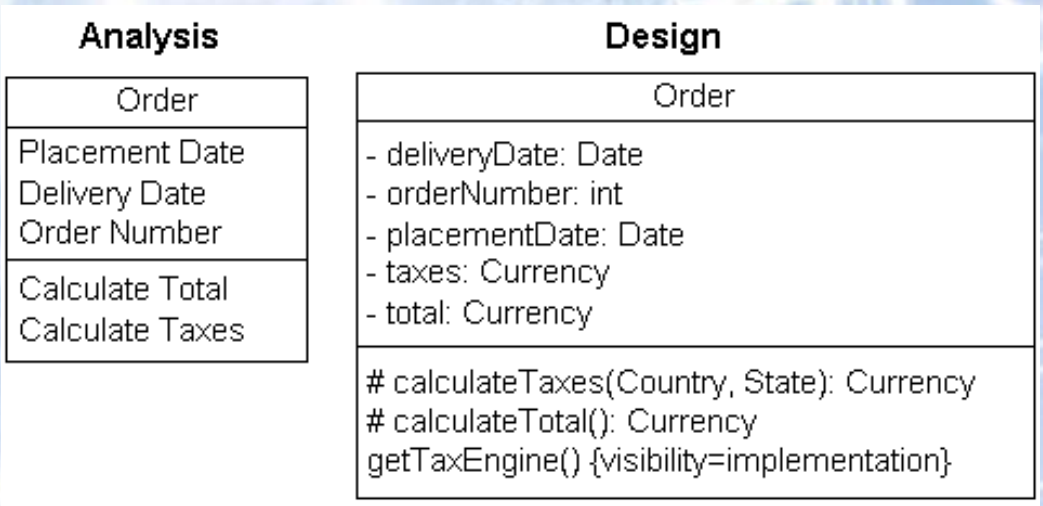
Οι κλάσεις αποτελούνται από 3 πράγματα:
a name,
attributes,
and operations.







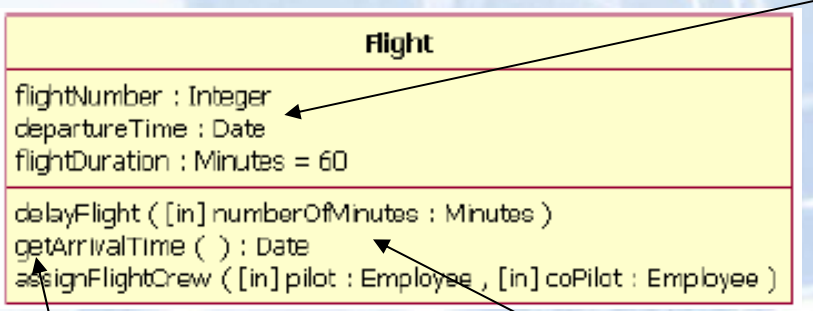
Analysis and design versions of a class



Κλάσεις (Class)

name : attribute type = default value

Attribute Name	Attribute Type
flightNumber	Integer
departureTime	Date
flightDuration	Minutes



Operations
delayFlight: 1 input parameter, no return value
getArrivalTime: 0 parameter, return value
assignFlightCrew: 2 input parameters, no return value

Operation Name	Parameters	Return Value Type
delayFlight	Name: numberOfMinutes, Type: Minutes	N/A
getArrivalTime	N/A	Date

Κλάσεις (Class)

Δημόσιο Μέρος (Public API)

Ιδιωτικό Μέρος (Λεπτομέρειες Υλοποίησης)

Στατικά Χαρακτηριστικά:

- Χρώμα
- Μοντέλο

Δυναμικά Χαρακτηριστικά:
Αλλάξε Ταχύτητα: (Υλοποίηση)
κλπ

Παράδειγμα Κλάσης (Ποδήλατο)

```
class Bicycle {

	int cadence = 0;
	int speed = 0;
	int gear = 1;

	void changeCadence(int newValue) {
		cadence = newValue;
	}

	void changeGear(int newValue) {
		gear = newValue;
	}

	void speedUp(int increment) {
		speed = speed + increment;
	}

	void applyBrakes(int decrement) {
		speed = speed - decrement;
	}

	void printStates() {
		System.out.println("cadence:"+cadence+" speed:"+speed+"
gear:"+gear);
	}
}
```

Αντικείμενα (Objects)

Γενικά

Αντικείμενο (object) είναι το στιγμιότυπο μίας κλάσης, δηλαδή αυτή καθαυτή η δομή δεδομένων (με αποκλειστικά δεσμευμένο χώρο στη μνήμη) βασισμένη στο «καλούπι» που προσφέρει η κλάση.
Παραδείγματος χάρη, σε μία αντικειμενοστραφή γλώσσα προγραμματισμού θα μπορούσαμε να ορίσουμε κάποια κλάση ονόματι BankAccount, η οποία αναπαριστά έναν τραπεζικό λογαριασμό, και να δηλώσουμε ένα αντικείμενο της με όνομα MyAccount. Το αντικείμενο αυτό θα έχει δεσμεύσει χώρο στη μνήμη με βάση τις μεταβλητές και τις μεθόδους που περιγράψαμε όταν δηλώσαμε την κλάση. Έτσι, στο αντικείμενο θα μπορούσε να περιέχεται ένα γνώρισμα Balance (=υπόλοιπο) και μία μέθοδος GetBalance (=επέστρεψε το υπόλοιπο).

Αντικείμενα (Objects)

