Φροντιστηριο JAVA

Οντοκεντρικός Προγραμματισμός

Τι θα συζητήσουμε σήμερα

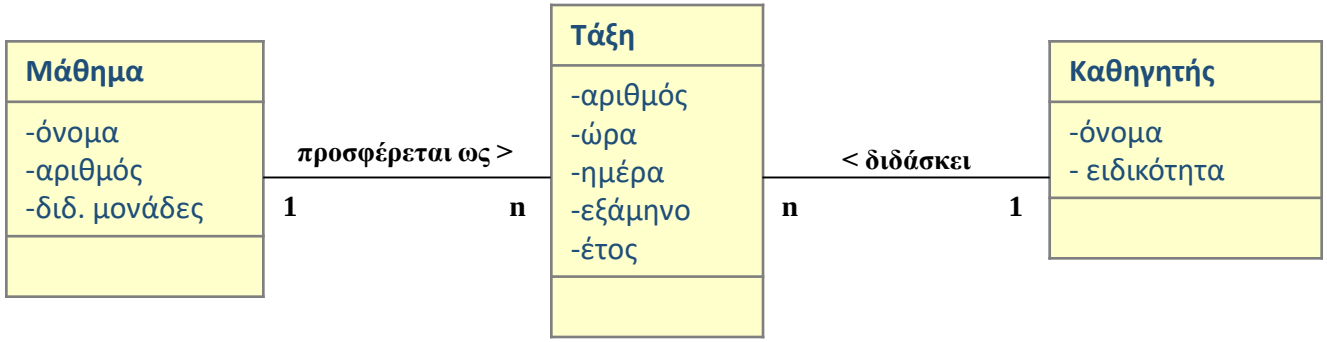
- Πώς υλοποιούμε συσχετίσεις μεταξύ κλάσεων
  - απλές και πολλαπλές συσχετίσεις
  - κληρονομικότητα
- Static, final
- Overloading – Overriding – Hiding
- Παραδείγματα κώδικα

Συσχετίσεις μεταξύ κλάσεων

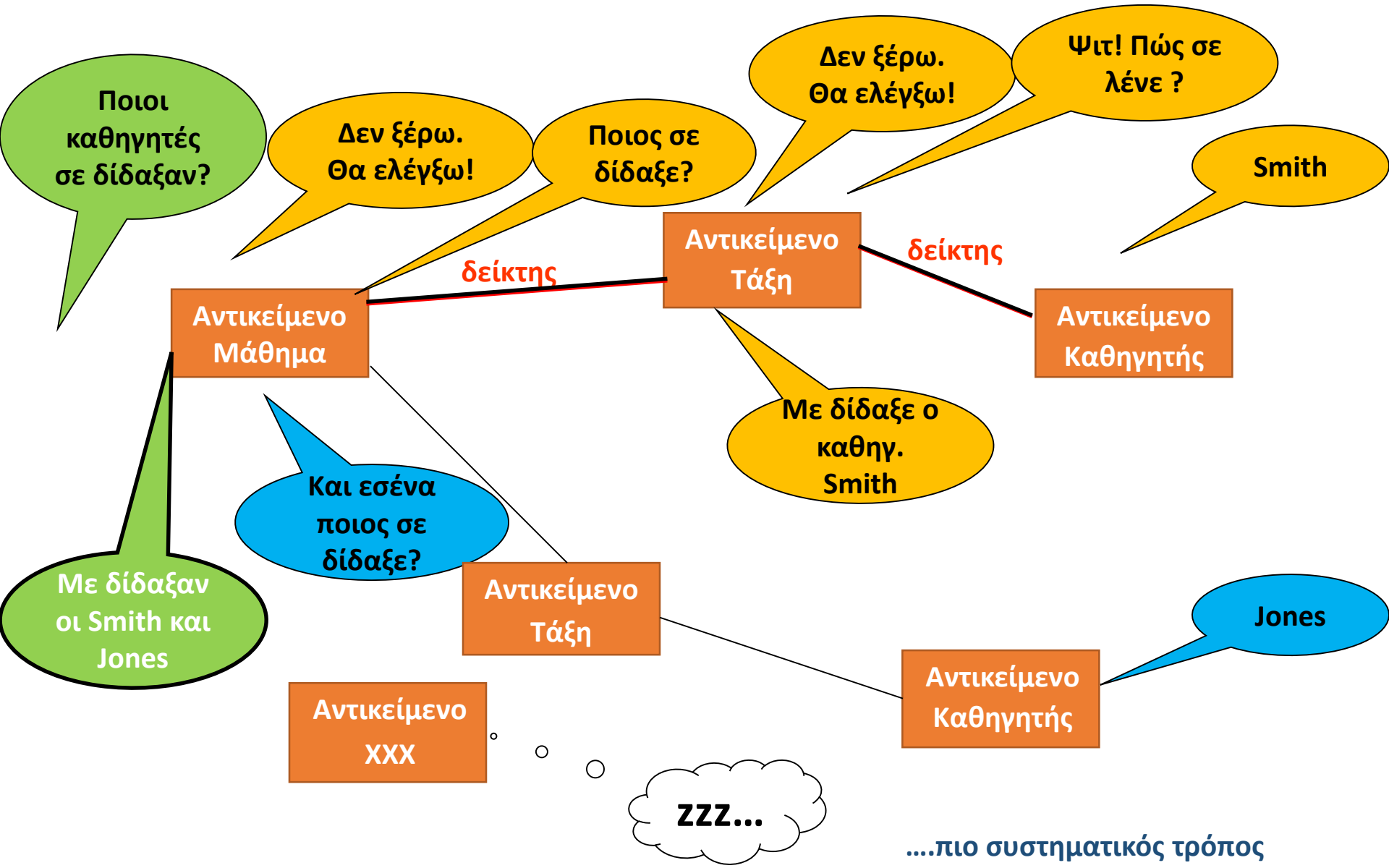
- Οι συσχετίσεις είναι η «**κόλλα**» ενός αντικειμενοστρεφούς συστήματος.
- Παρέχουν την υποδομή για την **επικοινωνία** μεταξύ αντικειμένων
- Υλοποιούνται ως **αναφορές** από μία κλάση προς μία άλλη

