ΔΙΕΘΝΕΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΤΗΣ ΕΛΛΑΔΟΣ

Τμήμα Μηχανικών Πληροφορικής και
Ηλεκτρονικών Συστημάτων

Αντικειμενοστρεφής Προγραμματισμός (Object Oriented Programming)

Βασικές Έννοιες OOP, Κληρονομικότητα, Σύνθεση (*Inheritance - Composition*)

Παναγιώτης Σφέτσος, PhD
http://aetos.it.teithe.gr/~sfetsos/
sfetsos@it.teithe.gr

Βασικές Αρχές και Έννοιες της Αντικειμενοστρεφούς Σχεδίασης και Προγραμματισμού *(1/2)*

- **Εισαγωγική παρουσίαση και μετά εμβάθυνση….**
- **Βασικές Έννοιες Αντικειμενοστρεφούς Σχεδίασης και Προγραμματισμού** *(OOD – OOP)*
- **Ενθυλάκωση** *(Encapsulation)*
- **Πολυμορφισμός** *(Polymorphism)*
- **Κληρονομικότητα** *(Inheritance)*
- **Σύνθεση** *(Composition)*
- **Αφαίρεση** *(Abstraction)*

Βασικές Αρχές και Έννοιες της Αντικειμενοστρεφούς Σχεδίασης και Προγραμματισμού *(2/2)*

Εισαγωγή

Πως **σχεδιάζαμε** και **υλοποιούσαμε** ένα σύστημα στον διαδικαστικό προγραμματισμό και πως γίνεται τώρα, στον OOD – OOP;

Επαναχρησιμοποίηση του κώδικα *(Reusability).*

- *Ο ρόλος της Κληρονομικότητας, Σύνθεσης - Πολυμορφισμού*

Συντηρησιμότητα του κώδικα *(Maintainability)*

- *Διαφορές στον Διαδικαστικό και Αντικειμενοστρεφή Προγραμματισμό*

Επεκτασιμότητα του κώδικα *(Extendabillity)*

- *Ο ρόλος της Ενθυλάκωσης, Κληρονομικότητας, κλπ.*

Ενθυλάκωση *(Encapsulation)*

Ενθυλάκωση δεδομένων *(data encapsulation)* είναι η ιδιότητα των κλάσεων να *'κρύβουν'* τα **ιδιωτικά** τους πεδία οι τιμές των οποίων μπορούν να τροποποιηθούν μόνο μέσω **δημοσίων μεθόδων**.

- Δηλ., **private πεδία – public μέθοδοι**
- Τα private πεδία δεν επιτρέπουν πρόσβαση εκτός κλάσης *(απόκρυψη δεδομένων – data hiding).*

Πλεονεκτήματα:

- Δυνατότητα τροποποίησης του κώδικα
- Ασφάλεια, συντήρηση, επεκτασιμότητα και ευελιξία στην σχεδίαση και υλοποίηση των συστημάτων

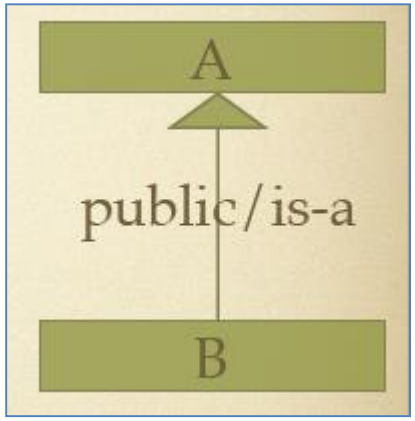
Κληρονομικότητα *(Inheritance)*

Στον ΟΟΡ *(και στην java)* μπορεί από μία ήδη υπάρχουσα κλάση (πρόγονος) να δημιουργηθεί μια νέα κλάση απόγονος που **κληρονομεί όλα τα χαρακτηριστικά και λειτουργίες της υπάρχουσας κλάσης.**

- Ο μηχανισμός αυτός καλείται **κληρονομικότητα**.
- Η υπάρχουσα κλάση ονομάζεται **βασική κλάση** *(base class)*, ή **κλάση γονέας** *(parent class)*, ή **υπερκλάση** *(superclass)*.
- Η νέα κλάση που δημιουργείται και κληρονομεί την υπερκλάση ονομάζεται **υποκλάση** *(subclass),* ή **κλάση παιδί,** ή **παράγωγη κλάση** *(derived class)*.
- Κληρονομικότητα σημαίνει **εξειδίκευση**. Εκτός από τα μέλη που κληρονομεί η υποκλάση μπορεί να έχει επιπλέον δικά της μέλη που την κάνουν πιο ειδική σε σχέση με την υπερκλάση που θεωρείται πιο γενική.
- Στην κληρονομικότητα έχουμε **ιεραρχία κλάσεων** *(class hierarchy),* που διαβάζεται από επάνω προς τα κάτω. Επάνω βρίσκεται η υπερκλάση και κάτω οι υποκλάσεις που εξειδικεύονται προσθέτοντας χαρακτηριστικά και λειτουργίες.

Αρχή της Υποκατάστασης της Liskov

Στην κληρονομικότητα, *'**αντικείμενα του υπερτύπου μπορούν να αντικατασταθούν από αντικείμενα του υποτύπου**, χωρίς να επηρεάσουν την ορθότητα και την συμπεριφορά του προγράμματος'.*



- Όπου μπορεί να χρησιμοποιηθεί ένα αντικείμενο του τύπου Α, μπορεί να χρησιμοποιηθεί ένα αντικείμενο του τύπου Β *(υποκατάσταση)*.

Ιεραρχία Τύπων

Χρησιμοποιούνται για να **παρέχουν διαφορετικές υλοποιήσεις ενός τύπου**. Δηλ., οι υποτύποι παρέχουν διαφορετικές υλοποιήσεις του υπερτύπου.

Πολυμορφισμός *(Polymorphism)*

Πολυμορφισμός είναι η δυνατότητα **εκτέλεσης διαφορετικών λειτουργιών μιας μεθόδου** που είναι κοινή σε διαφορετικές κλάσεις (υπερκλάση και υποκλάση), **ανάλογα με το αντικείμενο της κλάσης που την καλεί.**

- Δηλ., **το ίδιο όνομα μεθόδου προκαλεί την εκτέλεση διαφορετικού κώδικα, ανάλογα με το αντικείμενο που την καλεί**.
- Έτσι θα μπορούσαμε να πούμε, σαν άλλο ορισμό, ότι πολυμορφισμός είναι η ικανότητα ενός αντικειμένου να πάρει πολλές μορφές
- Στον ΟΟΡ αυτό επιτυγχάνεται όταν μια **μεταβλητή αναφοράς υπερκλάσης** χρησιμοποιείται για να αναφερθεί σε ένα αντικείμενο υποκλάσης.

Αφαίρεση *(Abstraction)*

Είναι ο μηχανισμός απόκρυψης των χαρακτηριστικών των αντικειμένων που δεν είναι χρήσιμα στην τρέχουσα εφαρμογή, έτσι ώστε **να μειωθεί η πολυπλοκότητα.**

- Οι αφηρημένες κλάσεις χρησιμοποιούνται για την αναπαράσταση των αντικειμένων στον πραγματικό κόσμο που είναι **αφηρημένα ως έννοιες**, όπως π.χ. το σχήμα, το όχημα κλπ. και για τα οποία **παρέχουν μία βασική υλοποίηση**.
- Δηλαδή, ορίζουμε μια υπερκλάση που καθορίζει την **γενική μορφή** που θα χρησιμοποιείται από όλες τις υποκλάσεις, αφήνοντας σε κάθε υποκλάση την δυνατότητα να συμπληρώσει τις επιμέρους λεπτομέρειες και να υλοποιήσει τις επιμέρους λειτουργίες.
- Άρα η αφηρημένη κλάση παρέχει μια ***κοινή διασύνδεση*** για τον χειρισμό ενός συνόλου κλάσεων.