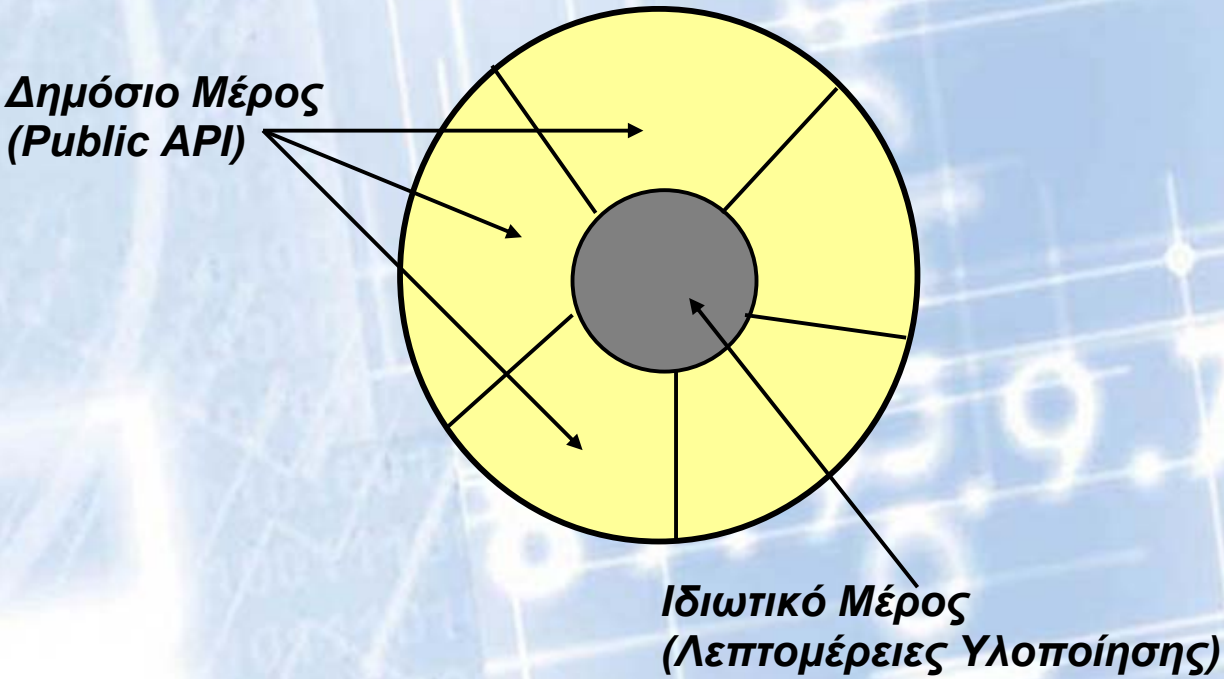
Γενικά

Ακολούθως θα μπορούσαμε να δημιουργήσουμε ακόμα ένα ή περισσότερα αντικείμενα της ίδιας κλάσης τα οποία θα είναι διαφορετικές δομές δεδομένων (διαφορετικοί τραπεζικοί λογαριασμοί στο παράδειγμα).

Ας σημειωθεί εδώ πως τα αντικείμενα μίας κλάσης μπορούν να προσπελάσουν τα ιδιωτικά περιεχόμενα άλλων αντικειμένων της ίδιας κλάσης.

Αντικείμενα (Objects)

Γενικά



Αντικείμενα (Objects)

Γενικά

Real-world objects – Αντικείμενα πραγματικού κόσμου

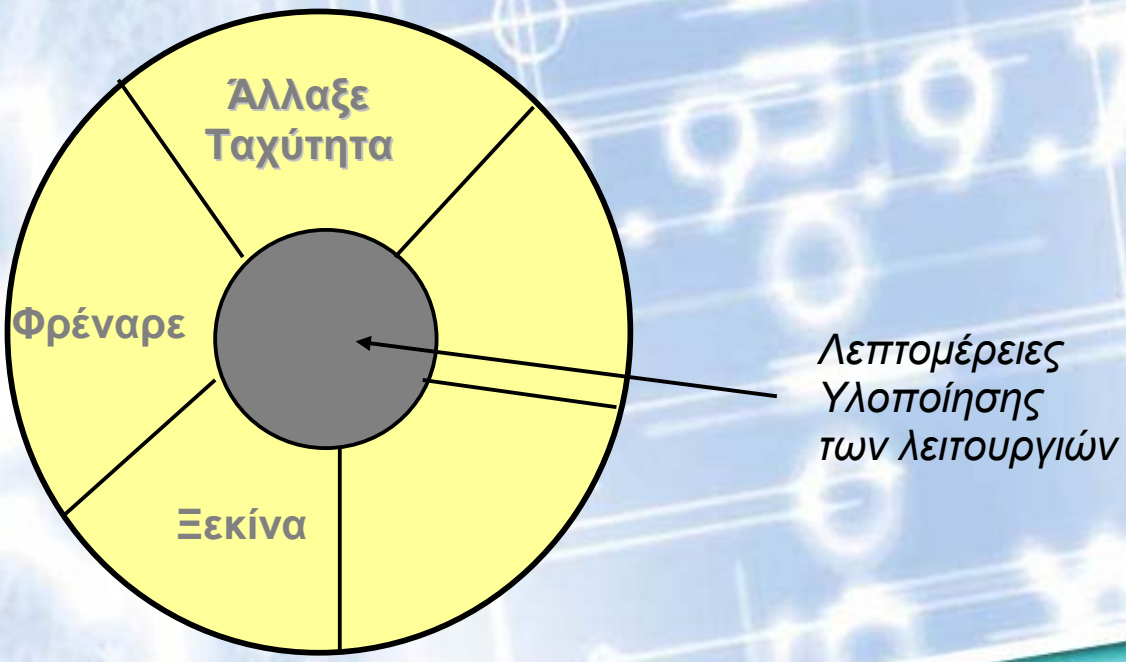
Real-world objects share two characteristics: They all have state and behaviour.

Dogs have:

- state (name, colour, breed, hungry) and
- behaviour (barking, fetching, wagging tail).

Bicycles also have

- state (current gear, current pedal cadence, current speed) and
- behaviour (changing gear, changing pedal cadence, applying brakes).

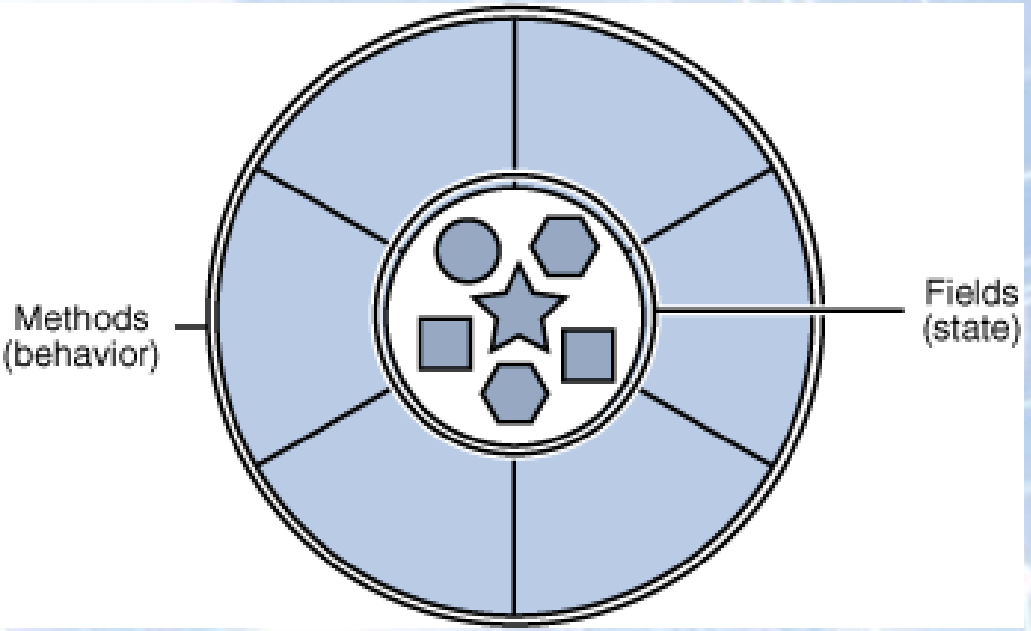


Αντικείμενα (Objects)