Ροή της πληροφορίας

- Κάθε γραμμή συσχέτισης σε ένα διάγραμμα, θα πρέπει να θεωρείται ως ένας **νοητικός 'αγωγός'** δια μέσου του οποίου μπορεί να ρέει η πληροφορία μεταξύ αντικειμένων.

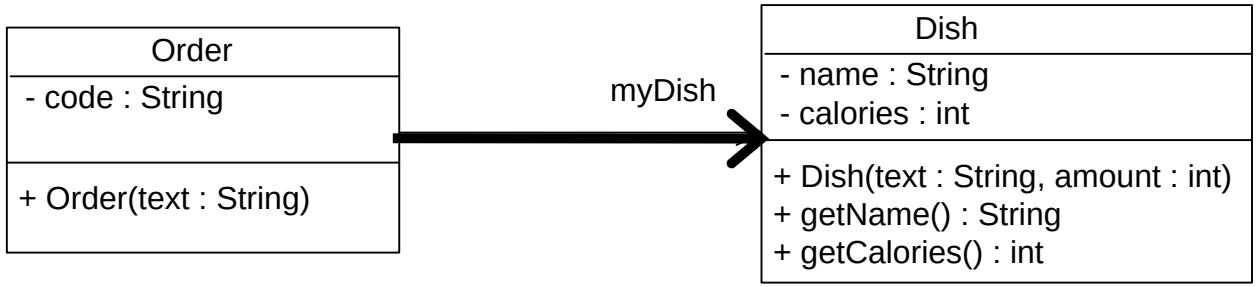


- *Έστω ότι κάποιος χρήστης επιθυμεί να αποκτήσει μία λίστα όλων των καθηγητών που έχουν διδάξει το μάθημα 'Εισαγωγή στον Αντικειμενοστρεφή Προγραμματισμό'.*
- *Επειδή κάθε αντικείμενο τύπου **Μάθημα** διατηρεί δείκτες προς όλες τις **Τάξεις**, τωρινές ή παρελθοντικές, το αντικείμενο **Μάθημα** μπορεί να 'ρωτήσει' τις συσχετιζόμενες **Τάξεις** το όνομα του **Καθηγητή** που δίδαξε ή διδάσκει την αντίστοιχη **Τάξη**.*



....πιο συστηματικός τρόπος ανάλυσης τέτοιας 'συνομιλίας' είναι τα διαγράμματα ακολουθίας