Επαναχρησιμοποίηση του κώδικα *(Reusability)*

- Στις διαδικαστικές γλώσσες (C, Pascal, κλπ.), ***αντιγραφή τμημάτων* κώδικα και τροποποίηση του** *(ουσιαστικά μικρό αποτέλεσμα)*.

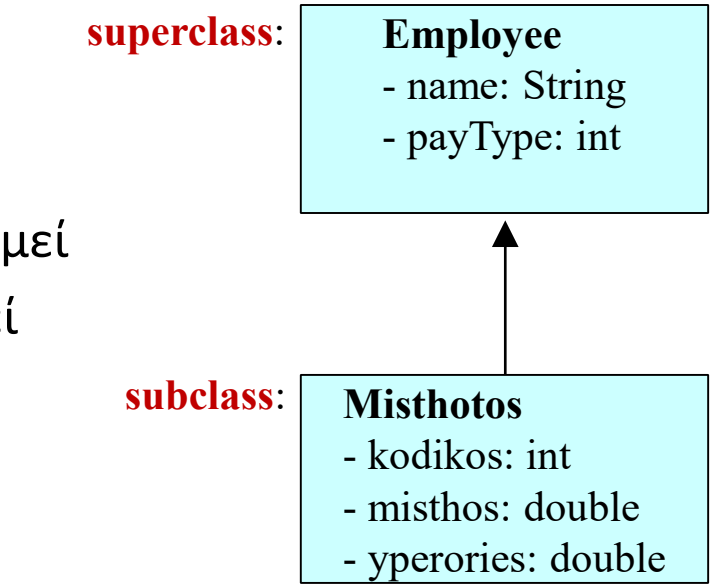
- Στον OOP, η επαναχρησιμοποίηση του κώδικα επιτυγχάνεται με την **δημιουργία νέων κλάσεων**, χρησιμοποιώντας:

Κληρονομικότητα *(Inheritance)*: Δημιουργία μιας υποκλάσης κάποιας ήδη υπάρχουσας κλάσης (υπερκλάσης), που *'κληρονομεί'* τα χαρακτηριστικά και λειτουργίες της υπερκλάσης. Άρα ο κώδικας δεν χρειάζεται να τροποποιηθεί αλλά μόνο να επεκταθεί (**Αρχή της Ανοικτής-Κλειστής Σχεδίασης** *(Open-Closed Principle).*

Σύνθεση *(Composition)*: Δημιουργία ***αντικειμένων κλάσεων*** που ***ήδη υπάρχουν*** μέσα σε μια νέα κλάση. Στην σύνθεση χρησιμοποιείται η λειτουργικότητα του κώδικα.

Κληρονομικότητα *(Inheritance)*

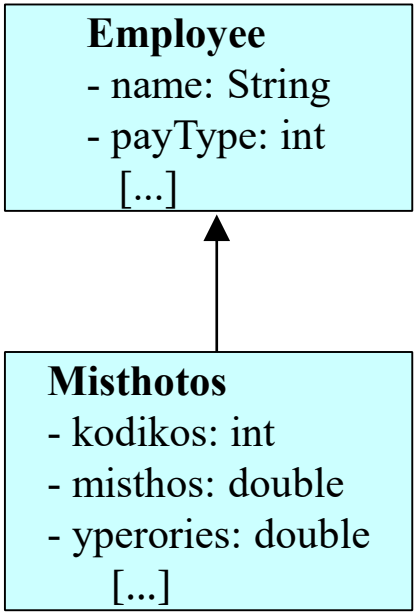
- Η κληρονομικότητα είναι **θεμελιώδης αρχή της Αντικειμενοστρέφειας..**
- Η δημιουργία μιας νέας κλάσης η οποία ονομάζεται **υποκλάση** *(subclass)* ή παράγωγη *(derived)* από μία υπάρχουσα που ονομάζεται **υπερκλάση** *(superclass)* ή βασική *(base class)*. Με τον τρόπο αυτόν επιτυγχάνεται ***επαναχρησιμοποίηση του κώδικα.***
- Η υποκλάση **κληρονομεί** όλα τα **χαρακτηριστικά** και **μεθόδους** της υπερκλάσης.
- Η υποκλάση μπορεί να:
  - **προσθέτει** νέα χαρακτηριστικά (επέκταση)
  - **προσθέτει** νέα λειτουργικότητα (επέκταση)
  - **χρησιμοποιεί** λειτουργικότητα που κληρονομεί
  - να **υπερβεί** λειτουργικότητα που κληρονομεί



Κληρονομικότητα *(Inheritance)*

- Η κληρονομικότητα στη Java δηλώνεται με την δεσμευμένη λέξη **extends**. Αν δεν δηλώνεται κληρονομικότητα, τότε η κλάση *κληρονομεί / 'επεκτείνει'* την κλάση **Object** (θα την δούμε αναλυτικά παρακάτω).

```
public class Employee
{
   private String name;
   private int payType;
   [...]
```



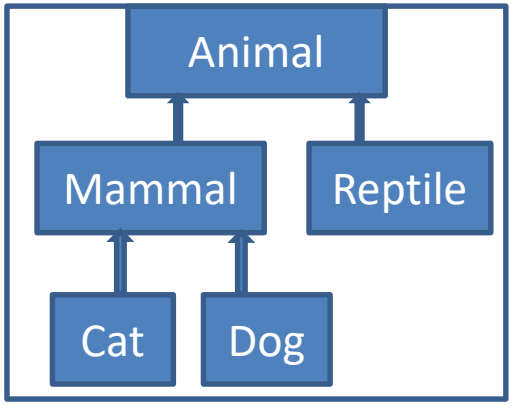

```
public class Misthotos extends Employee
{
   private int kodikos;
   private double misthos;
   private double yperories;
   [...]
```

```
Misthotos emp1 = new Misthotos();
```

Κληρονομικότητα *(Inheritance)*

Ιεραρχία κλάσεων και κανόνες **κληροδότησης:**

- Στη Java κάθε κλάση έχει **μόνο μία υπερκλάση** (δεν επιτρέπεται η πολλαπλή κληρονομικότητα).
- Η κληρονομικότητα δημιουργεί μία **ιεραρχία κλάσεων**
  - οι κλάσεις υψηλότερα στην ιεραρχία είναι πιο *γενικές* και *αφηρημένες.*
  - οι κλάσεις χαμηλότερα στην ιεραρχία είναι πιο *ειδικές* και *συγκεκριμένες*.
  - δεν υπάρχει περιορισμός στο πλήθος των υποκλάσεων της ιεραρχίας και στο βάθος του ιεραρχικού δένδρου.




```
public class Animal { }
public class Mammal extends Animal { }
public class Reptile extends Animal { }
public class Dog extends Mammal { }
public class Cat extends Mammal { }
```

Κληρονομικότητα *(Inheritance)*

- Η υποκλάση κληρονομεί όλα τα **public** μέλη της υπερκλάσης (**όχι τους δομητές**)
- Η υποκλάση έχει **άμεση πρόσβαση** σε όλα τα **public** και **protected** μέλη της υπερκλάσης, αλλά και στα **default** αν βρίσκονται στο ίδιο πακέτο με αυτήν.
- Τα **private** μέλη της κλάσης δεν κληρονομούνται άμεσα, αλλά έχουμε πρόσβαση σε αυτά μέσω των public μεθόδων της υπερκλάσης *(π.χ. setter – getter)*.