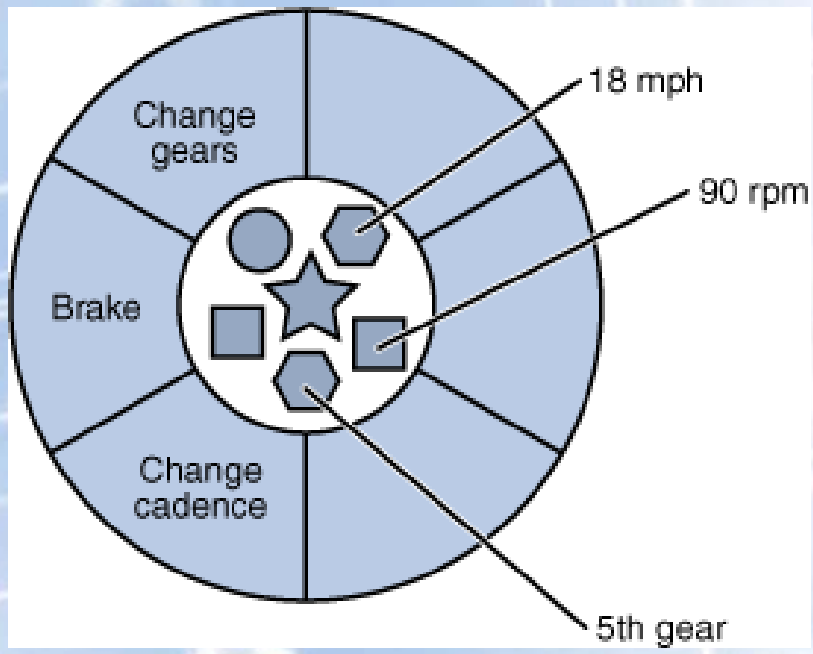
Γενικά

Software Object - Αντικείμενο Λογισμικού

- Μοντελοποιεί, αναπαριστά, αντιστοιχεί σε κάποιο αντικείμενο του πραγματικού κόσμου
- ομαδοποίηση μεταβλητών (στατικό μέρος) και σχετιζομένων μεθόδων (δυναμικό μέρος)



A software object.



A bicycle modelled as a software object.

Αντικείμενα (Objects)

Γενικά

Πλεονεκτήματα δομής κώδικα σε αντικείμενα (Objects)

1. **Modularity**: Ο πηγαίος κώδικας ενός αντικειμένου γράφεται και συντηρείται ανεξάρτητα από τον πηγαίο κώδικα άλλων αντικειμένων.
2. **Information-hiding**: Με το να αλληλεπιδρούμε μόνο με τις μεθόδους του αντικειμένου, οι λεπτομέρειες της υλοποίησης του αποκρύπτονται από τον έξω κόσμο.
3. **Code re-use**: Αν το αντικείμενο υπάρχει (ακόμα και αν είναι γραμμένο από άλλον) τότε μπορείτε να το χρησιμοποιήσετε. Αυτό δίνει την δυνατότητα τα αντικείμενα να υλοποιούνται (implement/test/debug) από ειδικούς στον εκάστοτε τομέα.
4. **Pluggability and debugging ease**: Εάν ένα αντικείμενο παρουσιάζει προβλήματα τότε μπορείτε να το διαγράψετε και να προσθέσετε ένα άλλο με παρόμοια χαρακτηριστικά.

Αντικείμενα (Objects)

Ιδιότητες (Attributes), Μέθοδοι (Methods - Operations)

- Ένα πρόγραμμα δημιουργεί πολλά αντικείμενα.
- Μέσα από την αλληλεπίδραση των αντικειμένων (με τις μεθόδους) το πρόγραμμα εκτελεί διάφορες εργασίες όπως η δημιουργία του GUI, υπολογισμός εξισώσεων, αποστολή και παραλαβή δεδομένων στο δίκτυο κλπ.
- Μόλις το αντικείμενο ολοκληρώσει την εργασία για την οποία δημιουργήθηκε αποδεσμεύει την πηγές που χρησιμοποίησε για να είναι διαθέσιμες προς χρήση από άλλα αντικείμενα

Αντικείμενα (Objects)

Ιδιότητες (Attributes), Μέθοδοι (Methods - Operations)

```
public class CreateObjectDemo {

    public static void main(String[] args) {

     //Declare and create a point object
     //and two rectangle objects.
     Point originOne = new Point(23, 94);
     Rectangle rectOne = new Rectangle(originOne, 100, 200);
     Rectangle rectTwo = new Rectangle(50, 100);

     //display rectOne's width, height, and area
     System.out.println("Width of rectOne: " +
                        rectOne.width);
     System.out.println("Height of rectOne: " +
                        rectOne.height);
     System.out.println("Area of rectOne: " +
rectOne.getArea());

     //set rectTwo's position
     rectTwo.origin = originOne;
```

Αντικείμενα (Objects)

```
        //display rectTwo's position
        System.out.println("X Position of rectTwo: "
                + rectTwo.origin.x);
        System.out.println("Y Position of rectTwo: "
                + rectTwo.origin.y);

        //move rectTwo and display its new position
        rectTwo.move(40, 72);
        System.out.println("X Position of rectTwo: "
                + rectTwo.origin.x);
        System.out.println("Y Position of rectTwo: "
                + rectTwo.origin.y);
    }
}
```

This program creates, manipulates, and displays information about various objects. Here's the output:

```
Width of rectOne: 100
Height of rectOne: 200
Area of rectOne: 20000
X Position of rectTwo: 23
Y Position of rectTwo: 94
X Position of rectTwo: 40
Y Position of rectTwo: 72
```

Αντικείμενα (Objects)

Ιδιότητες (Attributes), Μέθοδοι (Methods - Operations)

Από τον Παραπάνω κώδικα ας απομονώσουμε τις παρακάτω εντολές:

```
Point originOne = new Point(23, 94);
Rectangle rectOne = new Rectangle(originOne, 100, 200);
Rectangle rectTwo = new Rectangle(50, 100);
```

Η πρώτη εντολή δημιουργεί ένα αντικείμενο τύπου **`Point`** ενώ η δεύτερη και τρίτη δημιουργούν αντικείμενα τύπου **`Rectangle`**

Κάθε εντολή έχει τρία μέρη:

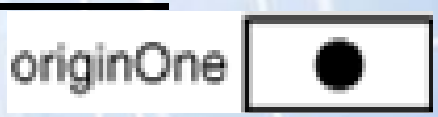
1. **Declaration**: Ο κώδικας σε **bold** είναι «variable declarations» που συνδέει το όνομα τις μεταβλητής με έναν τύπο αντικειμένου.
2. **Instantiation**: η λέξη **`new`** είναι «a Java operator» η οποία δημιουργεί το αντικείμενο.
3. **Initialization**: Η λέξη new ακολουθείται από τον «constructor», ο οποίος αρχικοποιεί το αντικείμενο.