Συσχετίσεις μεταξύ κλάσεων

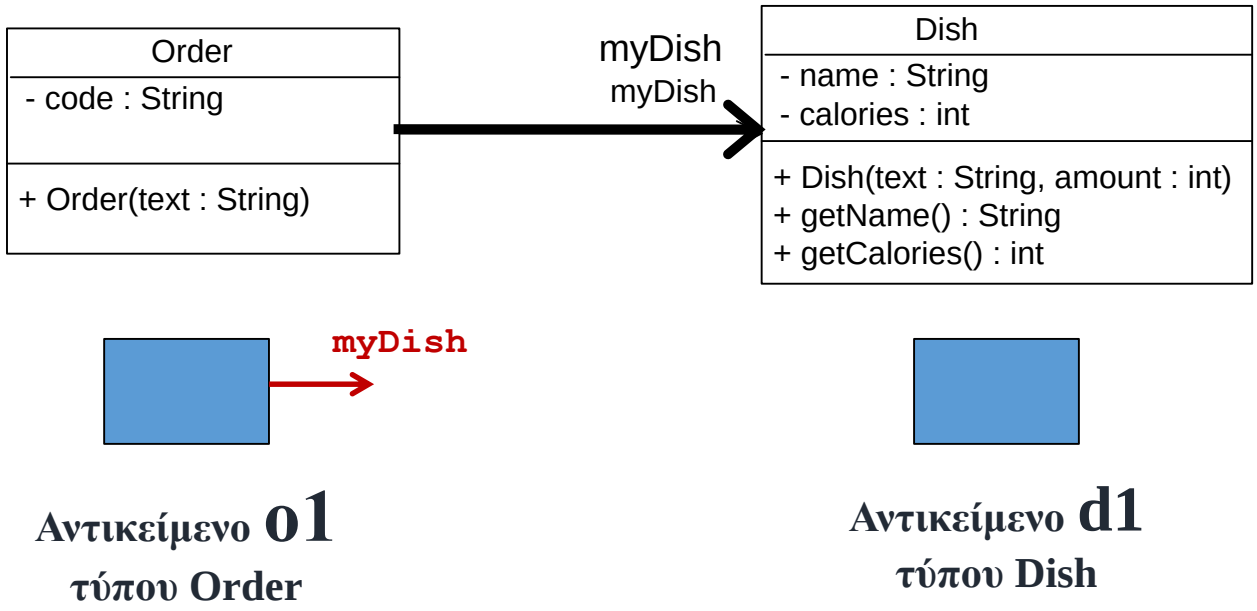



```
public class Order {

  private Dish myDish;    //αναφορά προς την κλάση Dish
                          //υλοποιεί τη συσχέτιση

  private String code;


  public Order(String text) {
     code = text;
  }
}
```

Συσχετίσεις μεταξύ κλάσεων

- Μία συσχέτιση παρέχει την «υποδομή» ώστε αντικείμενα δύο κλάσεων να μπορούν να επικοινωνήσουν (να ανταλλάξουν μηνύματα)
- Κατά τη διάρκεια της εκτέλεσης δημιουργούνται συνδέσεις (links) μεταξύ αντικειμένων που μπορούν να τροποποιηθούν ή να καταστραφούν
- Εάν σε μία αναφορά (που υλοποιεί μία συσχέτιση) δεν δοθεί τιμή κατά τη διάρκεια εκτέλεσης, οποιαδήποτε κλήση μεθόδου μέσω αυτής της αναφοράς είναι άνευ νοήματος (πρόκειται για pointer που δεν «δείχνει» πουθενά)
  - Οδηγεί σε σφάλμα Null Pointer Exception

Συσχετίσεις μεταξύ κλάσεων - Συνδέσεις



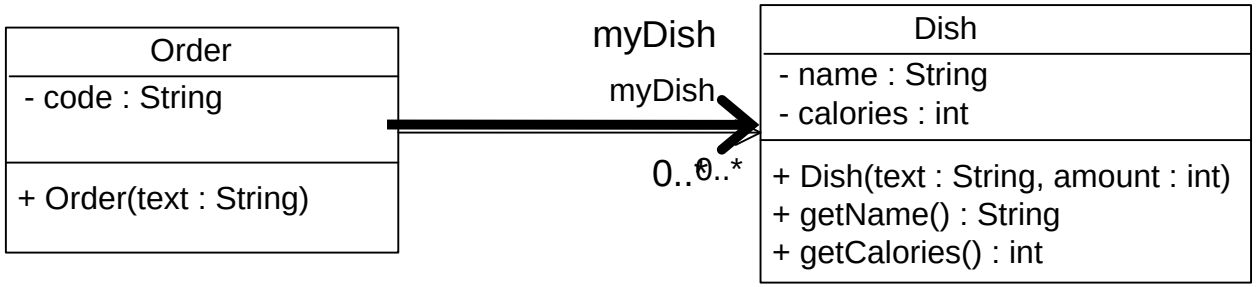
- "Συνδέουμε" το αντικείμενο o1 με το αντικείμενο d1, δίνοντας στην ιδιότητα myDish του αντικειμένου o1 την τιμή d1
- Ενδεχομένως να χρειαστεί κατάλληλη μέθοδος set( ) στην κλάση Order

Συσχετίσεις μεταξύ κλάσεων – Συνδέσεις (associations)


```
public class Order {
  private Dish myDish;   //αναφορά προς την κλάση Dish

  public void setDish(Dish aDish) {
     myDish = aDish;
  }
}
```

```
public static void main(String[] args) {
    Order o1 = new Order("XY3542");
    Dish d1 = new Dish("Arakas");
    o1.setDish(d1);  //εδώ δημιουργείται η σύνδεση
                     //το αντικείμενο o1 μπορεί πλέον
                     //να αποστείλει μηνύματα στο d1
}
```