Αρχικοποιήσεις - Δομητές της υπερκλάσης

- Η αρχικοποίηση του αντικειμένου της υποκλάσης εξασφαλίζεται με την κατάλληλη ***κλήση του δομητή της υπερκλάσης***. Η κλήση του δομητή της υπερκλάσης γίνεται με την χρήση του όρου **super(.)** Η super πρέπει να είναι η πρώτη εντολή του δομητή της υποκλάσης.

Κληρονομικότητα *(Inheritance)*

- Με την super έχουμε πρόσβαση σε κάθε μεταβλητή μέλος της υπερκλάσης από την υποκλάση.

```
super.<variable>
π.χ. super.size=38;
```

- Αν δεν βάλουμε την super() στον δομητή, τότε ο compiler της java αυτόματα θα καλέσει τον default δομητή της υπερκλάσης, αλλά το σωστό είναι να μην το αφήνουμε στη java.

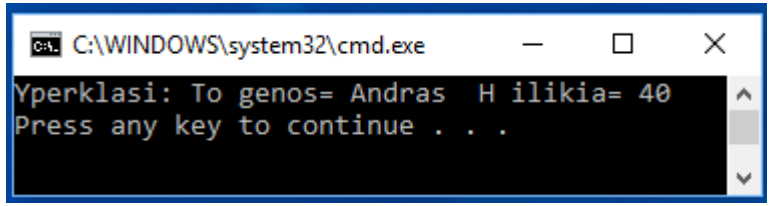
```
public class Employee
{  private String name;
   private int payType;

   public Employee(String n, int pt)
   {  name=n;
      payType=pt; }
         [...]
```

```
public class Misthotos extends Employee
{  private int kodikos;
    private double misthos;
    private double yperories;
 public Misthotos(String n, int pt, int k, double m, double y)
{   super(n, pt);
     kodikos=k;
       [...]
```

Κληρονομικότητα *(Inheritance)*

Παράδειγμα 1ο: Χρήση της **παραμετρικής super()** για να καλέσουμε τον **δομητή της υπερκλάσης**. Το αρχείο αποθηκεύεται με το όνομα: *Subclass.java*

```
class Superclass {
 String gender;
 int age;

 Superclass(String g, int a) {this.gender=g;this.age = a; }

 public void getGenderAndAge(){
  System.out.println("Yperklasi: To genos= "+gender+"H ilikia= "+age);
}}

class Subclass extends Superclass {
   Subclass(String g1, int a1) {
      super(g1, a1); }

public static void main(String argd[]) {
      Subclass s = new Subclass("Andras", 40);
      s.getGenderAndAge();  } }
```



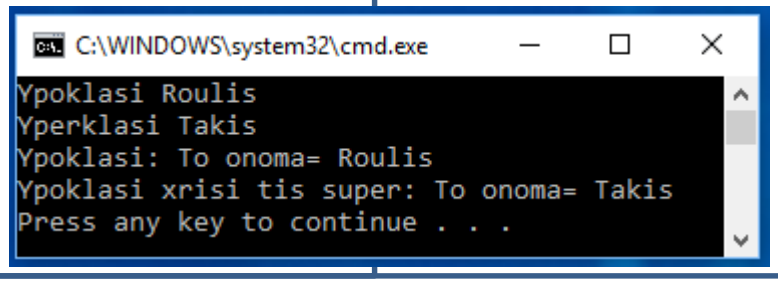
Κληρονομούμε την *λειτουργικότητα* (μέθοδο) της υπερκλάσης

Κληρονομικότητα *(Inheritance)*

Παράδειγμα 2ο: Χρήση της **super** στην υποκλάση για πρόσβαση σε πεδίο της υπερκλάσης. Το αρχείο αποθηκεύεται με το όνομα: *Subclass1.java*

```
class Superclass1 {
 String name = "Takis";

 public void display()
  {System.out.println("Yperklasi "+name);
 }
}

public class Subclass1 extends Superclass1
{
 String name = "Roulis";

 public void display()
  {System.out.println("Ypoklasi "+name);}
```

```
public void A_method() {
 Subclass1 s = new Subclass1();
 s.display();
 super.display();
 System.out.println("Ypoklasi: To
       onoma= "+ s.name);
  System.out.println("Ypoklasi xrisi
    tis super: To onoma= "+
      super.name); }

public static void main(String args[])
 {
      Subclass1 obj = new Subclass1();
      obj.A_method();  } }
```



Κληρονομικότητα *(Inheritance)*

Δημιουργία και καταστροφή αντικειμένων υποκλάσεων και υπερκλάσεων

- Η υποκλάση κληρονομεί μεν τα **public** και **protected** μέλη της υπερκλάσης, αλλά **όχι τους δομητές** *(δεν κληρονομούνται)*.
- Γι' αυτό **η πρώτη εντολή στον δομητή** θα είναι η ***super()*** ή ***super(n1, n2,…)***, δηλ., η super με ή χωρίς παραμέτρους, ανάλογα με τις ανάγκες του προγράμματος. Η super καλεί τον δομητή της υπερκλάσης για την αρχικοποίηση των αντικειμένων.
- Η καταστροφή των αντικειμένων της υποκλάσης και της υπερκλάσης *(διαγραφή από τη μνήμη)* γίνεται με την κλήση της **finalize()** που υπερκαλύπτουμε και γράφουμε στο σώμα της τον κώδικα αποδέσμευσης των πόρων π.χ.

  <αντικείμενο-Α>. finalize() Θα αναλύσουμε παρακάτω την κλάση Object, την finalize() και την υπερκάλυψη/υπέρβαση των μεθόδων.

Κληρονομικότητα *(Inheritance)*

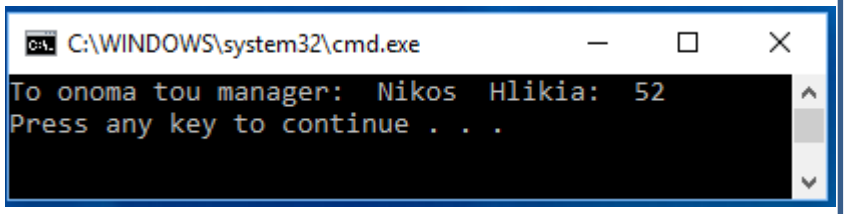
Προσπελασιμότητα ή ορατότητα στην κληρονομικότητα:

	Από μέθοδο στην ίδια κλάση	Από μέθοδο άλλης κλάσης στο ίδιο package	Από μέθοδο υποκλάσης	Από μέθοδο άλλης public κλάσης
private	✦			
package	✦	✦		
protected	✦	✦	✦	
public	✦	✦	✦	✦

Κληρονομικότητα *(Inheritance)*

Παράδειγμα: Χρήση χαρακτηριστικών και μεθόδων της υπερκλάσης. Προσπέλαση σε **private μέλη της υπερκλάσης** με setter()-getter().

```
class Employee1 {
  private String name = "Kanena onoma";
  private int age;
  public void setName(String name) {this.name = name;}
  public String getName() {return name;}
  public void setAge(int age) {this.age=age;}
  public int getAge() {return age;}}
class Manager extends Employee1 { }
class TestEmployee1 {
  public static void main(String[] args) {
    Manager mgr = new Manager();
    mgr.setName("Nikos"); //to onoma tou Mgr, (stin yperklasi)
    String mgrName = mgr.getName(); //to onoma tou Mgr, (apo tin yperklasi)
    mgr.setAge(52); //set Hlikia tou Mgr (stin yperklasi)
    int mgrHlikia=mgr.getAge(); //Hlikia tou Mgr, (apo tin yperklasi)
    System.out.println("To onoma tou manager:  " + mgrName+ "  Hlikia:  "+
                                mgrHlikia);}}
```



Κληρονομικότητα *(Inheritance)*

Σχέσεις: **Είναι-Ένα** *(IS-A)* και **Έχει-Ένα** *(HAS-A)*

Η σχέση *'IS-A'* ισχύει πάντοτε μεταξύ δύο κλάσεων Α και Β όπου η μία **κληρονομεί από την άλλη** είτε άμεσα *(η Β υποκλάση της Α)*, είτε έμμεσα (η Β κληρονομεί από μία κλάση Χ, η οποία έχει κληρονομήσει από την Α).