Αντικείμενα (Objects)

Ιδιότητες (Attributes), Μέθοδοι (Methods - Operations)

Declare a Variable

```
Point originOne;
```

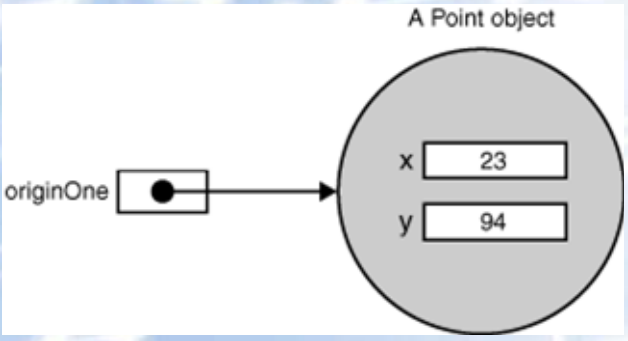


Instantiating a Class

```
Point originOne = new Point(23, 94);
```

Initializing an Object

```
public class Point {
    public int x = 0;
    public int y = 0;
    //constructor
    public Point(int a, int b) {
    x = a;
    y = b;
    }
}
```



Αντικείμενα (Objects)

Ιδιότητες (Attributes), Μέθοδοι (Methods - Operations)

Referencing an Object's Fields

```
objectReference.fieldName

e.g.: System.out.println("Width of rectOne: " +
rectOne.width);
```

Calling an Object's Methods

```
objectReference.methodName(argumentList);
   or
objectReference.methodName();

e.g.: System.out.println("Area of rectOne: " +
rectOne.getArea());

int areaOfRectangle = new Rectangle(100,
50).getArea();
```

Αντικείμενα (Objects)

Ιδιότητες (Attributes), Μέθοδοι (Methods - Operations)

The Garbage Collector

Σε κάποιες γλώσσες πρέπει να κρατάμε ιστορικό των αντικειμένων που δημιουργούνται και να αποδεσμεύουμε τους πόρους του συστήματος καταστρέφοντας τα αντικείμενα όταν αυτά έχουν ολοκληρώσει την εργασία τους.

Στην JAVA την δουλειά αυτή αναλαμβάνει ο «Garbage Collector» ο οποίος εντοπίζει τα αντικείμενα τα οποία έχουν ολοκληρώσει την εργασία τους και δεν έχουν νόημα ύπαρξης. Τα αντικείμενα αυτά καταστρέφονται και απελευθερώνονται οι πόροι του συστήματος.

Αντικείμενα (Objects)

Άσκηση

Φτιάξτε το δικό σας παράδειγμα:

- **Κοιτάξτε γύρω σας και προσδιορίστε μέσα στην τάξη αντικείμενα του πραγματικού κόσμου**
- **Προσδιορίστε τα ανάλογα αντικείμενα λογισμικού.**
- **Δώστε τους state και behavior**

Fundamental Principles of OO Programming

- **Encapsulation,**
- **Polymorphism,**
- **Inheritance**

Fundamental Principles of OO Programming

Encapsulation

Ορισμοί

- **Encapsulation (ουσιαστικό): the condition of being enclosed (as in a capsule)**
- **Ενθυλάκωση (ουσιαστικό): η κατάσταση του να βρίσκεσαι μέσα σε ένα θύλακα (κάψουλα)**
- **Ενθυλάκωση δεδομένων (data encapsulation) καλείται η ιδιότητα που προσφέρουν οι κλάσεις να «κρύβουν» τα ιδιωτικά δεδομένα τους από το υπόλοιπο πρόγραμμα και να εξασφαλίζουν πως μόνο μέσω των δημόσιων μεθόδων τους θα μπορούν αυτά να προσπελαστούν.**

Fundamental Principles of OO Programming

Encapsulation

- **Το αντικείμενο θα πρέπει να είναι αυτό-διαχειριζόμενο, κάθε αλλαγή στο αντικείμενο θα πρέπει να γίνεται μέσα από τις μεθόδους του αντικειμένου.**
  - **Θα πρέπει να κάνουμε πολύ δύσκολο αν όχι ακατόρθωτο να γίνονται αλλαγές από άλλα αντικείμενα.**
  - **Ο χρήστης ενός αντικειμένου μπορεί να καλέσει τις μεθόδους του χωρίς να γνωρίζει τον τρόπο με τον οποίο εκτελούνται αυτές οι μέθοδοι.**
- **Τα παραπάνω κάνουν ευκολότερες τις αλλαγές στην υλοποίηση των μεθόδων ενός αντικειμένου**