Συσχετίσεις με πολλαπλότητα



- Η πολλαπλότητα “ 0..* ” στο άκρο της κλάσης Dish, υποδηλώνει ότι ένα αντικείμενο τύπου Order μπορεί να συσχετιστεί με πολλά αντικείμενα τύπου Dish
- Υλοποιείται ως μία δομή δεδομένων στην κλάση Order που περιλαμβάνει αναφορές προς αντικείμενα τύπου Dish
- Η πιο ευρέως διαδεδομένη και εύκολη στη χρήση δομή δεδομένων της Java είναι η ArrayList

ArrayList

- Για να χρησιμοποιηθεί πρέπει πρώτα να συμπεριληφθεί το πακέτο (κατάλογος) στο οποίο είναι δηλωμένη

  import java.util.*; (συμπερίληψη όλων των κλάσεων του πακέτου)

- Αποτελεί ουσιαστικά έναν **δυναμικό πίνακα** που μπορεί να φιλοξενήσει στοιχεία οποιουδήποτε τύπου
- Δημιουργία αντικειμένου ArrayList

```
ArrayList dishes = new ArrayList();
```

- Από την Java 1.5 μπορούμε να δηλώσουμε τον τύπο των στοιχείων που φιλοξενούνται στη δομή (π.χ. αντικείμενα Dish) ως εξής:

```
ArrayList<Dish> dishes = new ArrayList<Dish>();
```

ArrayList

- Εισαγωγή στοιχείου στη δομή καλώντας τη μέθοδο add():

```
Dish d1 = new Dish("Steak");
dishes.add(d1);
```

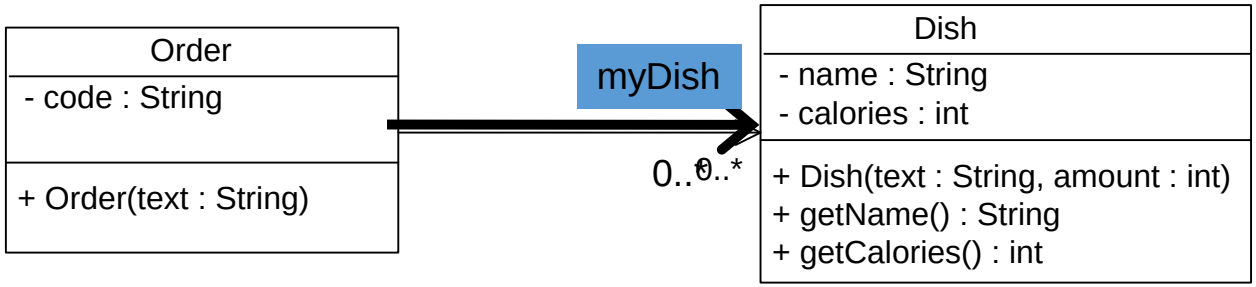
- Λήψη του στοιχείου που βρίσκεται στη θέση i με τη μέθοδο get():

```
Dish d = dishes.get(i);
```

- Διαγραφή του στοιχείου d1 από τη δομή:

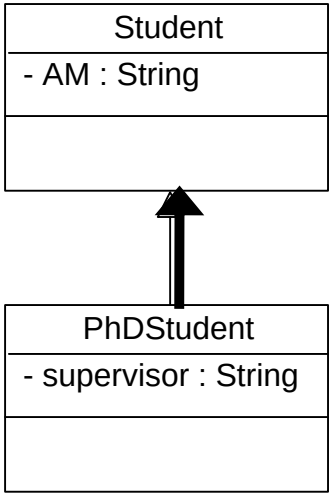
```
dishes.remove(d1);
```

Συσχετίσεις με πολλαπλότητα



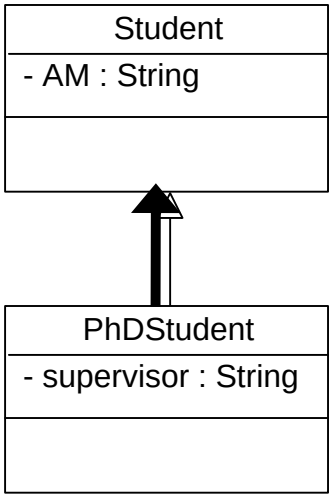

```
public class Order {
  private ArrayList<Dish> myDishes =
                                  new ArrayList<Dish>();

  public void addDish(Dish aDish) {
     myDishes.add(aDish);
  }

  public Dish getDish(int i) {
     myDishes.get(i);
  }
}
```

Κληρονομικότητα



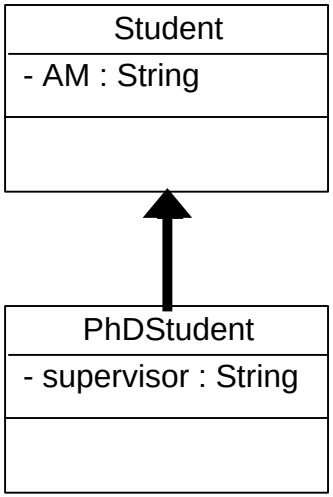

```
public class PhDStudent extends Student {

    private String supervisor;

}
```