Π.χ. Cat *'IS-A'* Mammal
Dog *'IS-A'* Mammal

Τα χαρακτηριστικά και οι λειτουργίες της κλάσης Mammal ισχύουν και για τις υποκλάσεις της. Η σχέση *'IS-A'* ισχύει μόνο προς την κατεύθυνση από κάτω προς τα πάνω και όχι αντιστρόφως.

Η σχέση: Έχει-Ένα *'HAS-A'*

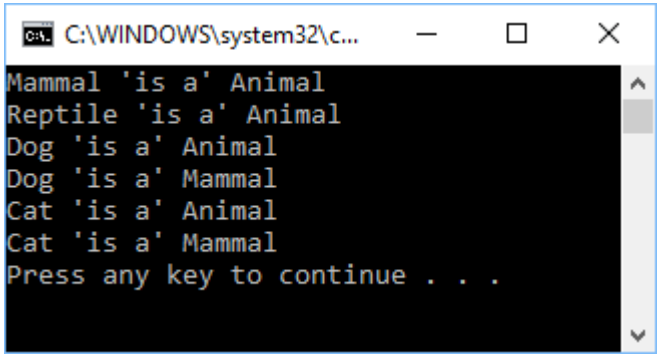
Δηλώνει μια **σχέση σύνθεσης,** δηλ. όταν μία κλάση περιέχει ως ***μεταβλητές*** μέλη ένα ή περισσότερα αντικείμενα άλλων κλάσεων *(σύνθεση ενός μεγαλύτερου και πιο πολύπλοκου αντικειμένου από πολλά μικρότερα).*

Π.χ. Car *'HAS-A'* Wheels
Car *'HAS-A'* Engine
: :

Κληρονομικότητα *(Inheritance)*

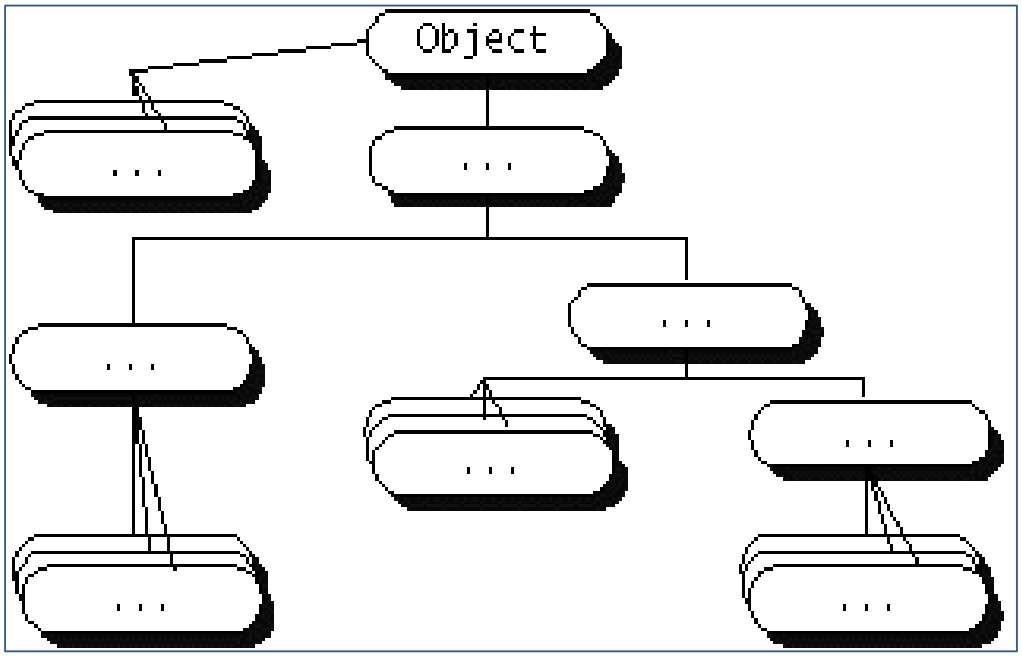
Παράδειγμα : Χρήση του τελεστή **instanceof** για τον έλεγχο της κληρονομικότητας. Η σύνταξη: **if (<*objectReference*> instanceof <*type*>)**

```
class Animal { }
class Mammal extends Animal { }
class Reptile extends Animal { }
class Cat extends Mammal {}

public class Dog extends Mammal {
   public static void main(String args[]) {
      Animal a = new Animal();
      Mammal m = new Mammal();
      Reptile r = new Reptile();
      Dog d = new Dog();
      Cat c = new Cat();
       if (m instanceof Animal)System.out.println("Mammal 'is a' Animal");
       if (r instanceof Animal)System.out.println("Reptile 'is a' Animal");
       if (d instanceof Animal)System.out.println("Dog 'is a' Animal");
       if (d instanceof Mammal)System.out.println("Dog 'is a' Mammal");
       if (c instanceof Animal)System.out.println("Cat 'is a' Animal");
       if (c instanceof Mammal)System.out.println("Cat 'is a' Mammal");
 }}
```



Κληρονομικότητα *(Inheritance)*

Στον OOP, και στην Java, οποιαδήποτε κλάση, ***θα πρέπει να κληρονομεί οπωσδήποτε από κάποια άλλη κλάση***. Όταν γράφουμε μια νέα κλάση αυτή ή θα την βάλουμε να κληρονομήσει από κάποια άλλη κλάση ή ο compiler θα την βάλει να κληρονομήσει την κλάση **java.lang.Object** *(υπερκλάση όλων των κλάσεων)*.



Τις μεθόδους **toString(),** και **finalize()**, που ήδη γνωρίζουμε, κληρονομούν όλες οι κλάσεις.

Οι υπόλοιπες μέθοδοι στην παρακάτω διαφάνεια:

Κληρονομικότητα *(Inheritance)*

clone():	Δημιουργεί και επιστρέφει ένα αντίγραφο του αντικειμένου. *Υπέρβαση/υπερκάλυψη της μεθόδου (θα αναλυθεί παρακάτω).*
equals(Object):	Εξετάζει την ισότητα 2 αντικειμένων και επιστρέφει *true* αν τα αντικείμενα είναι ίδια, διαφορετικά *false*. *(υπέρβαση της μεθόδου)*.
finalize():	Καλείται από 'συλλέκτη σκουπιδιών' για τα ανενεργά αντικείμενα.
getClass():	Επιστρέφει το αντικείμενο που αντιστοιχεί στην τρέχουσα κλάση κατά το runtime *(από την JVM)*.
hashCode():	Επιστρέφει τον κωδικό hash της κλάσης.
notify():	Ενημερώνει συγκεκριμένο νήμα για κάποια αλλαγή κατάστασης.
notifyAll():	Ενημερώνει τα νήματα σε αναμονή για κάποια αλλαγή κατάστασης.
toString():	Επιστρέφει την string απεικόνιση ενός αντικειμένου, *(υπέρβαση της μεθόδου).*
wait():	Θέτει το τρέχον νήμα σε αναμονή.