Fundamental Principles of OO Programming

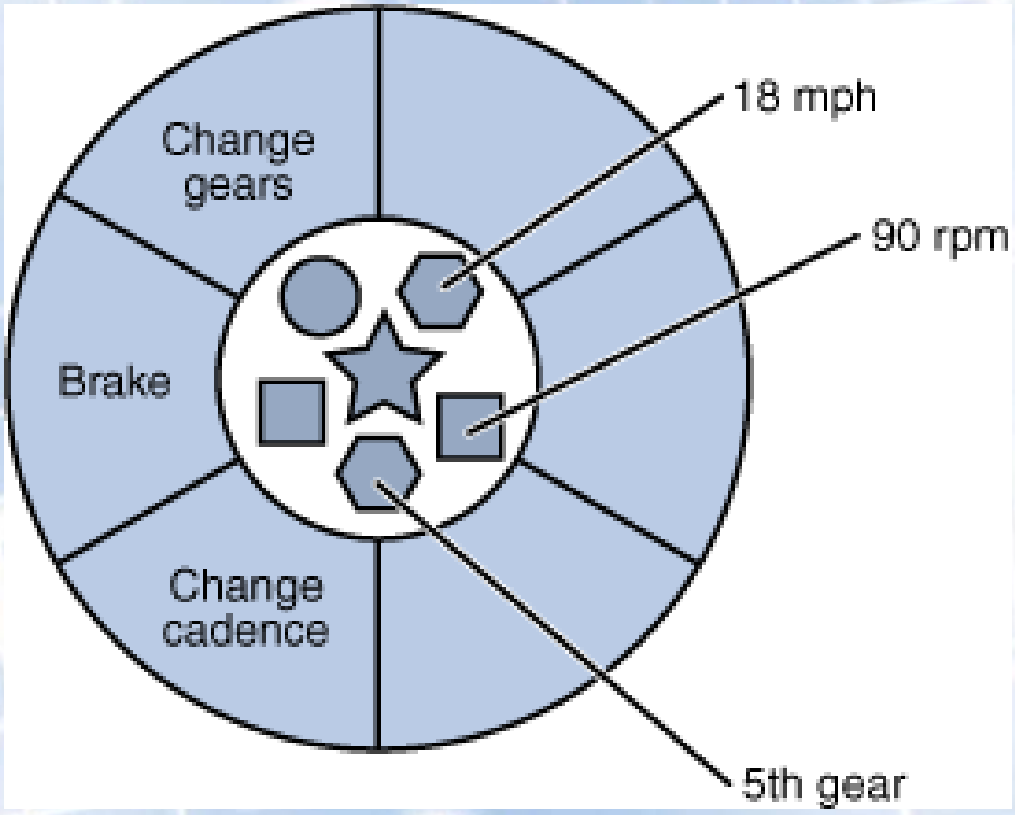
Encapsulation

- Ένα αντικείμενο με ενθυλάκωση μπορεί να θεωρηθεί ως ένα μαύρο κουτί.
  - Η εσωτερική του δομή είναι άγνωστη – κρυφή στα άλλα αντικείμενα, χρήστες ή εξυπηρετητές.
- Ποιος ο λόγος για encapsulation
  - Προγραμματίζοντας με την έννοια του *black box*, ο χρήστης του αντικειμένου χρησιμοποιεί τις διασυνδέσεις τους αντικειμένου χωρίς να γνωρίζει την εσωτερική του δομή,
  - Η διασυνδέσεις δεν μπορούν να παρακαμφθούν,
  - Η ενθυλάκωση κάνει την διαχείριση του αντικειμένου εύκολη καθώς όλη η επικοινωνία με αυτό περνά από καλά προσδιορισμένες υπηρεσίες (services)

Fundamental Principles of OO Programming

Encapsulation

- **Η διασύνδεση του αντικειμένου γίνεται μέσω των μεθόδων του.**
  - **Change Gear**
  - **Brake**
  - **Change Cadence**



Fundamental Principles of OO Programming

Encapsulation Level

- **Η Ενθυλάκωση προσδιορίζεται στην κλάση και όχι στο αντικείμενο (στιγμιότυπο της κλάσης).**

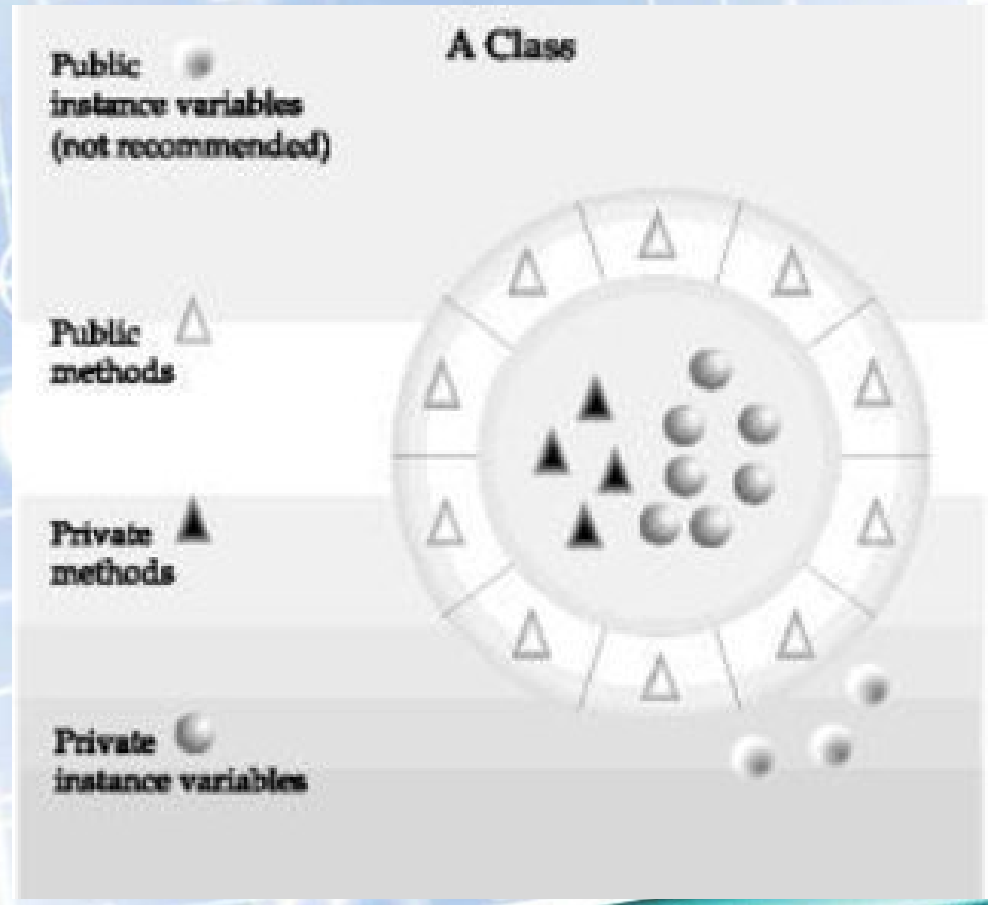
- **Στην Java αντικείμενα της ίδιας κλάσης έχουν πρόσβαση σε ιδιωτικά (private) μέρη άλλων αντικειμένων της ίδιας κλάσης.**

Fundamental Principles of OO Programming

Encapsulation

- **Η Ενθυλάκωση υλοποιείται μέσω κατάλληλων *visibility modifiers***

- **Ο λόγος που προστατεύονται τα αντικείμενα είναι:**
  - **Πολλές φορές ένα data element εξαρτάτε από άλλα,**
  - **Μόνο κάποιες μεταβλητές ανήκουν στην δημόσια διασύνδεση,**
  - **Ελαχιστοποιούνται τα bugs.**



Fundamental Principles of OO Programming

Encapsulation

visibility modifiers in Java

Access Levels				
Modifier	**Class**	**Package**	**Subclass**	**World**
public	Y	Y	Y	Y
protected	Y	Y	Y	N
no modifier	Y	Y	N	N
private	Y	N	N	N

Encapsulation
(Αρχές / Ιδιότητες)

- Information Hiding
- Interface and Implementations
- Standardization
- Access Control mechanisms (private /public etc.)

