ΔΙΕΘΝΕΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΤΗΣ ΕΛΛΑΔΟΣ

Τμήμα Μηχανικών Πληροφορικής και
Ηλεκτρονικών Συστημάτων

Αντικειμενοστρεφής Προγραμματισμός (Object Oriented Programming)

Υπερφόρτωση (Overloading) – Υπέρβαση (Overriding) – Upcasting – Downcasting – Final classes, methods

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

Περιεχόμενα Μαθήματος

- **Υπερφόρτωση (Overloading) – Υπέρβαση (Overriding) -**
- **Τελικές μέθοδοι και κλάσεις** *(Final methods and classes)*
- **Upcasting – Downcasting**

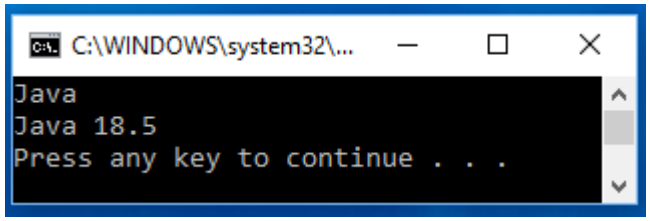
Υπερφόρτωση μεθόδων *(method overloading)*(1/7)

Στατικός ή Compile Time Πολυμορφισμός:

- Η εκτέλεση μεθόδων που έχουν το ίδιο όνομα αλλά διαφορετικές παραμέτρους (τύπος, πλήθος ή σειρά) καλείται **υπερφόρτωση μεθόδων** *(method overloading).* Στην περίπτωση αυτή ο *τύπος των δεδομένων,* ο *αριθμός ή η σειρά των παραμέτρων* λαμβάνονται υπόψη στον ***προσδιορισμό της μεθόδου που θα εκτελεστεί***.

(1) Διαφορετικός πλήθος δεδομένων

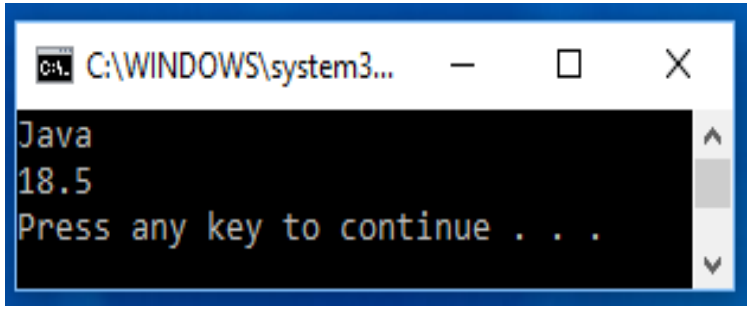
```
class Overloading1 {
   public void Emfanise(String s)
     {System.out.println(s); }
   public void Emfanise(String s, double i)
     {System.out.println(s + " "+ i);}}

class DifferentNumberOfParameters {
  public static void main(String args[]) {
      Overloading1 obj = new Overloading1();
      obj.Emfanise("Java");
      obj.Emfanise("Java", 18.50); } }
```



Υπερφόρτωση μεθόδων *(method overloading)*(2/7)

(2) Διαφορετικός τύπος δεδομένων

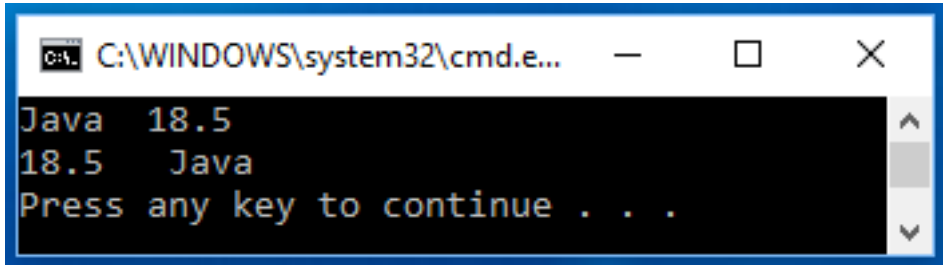
```
class Overloading2 {
 public void Emfanise(String s)
   {System.out.println(s); }
 public void Emfanise(double s)
   {System.out.println(s);}}

class DifferenceInDataTypes {
  public static void main(String args[]) {
    Overloading2 obj = new Overloading2();
    obj.Emfanise("Java");
    obj.Emfanise(18.50);
  }
}
```