Κληρονομικότητα *(Inheritance)*

Υπέρβαση/Υπερκάλυψη Μεθόδων *(Method Overriding) (1/2)*

Όταν η υλοποίηση μιας μεθόδου της υπερκλάσης δεν παρέχει την λειτουργικότητα που απαιτεί η υποκλάση, τότε η **μέθοδος** αυτή **μπορεί να υλοποιηθεί με διαφορετικό τρόπο** στην υποκλάση από ότι στη υπερκλάση. **Υπέρβαση** ή **υπερκάλυψη** μεθόδου είναι η διαδικασία όπου μία υποκλάση επανα-υλοποιεί μία μέθοδο που κληρονόμησε από την υπερκλάση.

- Χαρακτηριστικά παραδείγματα η χρήση των μεθόδων της **Object** με υπέρβαση.
- Απαραίτητη στον **Πολυμορφισμό**.

Προσοχή κατά την εκτέλεση:

- Δε πρέπει να αλλάξουμε τον τύπο επιστροφής της μεθόδου ή την υπογραφή της.
- Μπορούμε να αυξήσουμε την πρόσβαση στη μέθοδο (π.χ. από protected σε public).
- Μπορούμε να μειώσουμε ή να διαγράψουμε δηλωμένα exceptions, όχι όμως να προσθέσουμε νέα.

Κληρονομικότητα *(Inheritance)*

Υπέρβαση/Υπερκάλυψη Μεθόδων *(Method Overriding) (2/2)*

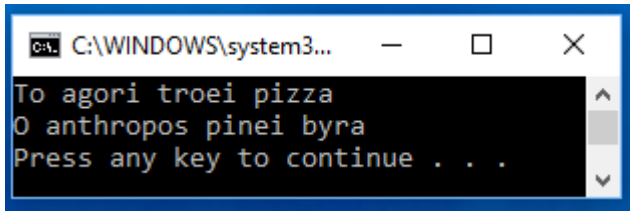
- Γράφουμε στο σώμα της μεθόδου τον κώδικα που υλοποιεί τη νέα λειτουργικότητα. Αν καλέσουμε τη μέθοδο με ***αντικείμενο της υποκλάσης*** θα κληθεί η μέθοδος της υποκλάσης με την νέα λειτουργικότητα, ενώ αν κληθεί με ***αντικείμενο της υπερκλάσης,*** θα κληθεί η δική της μέθοδος με την αρχική λειτουργικότητα. Στον πολυμορφισμό θα χρησιμοποιήσουμε και τις δύο περιπτώσεις.

- Προσοχή στις έννοιες της **υπερφόρτωσης** *(overload)* και της **υπέρβασης** *(override)* των μεθόδων. Είναι δύο διαφορετικές έννοιες και χρήσεις (δες παράδειγμα στις διαφάνειες του επόμενου μαθήματος).

Κληρονομικότητα *(Inheritance)*

Παράδειγμα: **Υπέρβαση μεθόδου**, αλλά και χρήση της super για πρόσβαση σε μέθοδο της υπερκλάσης.

```
class Human{
   public void eat() {System.out.println("O anthropos troei pizza");}
   public void drink() {System.out.println("O anthropos pinei byra");}
}

class Boy1 extends Human{
   public void eat(){System.out.println("To agori troei pizza");}  //ypervasi
   public void drink(){System.out.println("To agori pinei byra");} //ypervasi

 public void A_method() {
   Boy1 b = new Boy1();
   b.eat();          //ypervasi methodou
   super.drink();    // xrisi leitourgias tis yperklasis
 }

public static void main( String args[]) {
      Boy1 obj = new Boy1();
      obj.A_method();
     /* super.drink(); Prosoxi! Lathos klisi apo
      * statiki methodo (main)  */
 }}
```



Κληρονομικότητα *(Άσκηση-1) (1/6)*

Άσκηση Employee *(υλοποίηση κληρονομικότητας)*

Η μισθοδοσία των υπαλλήλων μιας εταιρείας περιλαμβάνει 2 κατηγορίες υπαλλήλων (1) τους Διοικητικούς και (2) τους Τεχνικούς. Για τις 2 κατηγορίες υπάρχουν δύο τύποι μισθοδοσίας (α) **0**=Μισθωτός και (β) **1**=Ωρομίσθιος. Ο βασικός μισθός για τους Διοικητικούς είναι 1200 Ευρώ και για τους Τεχνικούς 800 Ευρώ, ενώ και οι 2 κατηγορίες μπορούν να λάβουν και ένα bonus (Χ)-Ευρώ. Ο μισθός υπολογίζεται με την σχέση: **Μισθός = Βασικός + Bonus**, για τους μισθωτούς και για τους ωρομίσθιους με τη σχέση: **Μισθός = Ώρες Εργασίας * Τιμή Ώρας** (όπου για τον Διοικητικό είναι 12 Ευρώ την ώρα, και για τον Τεχνικό 10 Ευρώ/ώρα).

Κατασκευαστικά:

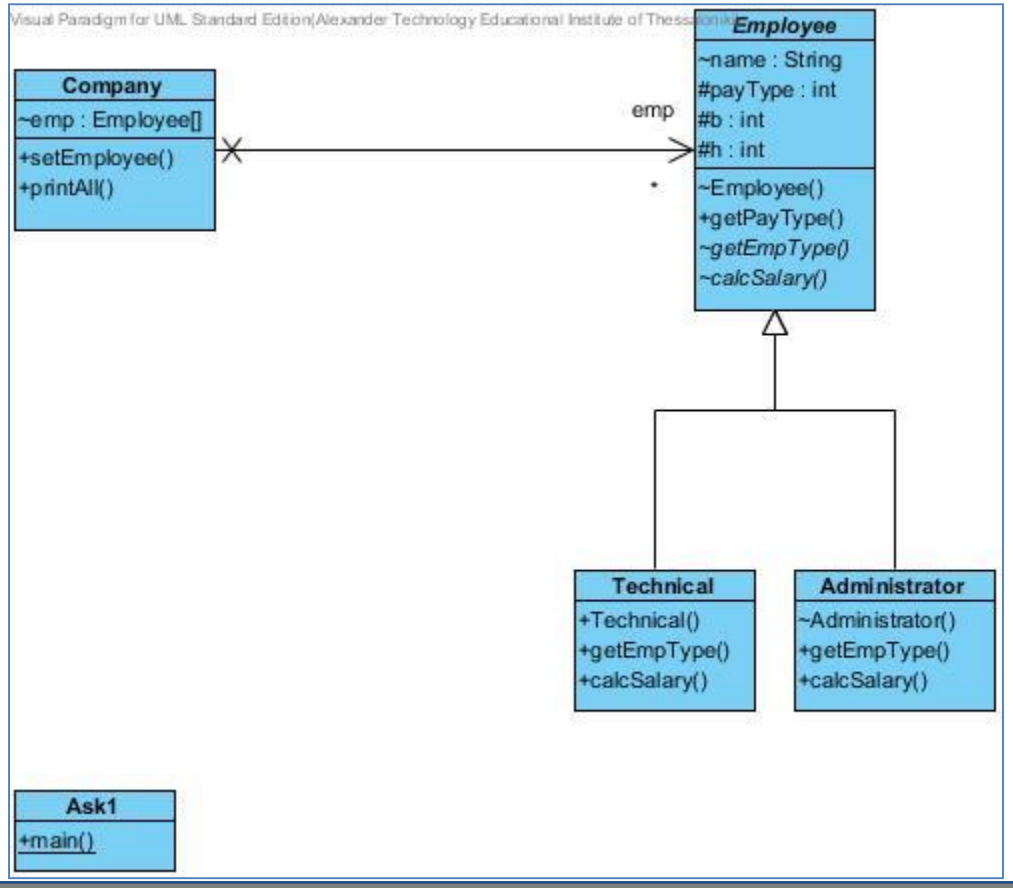
Η ***υπερκλάση:*** **Employee,** θα ορίζει 4 χαρακτηριστικά που είναι κοινά και για τις δύο κατηγορίες (1) Name, (2) Bonus, (3) Hours (για τους ωρομίσθιους) και (4) Τύπος Υπάλληλου (0=Διοικητικός, 1=Τεχνικός) και θα υλοποιεί δύο σημαντικές μεθόδους (α) **getPayType()** που επιστρέφει τα μηνύματα 'Μισθός' ή 'Με την Ώρα' ανάλογα του τύπου μισθοδοσίας, (β) **getEmpType()** που επιστρέφει το μήνυμα "Employee", και (γ) **calcSalary()**, που υπολογίζει τον μισθό του Employee.