Fundamental Principles of OO Programming

Polymorphism

Ο ορισμός του Πολυμορφισμού (κατά Webster):

Πολυμορφισμός (ουσιαστικό): a genetic variation that produces differing characteristics in individuals of the same population or species

Πολύμορφος (επίθετο): having, assuming, or passing through many or various forms, stages, or the like

Οι βασικές αρχές της γενίκευσης και της αφαίρεσης είναι:

- Reuse
- Interoperability

Η βασική ιδέα: Ο προγραμματιστής χρησιμοποιεί μια γενική διασύνδεση (general interface), και η Java επιλέγει την σωστή μέθοδο

Fundamental Principles of OO Programming

Polymorphism

Αρχές

Αν η κλάση Β είναι υποκλάση της Α τότε μπορούμε να αντικαταστήσουμε ένα στιγμιότυπο της Α με ένα στιγμιότυπο της Β

Αν η Α ανταποκρίνεται στο μήνυμα m τότε η Β ανταποκρίνεται στο ίδιο μήνυμα m

Fundamental Principles of OO Programming

Polymorphism

Overloaded methods are methods with the same ***name signature*** but either a different number of parameters or different types in the parameter list. For example 'spinning' a number may mean increase it, 'spinning' an image may mean rotate it by 90 degrees. By defining a method for handling each type of parameter you achieve the effect that you want.

Overridden methods are methods that are redefined within an inherited or subclass. They have the ***same*** signature and the subclass definition is used.

Fundamental Principles of OO Programming

Polymorphism

Polymorphism is the capability of an action or ***method*** to do different things based on the object that it is acting upon.

Overloading and overriding are two types of polymorphism .

A third type: ***dynamic method binding***.
Assume that three subclasses (Cow, Dog and Snake) have been created based on the Animal abstract class, each having their own speak() method.
Notice that although each method reference was to an Animal (but no animal objects exist), the program is able to resolve the correct method related to the subclass object at runtime. This is known as dynamic (or late) method binding.

Fundamental Principles of OO Programming

Polymorphism

```
public class AnimalReference
{
  public static void main(String args[])
  Animal ref                  // set up var for an Animal
  Cow aCow = new Cow("Bossy"); // makes specific objects
  Dog aDog = new Dog("Rover");
  Snake aSnake = new Snake("Ernie");
  // now reference each as an Animal
  ref = aCow; ref.speak();
  ref = aDog; ref.speak();
  ref = aSnake; ref.speak();
}
```

Fundamental Principles of OO Programming

Polymorphism

Polymorphism (Αρχές / Ιδιότητες)

- Many forms of same function
- Abstract Methods
- Abstract Classes
- Interfaces

Fundamental Principles of OO Programming

Inheritance

- Κληρονομικότητα ονομάζεται η ιδιότητα των κλάσεων να επεκτείνονται σε νέες κλάσεις, ρητά δηλωμένες ως κληρονόμους (υποκλάσεις ή 'θυγατρικές κλάσεις'), οι οποίες μπορούν να επαναχρησιμοποιήσουν τις μεταβιβάσιμες μεθόδους και ιδιότητες της γονικής τους κλάσης αλλά και να προσθέσουν δικές τους.
- Στιγμιότυπα των θυγατρικών κλάσεων μπορούν να χρησιμοποιηθούν όπου απαιτούνται στιγμιότυπα των γονικών (εφόσον η θυγατρική είναι κατά κάποιον τρόπο μία πιο εξειδικευμένη εκδοχή της γονικής), αλλά το αντίστροφο δεν ισχύει.

Fundamental Principles of OO Programming

Inheritance

- Παράδειγμα κληρονομικότητας είναι μία γονική κλάση Vehicle (Όχημα) και οι δύο πιο εξειδικευμένες υποκλάσεις της Car (Αυτοκίνητο) και Bicycle (Ποδήλατο), οι οποίες λέμε ότι "κληρονομούν" από αυτήν.
- Πολλαπλή κληρονομικότητα είναι η δυνατότητα που προσφέρουν ορισμένες γλώσσες προγραμματισμού μία κλάση να κληρονομεί ταυτόχρονα από περισσότερες από μία γονικές. Από μία υποκλάση μπορούν να προκύψουν νέες υποκλάσεις που κληρονομούν από αυτήν, με αποτέλεσμα μία ιεραρχία κλάσεων που συνδέονται μεταξύ τους "ανά γενιά" με σχέσεις κληρονομικότητας.

Fundamental Principles of OO Programming

Inheritance

- *Υποκλάσεις* (subclasses) και *υπερκλάσεις* (superclasses).
- Κάθε υποκλάση κληρονομεί τη συμπεριφορά και την κατάσταση της υπερκλάσης της.
- Οι υποκλάσεις μπορούν να επανεγγράψουν κληροδοτούμενη συμπεριφορά και κατάσταση.
- Ιεραρχία κλάσεων

Πλεονεκτήματα:

- Οι υποκλάσεις προσφέρουν *εξειδικευμένη* συμπεριφορά από τα κοινά στοιχεία που προσφέρει η υπερκλάση. Επαναχρησιμοποίηση του κώδικα της υπερκλάσης.
- Οι προγραμματιστές μπορούν να υλοποιήσουν υπερκλάσεις που λέγονται *αφαιρετικές κλάσεις* (abstract classes) και ορίζουν γενικευμένη συμπεριφορά, η οποία εξειδικεύεται από τις αντίστοιχες υποκλάσεις. Οι αφαιρετικές κλάσεις δεν έχουν στιγμιότυπα

Inheritance Principles (Αρχές / Ιδιότητες)

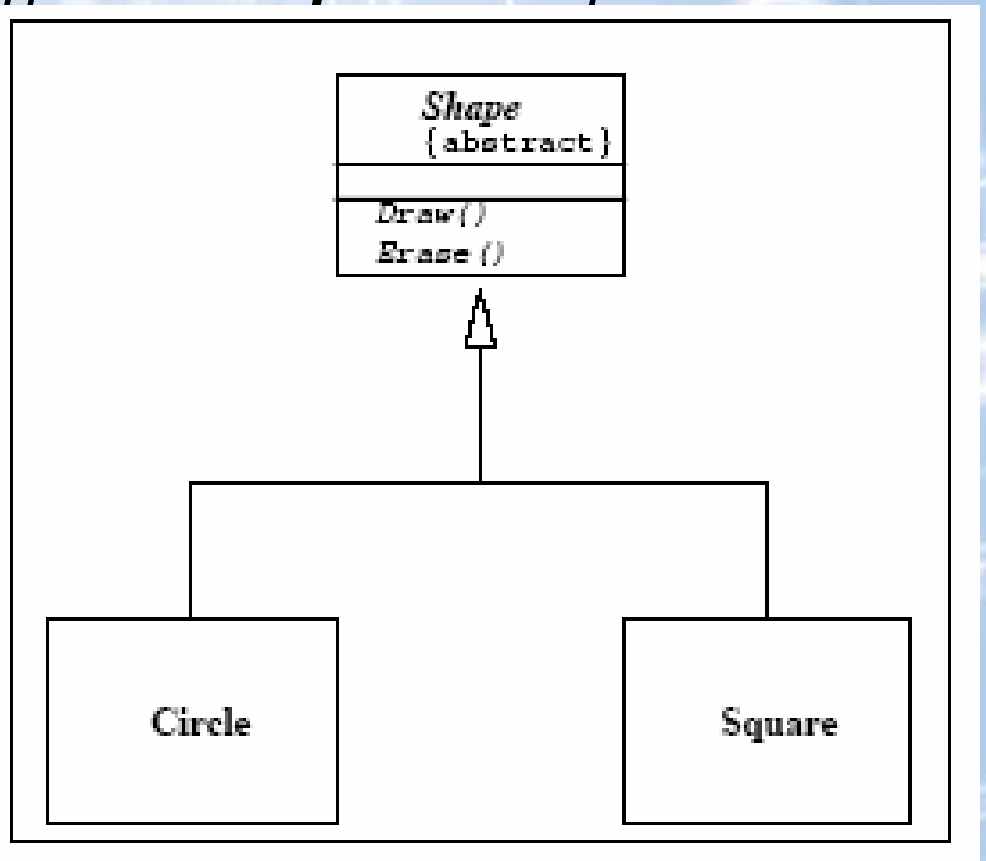
- Hierarchy
- Reusability
- Extensibility
- Expressive power
- Reflects many real-world problems