Κληρονομικότητα



- Η υποκλάση (PhDStudent) κληρονομεί τις ιδιότητες και τις μεθόδους της υπερκλάσης (Student)
- Η υποκλάση μπορεί να δηλώσει επιπλέον μεθόδους ή ιδιότητες
- Η υποκλάση μπορεί να επαναορίσει (επικαλύψει) μεθόδους που έχουν οριστεί στην υπερκλάση

Αρχή της Υποκατάστασης

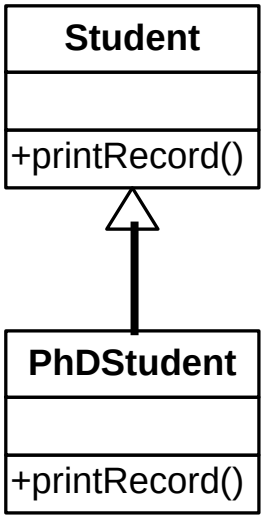


Όπου μπορεί να χρησιμοποιηθεί ένα αντικείμενο του τύπου Student, μπορεί να χρησιμοποιηθεί ένα αντικείμενο του τύπου PhDStudent (υποκατάσταση)

Σε μία αναφορά προς την υπερκλάση είναι απολύτως έγκυρο να αναθέσουμε ως τιμή ένα αντικείμενο οποιασδήποτε υποκλάσης

```
...
   Student p;

   p = new Student();      //επιτρεπτό

   p = new PhDStudent();   //επιτρεπτό
```

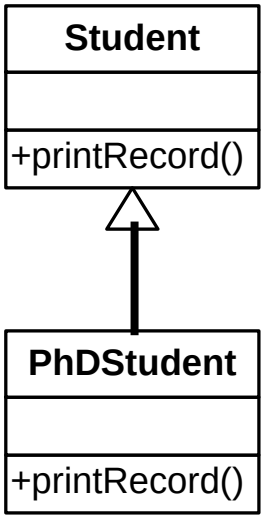
Πολυμορφτισμός



Το γεγονός ότι η μέθοδος printRecord() σημειώνεται και στην κλάση PhDStudent
υποδηλώνει ότι η μέθοδος επαναορίζεται στην υποκλάση

```
. . .
   Student p;

   p = new PhDStudent();

   p.printRecord(); //παρόλο που η αναφορά p είναι
                    //δηλωμένη ως αναφορά τύπου Student
                    //η μέθοδος που καλείται είναι η
                    //printRecord() της PhDStudent
```

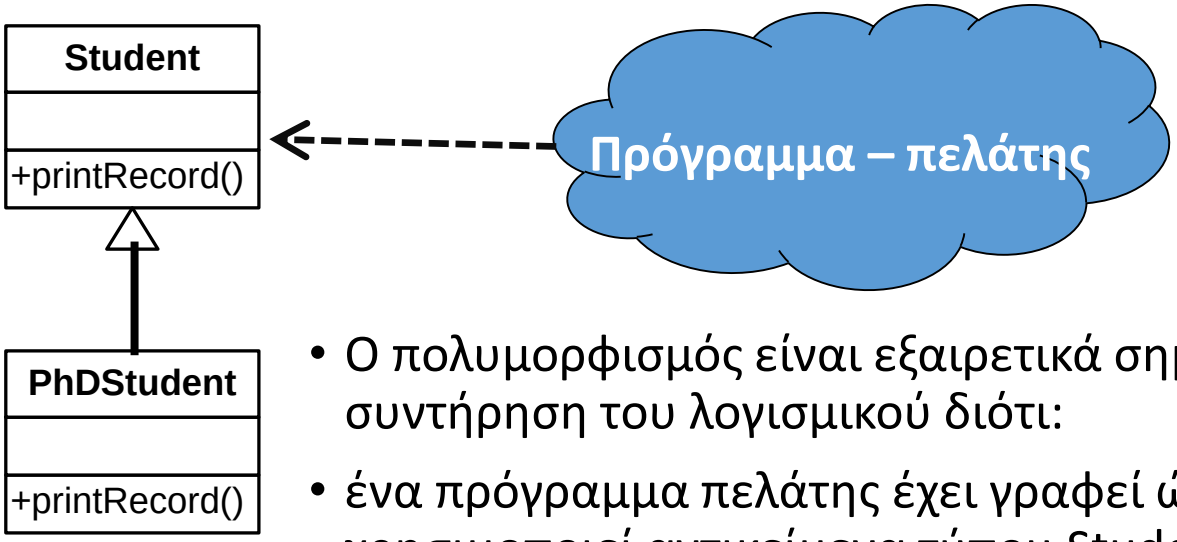
Πολυμορφτισμός



Το γεγονός ότι η μέθοδος printRecord() σημειώνεται και στην κλάση PhDStudent
υποδηλώνει ότι η μέθοδος επαναορίζεται στην υποκλάση

```
. . .
   Student p;

   p = new PhDStudent();

   p.printRecord();//η Java αναμένει μέχρι την εκτέλεση
                   //του προγράμματος για να δεί που
                   //"δείχνει" η αναφορά p
                // = δυναμική ή καθυστερημένη διασύνδεση
                // = ΠΟΛΥΜΟΡΦΙΣΜΟΣ
```