Οι ***υποκλάσεις:*** **Administrator** και **Technical** θα κληρονομούν τα 4-χαρακτηριστικά και την μέθοδο getPayType() από την υπερκλάση, ενώ θα υπερβαίνουν *(override)* τις μεθόδους getEmpType() και calcSalary(). Χρησιμοποιήστε μια βοηθητική κλάση την **Company** στην οποία αρχικοποιείται ένας πίνακας 4-αντικειμένων όλων των τύπων και επιπλέον 2 μεθόδους (α) **setEmployee(),** που ειάγει τα αντικείμενα στον πίνακα και (β) **printAll()** που εμφανίζει τα στοιχεία του κάθε υπάλληλου και τον μισθό του.

Κληρονομικότητα *(Άσκηση-1) (2/6)*

UML – Class Diagram *(Employee – Inheritance)*

Το διάγραμμα κλάσεων της UML *(Unified Modeling Language)* απεικονίζει την κληρονομικότητα και τις σχέσεις μεταξύ των κλάσεων στην άσκηση Employee.



Κληρονομικότητα *(Άσκηση-1) (3/6)*

```
class Employee {
  protected String name;
  protected int b;  //bonus
  protected int h;  //hours
  protected int payType; //0=salary, 1=byhour

  Employee(String s, int b_, int h_, int p) {name=s; b=b_; h=h_; payType=p;}
  public String getName() {return name;}

  public String getPayType() {
   String pType;
   if (payType==0) pType="Misthos";
    else pType="Me tin Ora";
       return pType; }

  public String getEmpType() {return "Employee";}
  public void calcSalary() {
       int s=1000;
           System.out.println(" Misthos Employee  = " + s); }}
```

Κληρονομικότητα *(Άσκηση-1) (4/6)*

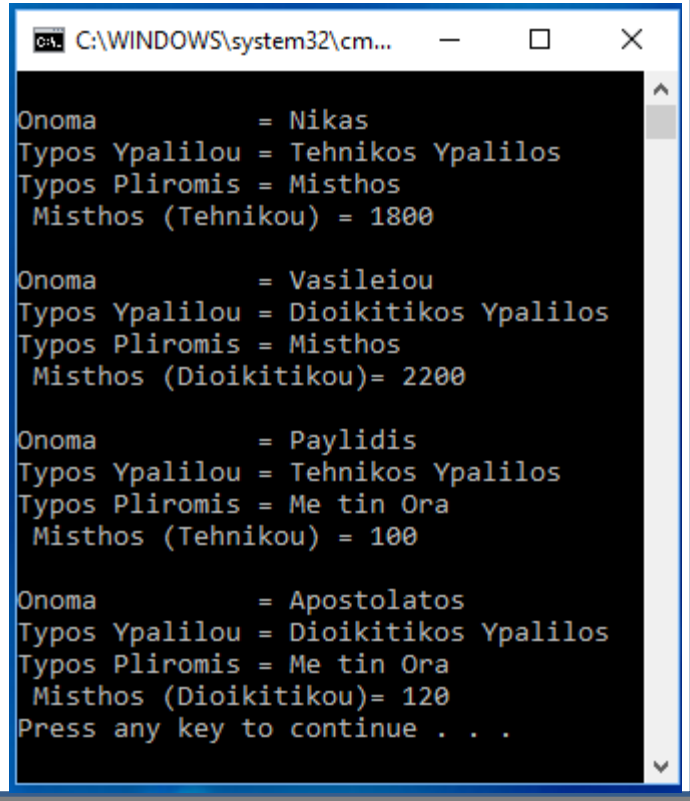
```
class Administrator extends Employee {
  Administrator(String s, int b, int h, int p) {super(s,b,h,p);}
  public String getEmpType() {return "Dioikitikos Ypalilos";}
  public void calcSalary() {
     int s=0;
         if (payType==0) s=1200+b; //vasikos=1200
             else s=(h*12);  //12 Euro per hour
            System.out.println(" Misthos (Dioikitikou) = " + s); }}

class Technical extends Employee {
  Technical(String s, int b, int h, int p) {super(s,b,h,p);}
  public String getEmpType() {return "Tehnikos Ypalilos";}
  public void calcSalary() {
            int s=0;
            if (payType==0) s=800+b; //vasikos=800
               else s=(h*10); //10 Euro per hour
                  System.out.println(" Misthos (Tehnikou)= " + s); } }
```

Κληρονομικότητα *(Άσκηση-1) (5/6)*

```
/* Βοηθητική κλάση ορισμού πίνακα αντικειμένων, γέμισμα του πινακα
 * με τα 4 αντικείμενα setEmployee(), και εμφάνιση των στοιχείων
 * των υπαλλήλων και του μισθού των */

class Company {
  Employee emp[]= new Employee[4];
  public void setEmployee(Employee e, int a) {emp[a]=e;}
  public void printAll() {
        for (int i=0;i<4;i++) {
          System.out.println();
          System.out.println("Onoma          = "+ emp[i].getName());
          System.out.println("Typos Ypalilou = "+ emp[i].getEmpType());
          System.out.println("Typos Pliromis = "+ emp[i].getPayType());
          emp[i].calcSalary(); } } }
```

Κληρονομικότητα *(Άσκηση-1) (6/6)*

```
class TestEmployee {
  public static void main(String[] args) {
  int a;
    Employee e1= new Technical("Nikas",1000,10,0);
    Employee e2=new Administrator("Vasileiou",1000,20,0);
    Employee e3= new Technical("Paylidis",10,10,1);
    Employee e4=new Administrator("Apostolatos",10,10,1);
    Company c = new Company();

    //εισαγωγή των αντικειμένων στον πίνακα
    c.setEmployee(e1,0);
    c.setEmployee(e2,1);
    c.setEmployee(e3,2);
    c.setEmployee(e4,3);
    c.printAll();
} }
```



Άσκηση-1

***Άσκηση υλοποίησης κληρονομικότητας και υπέρβασης μεθόδων:* Τηλέφωνο**

Δημιουργήστε τις υποκλάσεις **StatheroTilefono** και **KinitoTilefono** και την υπερκλάση **Tilefono**. Οι δύο υποκλάσεις θα κληρονομούν από την υπερκλάση χαρακτηριστικά και μεθόδους. Όλες οι κλάσεις που θα δημιουργήσετε να έχουν ένα πλήρη δομητή, τα κατάλληλα πεδία, τις μεθόδους setters() και getters(), toString() και τι μεθόδους: (1) **TyposThlefonou**, (2) **klisiKinitou** και (3) **klisiStatherou** που πραγματοποιούν, ανάλογα με τον τύπο του τηλεφώνου, κλήσεις στο κινητό και σταθερό τηλέφωνο. Η χρέωση είναι 0.10 και 0.20 το λεπτό προς το σταθερό και κινητό αντίστοιχα. Το πρόγραμμα θα υπολογίζει τα επιμέρους κόστη για τις κλήσεις προς κινητό και σταθερό αλλά και το συνολικό κόστος όλων των **N – κλήσεων**.
Στην main δημιουργήστε ένα πίνακα N - αντικειμένων με κινητά και σταθερά και με τις αντίστοιχες κλήσεις και υλοποιήστε τις μεθόδους και τους υπολογισμούς που προαναφέρθηκαν. Τα αποτελέσματα θα εμφανίζονται με τα κατάλληλα μηνύματα.