Fundamental Principles of OO Programming

Inheritance

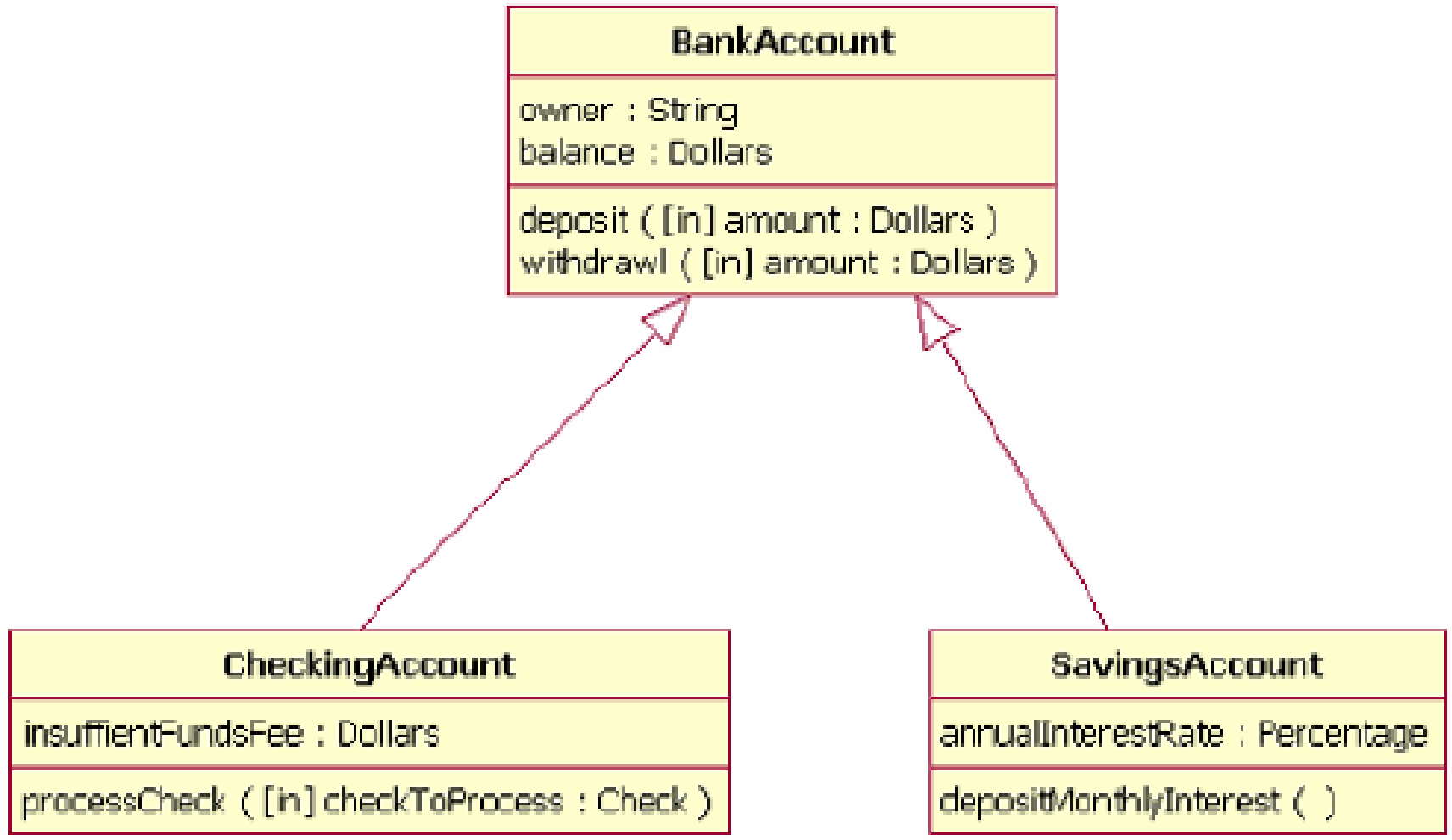
Η σημειογραφία που χρησιμοποιείται είναι ένα βέλος που δείχνει προς την superclass.