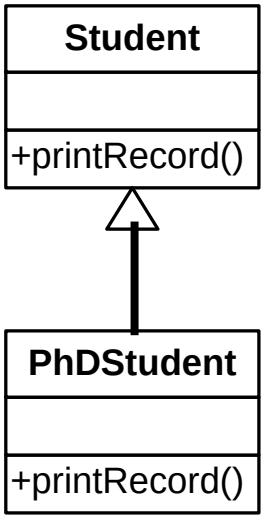
Πολυμορφισμός



- Ο πολυμορφισμός είναι εξαιρετικά σημαντικός για τη συντήρηση του λογισμικού διότι:
- ένα πρόγραμμα πελάτης έχει γραφεί ώστε να φαίνεται ότι χρησιμοποιεί αντικείμενα τύπου Student (και κατά συνέπεια καλεί τη μέθοδο printRecord)

- αλλά το ίδιο πρόγραμμα πελάτης μπορεί να λειτουργήσει με οποιοδήποτε αντικείμενο υποκλάσης της Student (ακόμα και κλάσεων που δεν υπάρχουν ακόμα αλλά θα δημιουργηθούν στο μέλλον)
- **ΧΩΡΙΣ ΚΑΜΜΙΑ ΑΠΟΛΥΤΩΣ ΑΛΛΑΓΗ ΣΤΟΝ ΚΩΔΙΚΑ ΤΟΥ ΠΕΛΑΤΗ**

Πολυμορφισμός




```
public static void main (String[ ] args)
{
ArrayList<Student> students = new ArrayList<Student>();
students.add(new Student( “5231”));
students.add(new PhDStudent(“6341”,”Makris”));

For (int i=0; i<students.size(); i++)
students.get(i).printRecord(); //Πολυμορφική κλήση
}
```

Στατικά μέλη κλάσεων

- Όλα τα αντικείμενα μιας κλάσης **έχουν τις ίδιες ιδιότητες** αλλά κάθε αντικείμενο **έχει διαφορετική τιμή** για κάθε ιδιότητα
- Υπάρχουν περιπτώσεις όπου **όλα τα αντικείμενα** μιας κλάσης θέλουμε να μοιράζονται **την ίδια τιμή** για **μία ιδιότητα**
  - Π.χ. ιδιότητα "πλήθος Υπαλλήλων" για μια κλάση Υπάλληλος
- Θα ήταν πλεονασμός κάθε αντικείμενο τύπου Υπάλληλος να έχει την ίδια τιμή για το "πλήθος Υπαλλήλων" και επιπλέον κάθε φορά που προστίθεται ένας νέος υπάλληλος να πρέπει να ενημερωθούν όλα τα αντικείμενα

Στατικά μέλη κλάσεων

- Στις περιπτώσεις αυτές αξιοποιούμε **τις στατικές ιδιότητες**
- Μία στατική ιδιότητα:
  - υφίσταται σε επίπεδο κλάσης (ίδια τιμή για όλα τα αντικείμενα)
  - υπάρχει ακόμα και όταν δεν έχουν δημιουργηθεί αντικείμενα της κλάσης (π.χ. πλήθοςΥπαλλήλων = 0)
- Δηλώνουμε ότι μια ιδιότητα είναι στατική με το **προσδιοριστικό static**
- **Για να μπορεί μια μέθοδος να προσπελάσει μια στατική ιδιότητα θα πρέπει να είναι και η μέθοδος στατική**
  - που σημαίνει ότι η μέθοδος μπορεί να κληθεί και επί της κλάσης (χωρίς να έχουμε στα χέρια μας αντικείμενο της)

Στατικά μέλη κλάσεων

```
class Employee {

   private static int count = 0;

   public Employee() {
      count++;
   }

   public static int getCount () {
      return count;
   }
}
…
   System.out.println(Employee.getCount());  // = 0
   Employee E1 = new Employee();
   System.out.println(E1.getCount());        // = 1
   Employee E2 = new Employee();
   System.out.println(E2.getCount());        // = 2
```