Σύνθεση *(Composition)*

Σύνθεση *(composition)* κλάσεων έχουμε όταν κατά την δημιουργία μιας νέας κλάσης χρησιμοποιούμε ως μέλη της ***χαρακτηριστικά που είναι αναφορές σε αντικείμενα άλλων κλάσεων.***

Ένα παράδειγμα:

Μια μηχανή αποτελείται από πολλά εξαρτήματα όπως κύλινδροι, βαλβίδες, πιστόνια, μπουζί, κλπ. Η σχέση που συνδέει τα εξαρτήματα με τη μηχανή είναι:

- Η μηχανή **έχει** μπουζί
- Η μηχανή **έχει** κυλίνδρους
- Η μηχανή **έχει** πιστόνια

Δηλαδή - κατά την OOP αναπαράσταση, η μηχανή έχει εξαρτήματα-*χαρακτηριστικά* που το καθένα από αυτά είναι αντικείμενο άλλων κλάσεων. Αυτού του είδους η σχέση ονομάζεται "**HAS-A** ή **whole/part** συσχέτιση" που σημαίνει **σύνθεση** *(composition).*

Σύνθεση *(Composition)*

- Όταν ένα πεδίο της νέας κλάσης είναι αντικείμενο άλλης κλάσης, τότε λέμε ότι η νέα κλάση είναι ένα **σύνθετο αντικείμενο** *(composite)* άλλων αντικειμένων.
- Αν ένα αντικείμενο Β περιέχεται σε ένα αντικείμενο Α, τότε το αντικείμενο-Α είναι υπεύθυνο για την ***δημιουργία και καταστροφή*** του αντικειμένου-Β.
- **Επαναχρησιμοποίηση κώδικα** *(code reuse),* όταν η σχέση μεταξύ δύο ή περισσοτέρων κλάσεων είναι "HAS-A".

Παράδειγμα 1ο:

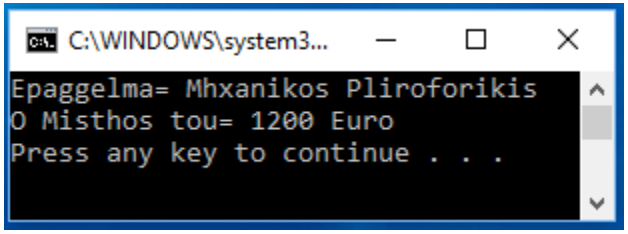
```
class Job {
  private String perigrafi;
  private int misthos;
  private int kodikos;
  public String getPerigrafi(){return perigrafi;}
  public void setPerigrafi(String per) {this.perigrafi = per;}
  public int getMisthos() {return misthos;}
  public void setMisthos(int misthos) {this.misthos = misthos;}
  public int getKodikos() {return kodikos; }
  public void setKodikos(int kodikos) {this.kodikos = kodikos;} }
```

Σύνθεση *(Composition)*

```
class Person {
    //composition has-a relationship
    private Job job;
    public Person(){
        this.job=new Job();
        job.setMisthos(1200);
        job.setPerigrafi("Mhxanikos Pliroforikis");
        }
    public int getMisthos() {return job.getMisthos(); }
    public String getPerigrafi() {return job.getPerigrafi(); }
 }
 class TestPerson {
    public static void main(String[] args) {
        Person person = new Person();
        String p = person.getPerigrafi();
        int misthos = person.getMisthos();
        System.out.println("Epaggelma= " + p);
        System.out.println("O Misthos tou= "+misthos + " Euro"); } }
```

Σύνθεση: Αρχικοποίηση αναφορών

Προσοχή!!
Αναφορά και Δημιουργία του αντικειμένου Job. Η απόδοση τιμών γίνεται ή στο σημείο ορισμού ή στον δομητή της κλάσης ή πριν την χρήση τους.



Σύνθεση *(Composition)*

Παράδειγμα 2ο:

Μία κλάση Employee θα λάβει στοιχεία από άλλες κλάσεις τις: (1) **Name:** που ορίζει το όνομα του υπάλληλου, και (2) **Address:** που ορίζει την διεύθυνση του υπάλληλου.

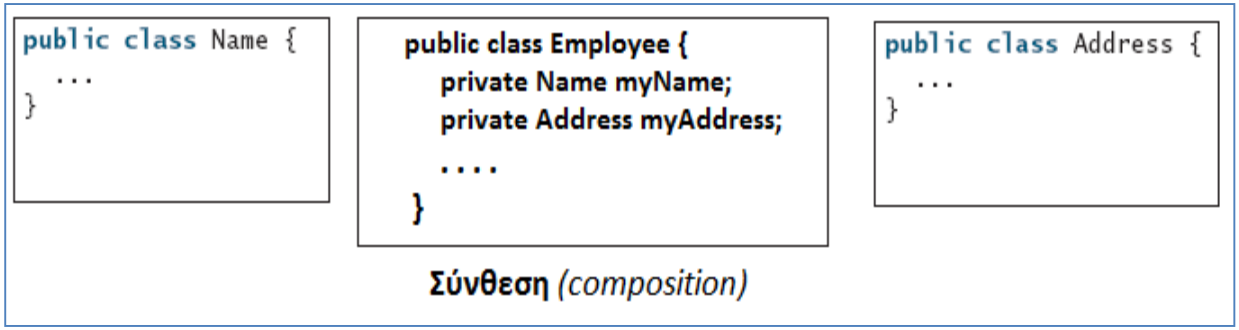
Η κλάση **Employee:** που ορίζει τα στοιχεία του υπάλληλου, χρησιμοποιώντας όμως σαν μεταβλητές ***αντικείμενα*** του των τύπων Name και Address.

Η **TestEmployee:** η οποία ορίζει αντικείμενα του τύπου Name, Address και Employee και που εμφανίζει όλα τα στοιχεία του Υπάλληλου.

```
class Name {
  String firstname;
  String lastname;
  public Name(String newFirstname, String newLastname) {
    firstname = newFirstname;
    lastname = newLastname;}
  public String getFirstname() {return firstname;}
  public String getLastname() {return lastname;}
  public String getFirstLast() {return firstname + " " + lastname;}
}
```

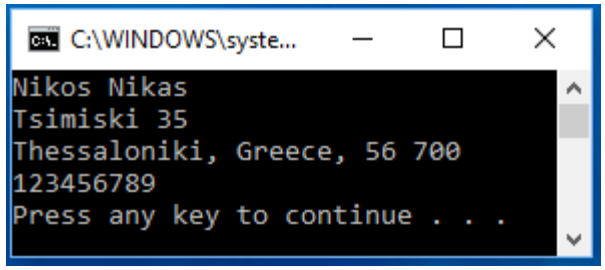
Σύνθεση *(Composition)*

```
class Address {
 String street;
 String city;
 String state;
 String zip;
 public Address(String newStreet, String newCity, String newState,
      String newZip) {
  street = newStreet;
  city = newCity;
  state = newState;
  zip = newZip;
 }
 public String getStreet(){return street; }
 public String getCity(){return city;}
 public String getState(){return state;}
 public String getZip(){return zip;}
 public String getFullAddress() {
    return street + "\n" + city + ", " + state + ", " + zip; }
}
```