*Στο παράδειγμα το **Shape** is a superclass του **Circle** και του **Square**.*



Fundamental Principles of OO Programming

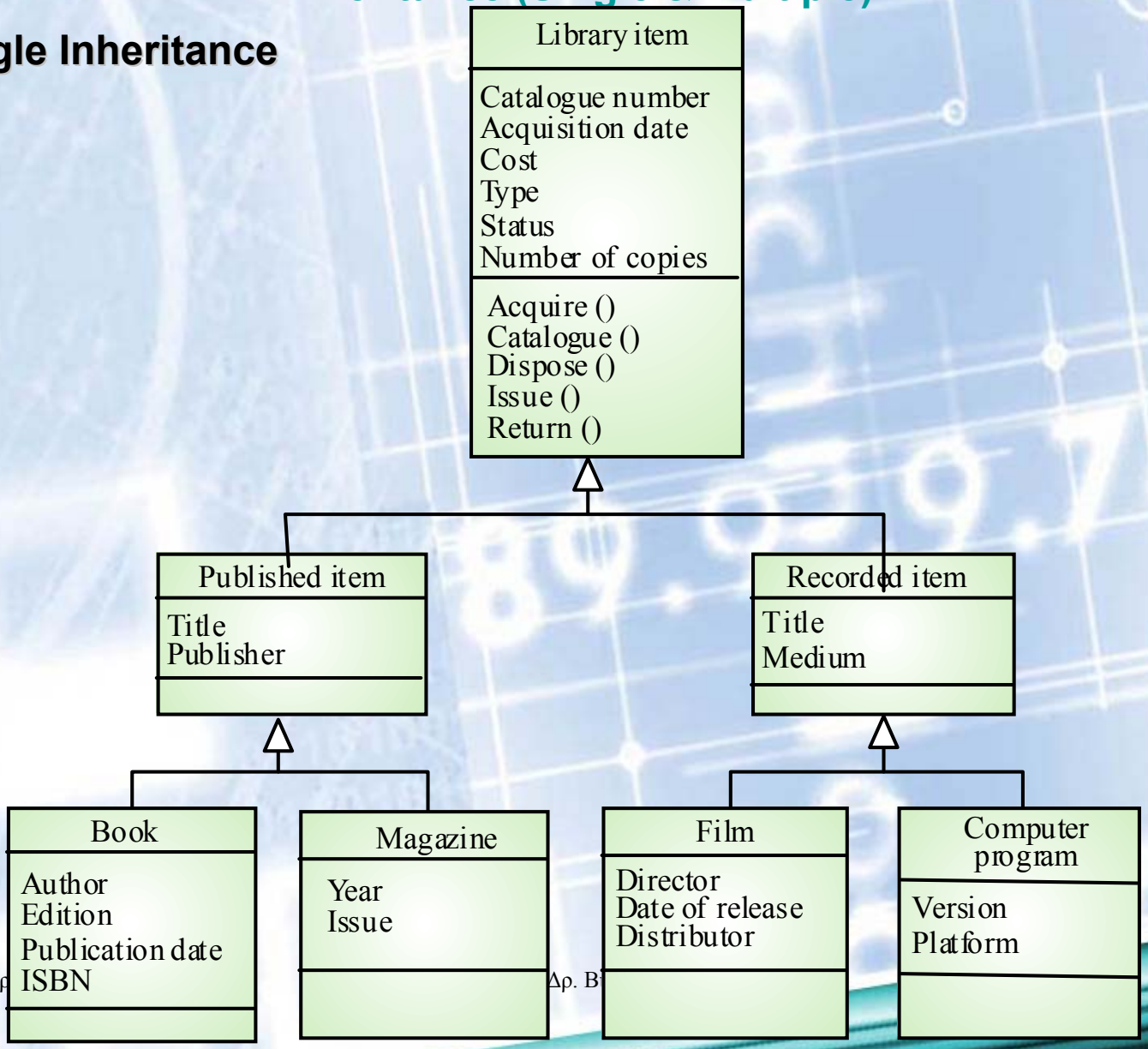
Inheritance



Fundamental Principles of OO Programming

Inheritance (Single & Multiple)

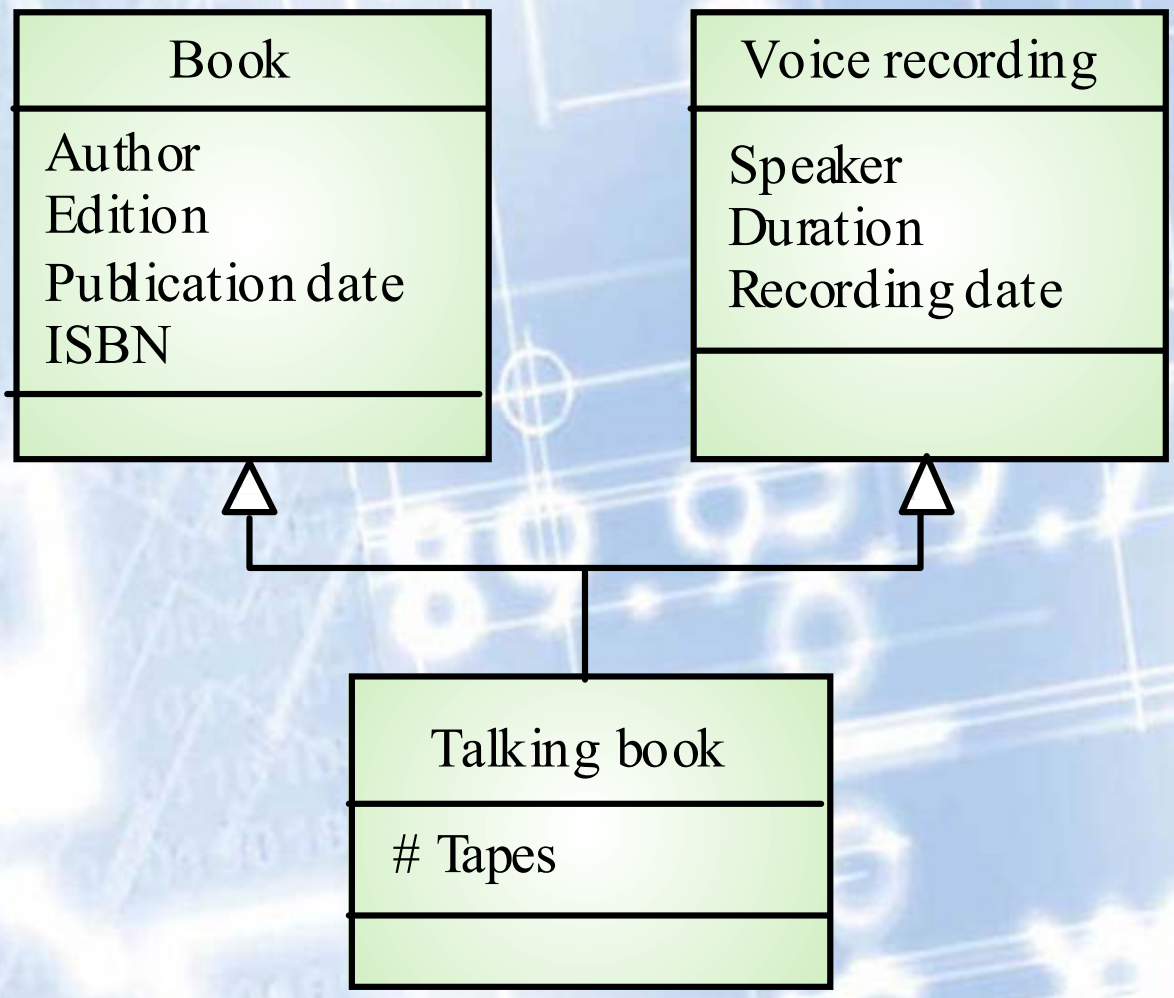
Single Inheritance



Fundamental Principles of OO Programming

Inheritance (Single & Multiple)

Multiple Inheritance



Fundamental Principles of OO Programming

Inheritance

Περισσότερα για την Κληρονομικότητα σε επόμενη διάλεξη με θέμα:

Interfaces and Inheritance