Overloading vs. Overriding

- OVERLOADING: Όταν έχουμε μεθόδους στην ίδια κλάση με το ίδιο όνομα αλλά με διαφορετικές υπογραφές (διαφορετικό πλήθος, τύποι ή σειρά ορισμάτων)
- Η υπογραφή μίας μεθόδου είναι το όνομα της και η λίστα με τους τύπους των ορισμάτων της μεθόδου
  - Η Java μπορεί να ξεχωρίσει μεθόδους με διαφορετική υπογραφή
  - Η υπερφόρτωση γίνεται μόνο ως προς τα ορίσματα, ΌΧΙ ως προς την επιστρεφόμενη τιμή
- OVERRIDING: Όταν έχουμε σε διαφορετικές κλάσεις (σε κλάση και υποκλάση) μεθόδους με το ίδιο όνομα την ίδια υπογραφή και επιστρεφόμενους τύπους.
  - Στην υποκλάση ισχύει η μέθοδος στιγμιοτύπου που ορίστηκε τοπικά

Hiding

- Αν μια υποκλάση ορίζει μια **στατική** μέθοδο που έχει ίδιο όνομα και υπογραφή με μια **στατική** μέθοδο της υπερκλάσης της, τότε η μέθοδος στην υποκλάση αποκρύπτει τη μέθοδο της υπερκλάσης
- Όταν πρόκειται για **μεθόδους κλάσεων (δηλαδή στατικές),** το ποια μέθοδος θα ενεργοποιηθεί **εξαρτάται από το αν καλείται από την κλάση ή την υπερκλάση**
  - Όταν η στατική μέθοδος καλείται επί αντικειμένου το ποια μέθοδος θα εφαρμοστεί εξαρτάται από το πώς έχει δηλωθεί το αντικείμενο (σε ποια κλάση ανήκει – early binding).

Παράδειγμα overriding-hiding

```
class Foo {
    public static void classMethod() {
        System.out.println("classMethod() in Foo");
    }

    public void instanceMethod() {
        System.out.println("instanceMethod() in Foo");
    }
}

class Bar extends Foo {
    public static void classMethod() {
        System.out.println("classMethod() in Bar");
    }

    public void instanceMethod() {
        System.out.println("instanceMethod() in Bar");
    }
}
```

```
class Test {
     public static void main(String[] args) {
          Foo f = new Bar();
          f.instanceMethod();
          f.classMethod();
     }
 }
```

Εκτελώντας παίρνουμε το εξής στην οθόνη:

instanceMethod() in Bar
classMethod() in Foo

Αν όμως ήταν `Bar f = new Bar();`

Θα εμφανιζόταν

instanceMethod() in Bar
classMethod() in Bar

Επεξήγηση

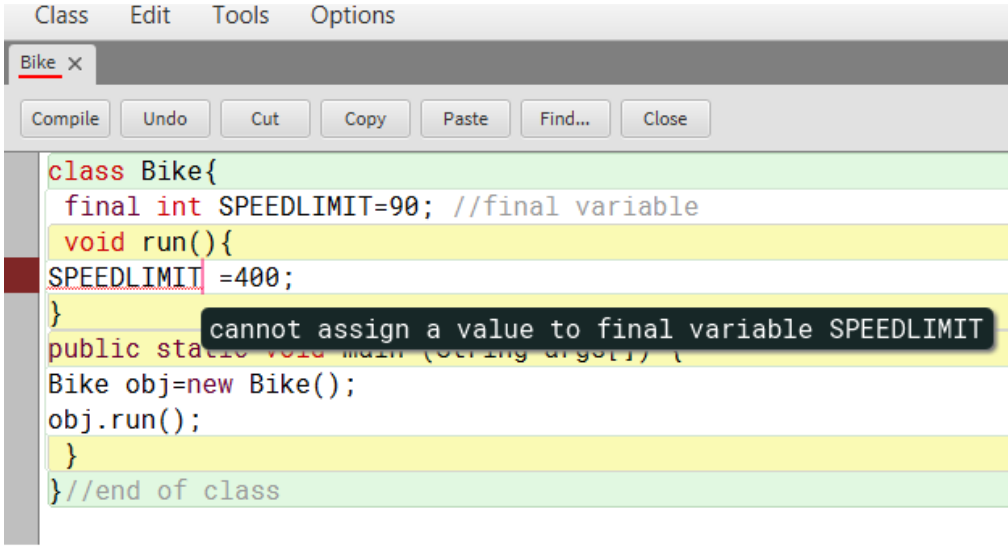
- Επειδή η instanceMethod() είναι **μέθοδος στιγμιοτύπου**, καλείται επί του αντικειμένου f που είναι τύπου Bar και άρα επικαλύπτει την αντίστοιχη μέθοδο στιγμιοτύπου στην κλάση Foo. Αν και δηλώσαμε το f σαν τύπου Foo το αντικείμενο που δημιουργήσαμε ήταν τύπου Bar (**καθυστερημένη διασύνδεση – late binding**).