Υπερφόρτωση μεθόδων *(method overloading)*(3/7)

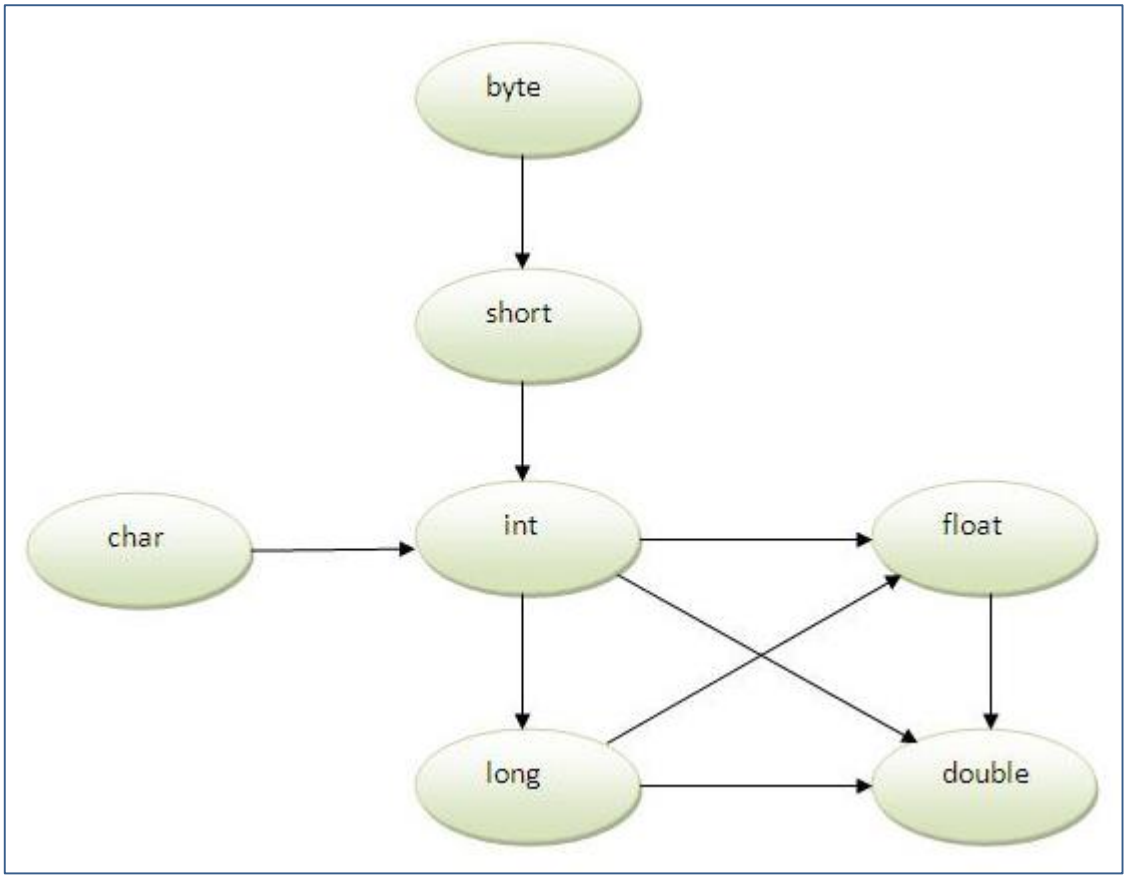
(3) Διαφορετική σειρά τύπων δεδομένων

```
class Overloading3 {
   public void Emfanise(String s, double d)
     {System.out.println(s+"  " + d); }
   public void Emfanise(double d, String s)
     {System.out.println(d +"   " + s);}}

class DifferentSequenceOfDataTypes {
   public static void main(String args[]) {
       Overloading3 obj = new Overloading3();
       obj.Emfanise("Java", 18.50);
       obj.Emfanise(18.50, "Java"); } }
```