Σύνθεση *(Composition)*

```
class Employee {
 private Name myName;          //antikeimeno typou Name
 private Address myAddress;    //antikeimeno typou Address
 private String AFM;
 public Employee(Name n, Address a, String newAFM) {
   myName = n;
   myAddress = a;
   AFM = newAFM; }
 public Name getName() {return myName; }
 public Address getAddress() {return myAddress;}
 public String getAFM() {return AFM; } }
class TestEmployeeComposition {
 public static void main(String[] args) {
  Name eponymia = new Name("Nikos", "Nikas");
  Address dieythinsi = new Address("Tsimiski 35", "Thessaloniki", "Greece",
                                   "56 700");
  String AFM = "123456789";
  Employee theEmployee = new Employee(eponymia, dieythinsi, AFM);
  System.out.println(theEmployee.getName().getFirstLast() + "\n" +
  theEmployee.getAddress().getFullAddress()+"\n"+theEmployee.getAFM());}}
```



Σύνθεση *(Composition)*

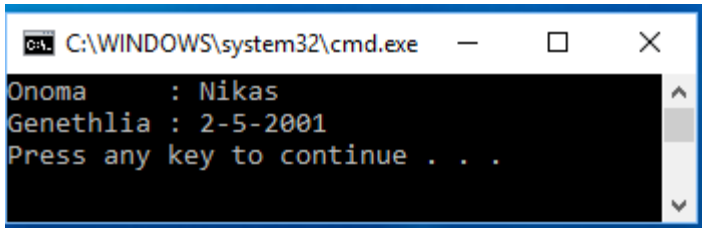
Παράδειγμα 3ο:

Η κλάση **Person** περιέχει σαν χαρακτηριστικό μια αναφορά σε αντικείμενο της κλάσης **Birthday.**

```
class Person {
 private double salary;
 private String name;
 private Birthday bday; //αναφορά σε αντικείμενο της Birthday
 public Person(int d,int m,int y,String name){
  bday=new Birthday(d, m, y); /αρχικοποίηση του αντικ. στον δομητή
  this.name=name;
 }
 public double getSalary() {return salary;}
 public String getName() {return name;}
 public Birthday getBday() {return bday;}//λήψη της Birthday
}
```

Σύνθεση *(Composition)*

```
class Birthday{
 int day, month,year;
 public Birthday(int d,int m,int y){
   day=d;
   month=m;
   year=y; }
 public String toString(){return String.format("%s-%s-%s",
       day,month,year); } }

class CompositionTst1 {
  public static void main(String[] args) {
   Person person=new Person(2, 5, 2001, "Nikas");
   System.out.println("Onoma     : "+person.getName());
   System.out.println("Genethlia : "+person.getBday());
  }
}
```



Composition and Inheritance *(1/3)*

Παράδειγμα

Στο παρακάτω παράδειγμα θα κάνουμε χρήση της σύνθεσης μαζί με την κληρονομικότητα, κατάσταση συνηθισμένη στον OOP. Η Ferrari κληρονομεί από την **Car** *(is-a Car),* αλλά χρησιμοποιεί και την μέθοδο **start()** της κλάσης **Engine** *(has-a Engine)*. Στην **InheritanceAndComposition** *(main),* ορίζουμε ένα αντικείμενο του τύπου Ferrari, και ενώ η κλάση Ferrari δεν έχει τις μεθόδους **setColor(), setMaxSpeed(), carInfo(),** ωστόσο τις χρησιμοποιεί γιατί τις κληρονομεί από την κλάση Car.

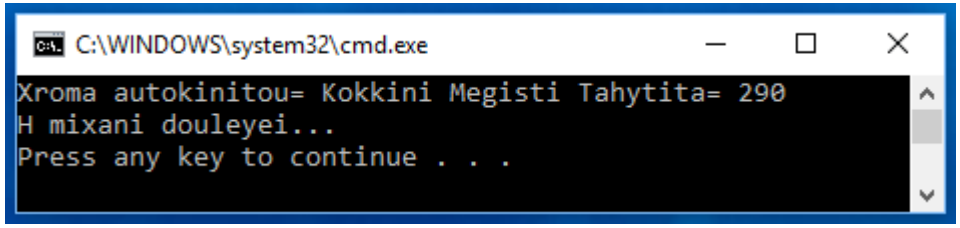
```
class Car {
  private String color;
  private int maxSpeed;
  public void carInfo(){
     System.out.println("Xroma autokinitou= "+color + " Megisti
                  Tahytita= " + maxSpeed); }
  public void setColor(String color){this.color = color;}
  public void setMaxSpeed(int maxSpeed) {this.maxSpeed = maxSpeed;}
}
```

Composition and Inheritance (2/3)

```
class Ferrari extends Car{
   /*H klasi Ferrari klironomei oles tis
   * methodous tis Car (ektos apo tis final kai static)*/

   public void FerrariDemo(){
        Engine FerrariEngine = new Engine();
        FerrariEngine.start(); }
}
class Engine {
   public void start(){
        System.out.println("H mixani douleyei...");
   }
   public void stop(){
        System.out.println("H mixani stamatise..."); }
}
```

Composition and Inheritance (3/3)

```
public class InheritanceAndComposition {
   public static void main(String[] args) {
        Ferrari myFerrari = new Ferrari();
        myFerrari.setColor("Kokkini");
        myFerrari.setMaxSpeed(290);
        myFerrari.carInfo();
        myFerrari.FerrariDemo();
   }
}
```



Κληρονομικότητα ή Σύνθεση ?

- Στη κληρονομικότητα κληρονομούμε και την **υλοποίηση** και τη **διασύνδεση** της υπερκλάσης.
- Δηλαδή, η κληρονομιά της διασύνδεσης εξασφαλίζει ότι η **υποκλάση λαμβάνει όλα -και τα ίδια- μηνύματα με την υπερκλάση** *(χρήση αντικειμένου της υποκλάσης αντί της υπερκλάσης).*
- Στη σύνθεση **δεν βλέπουμε την διασύνδεση** των κλάσεων των ενσωματωμένων αντικειμένων *(private μέλη των κλάσεων).*
- Άρα, **ευκολότερη η αλλαγή κλάσης που εκτελεί σύνθεση** από ότι κληρονομικότητα. Η αλλαγή στην υπερκλάση επηρεάζει την ιεραρχική κληρονομικότητα των υποκλάσεων.

Κληρονομικότητα ή Σύνθεση ?

- Στη σύνθεση τα αντικείμενα μέλη της νέας κλάσης είναι συνήθως private, οπότε τα **μεταβάλουμε εύκολα** χωρίς να επηρεάσουμε τον υπόλοιπο κώδικα. Επίσης, μπορούμε να **αλλάξουμε τα αντικείμενα μέλη** κατά την **εκτέλεση του προγράμματος**, αλλάζοντας **δυναμικά** τη συμπεριφορά του προγράμματος. Στην κληρονομικότητα δεν έχουμε αυτή την δυνατότητα *(περιορισμοί του compiler).*
- Στην κληρονομικότητα μια μέθοδος της υποκλάσης, με την ίδια υπογραφή με την υπερκλάση, **δεν επιτρέπεται να αλλάξει τον επιστρεφόμενο τύπο της μεθόδου**. Στην σύνθεση υλοποιούμε τις δικές μας μεθόδους, αποφεύγοντας αυτό τον περιορισμό.
- Η σύνθεση είναι πιο **ευέλικτη** και **εύκολα υλοποιήσιμη** και πρέπει να προτιμάται.