- Επειδή η μέθοδος classMethod() είναι **μέθοδος κλάσης** ο compiler δεν θεωρεί ότι χρειάζεται να περιμένει κατά το χρόνο εκτέλεσης ώστε να δημιουργηθούν στιγμιότυπα. Ελέγχει απλά τον τύπο που δηλώνεται κάθε αντικείμενο και αποφασίζει με βάση αυτό ποια μέθοδο θα εφαρμόσει. Εφόσον το f δηλώθηκε σαν τύπου Foo, ο compiler θεωρεί ότι το f.classMethod() σημαίνει Foo.classMethod

Final μεταβλητές

- Δε μπορούμε να αλλάξουμε τις τιμές τους (είναι ουσιαστικά σταθερές)
- Μια final μεταβλητή που δεν έχει αρχική τιμή όταν ορίζεται λέγεται blank. Μπορεί να αρχικοποιηθεί μόνο μέσα σε ένα constructor.

```
class Bike{
 final int SPEEDLIMIT=90;//final variable
 void run(){
   SPEEDLIMIT =400;
 }
 public static void main(String args[]){
 Bike obj=new Bike();
 obj.run();
 }
}//end of class
```

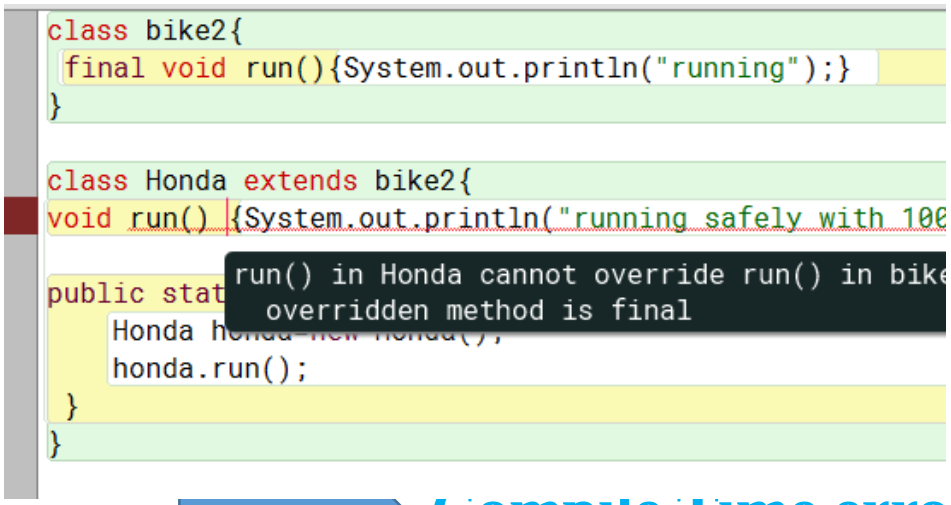


Compile Time error

Final μέθοδοι

- Όταν μια μέθοδος ορίζεται final δε μπορεί να επικαλυφθεί (να γίνει overriden)

```
class Bike{
  final void run(){System.out.println("running");}
}

class Honda extends Bike{
   void run(){System.out.println("running safely with 100kmph");}

   public static void main(String args[]){
   Honda honda= new Honda();
   honda.run();
   }
}
```

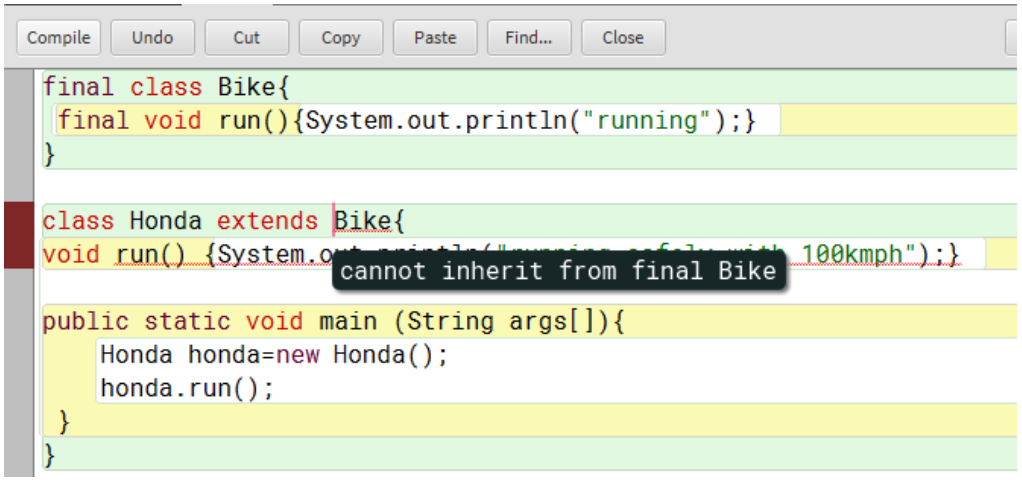


Compile Time error

Final κλάσεις

- Όταν μια κλάση ορίζεται σαν final δε μπορούμε να κατασκευάσουμε απογόνους της

```
final class Bike{}

class Honda extends Bike{
  void run(){System.out.println("running safely with 100kmph");}

  public static void main(String args[]){
  Honda honda= new Honda();
  honda.run();
  }
}
```



Compile Time error