Υπερφόρτωση μεθόδων *(method overloading)*(4/7)

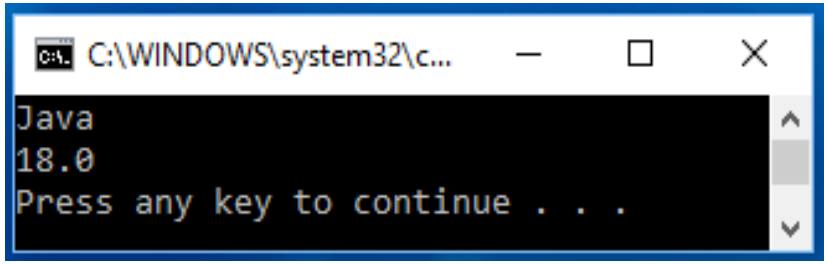
- Αν δεν ταιριάζουν οι τύποι των παραμέτρων, και για να επιλέξει η java την μέθοδο που θα εκτελέσει μπορεί να κάνει ***αυτόματη μετατροπή τύπου δεδομένων (type promotion)*** *(π.χ. από int σε double, κλπ.)*



Υπερφόρτωση μεθόδων *(method overloading)*(5/7)

Type Promotion:

```
class Overloading4 {
 public void Emfanise(String s)
  {System.out.println(s); }
 public void Emfanise(double d)
  {System.out.println(d);}}

class TypePromotion {
 public static void main(String args[]) {
   Overloading4 obj = new Overloading4();
   obj.Emfanise("Java");
   obj.Emfanise(18);//int se double
  }
}
```



Υπερφόρτωση μεθόδων *(method overloading)*(6/7)

- Δεν είναι δυνατή η υπερφόρτωση μεθόδων με την αλλαγή του επιστρεφόμενου τύπου *(return type)* της μεθόδου *(ασάφεια κλήσης)*.

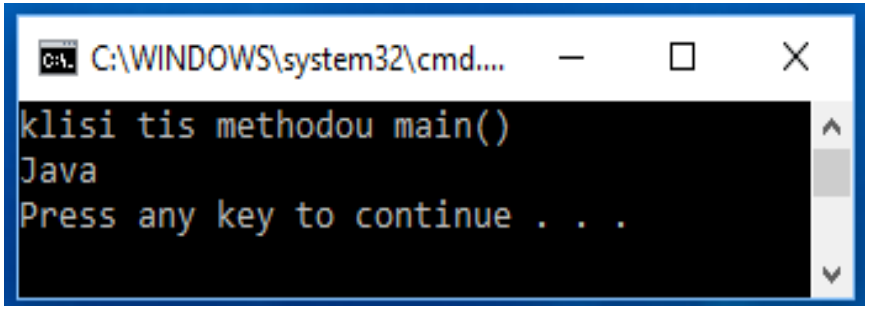
```
class ReturnType{
 int sum(int a, int b){return (a + b);}
 double sum(int a, int b){return(a + b);}

 public static void main(String args[]){
   ReturnType obj=new ReturnType();
   int result=obj.sum(31,35); //Compile Time Error
 }                                //ασάφεια, ποια sum θα κληθεί;
}
```

```
ReturnType.java:3: error: method sum(int,int) is already defined in
class ReturnType
  double sum(int a,int b){return(a+b);}
         ^
1 error
```

Υπερφόρτωση μεθόδων *(method overloading)*(7/7)

- Μπορούμε να υπερφορτώσουμε την **main() – μέθοδο**.

```
class MainNameOverloading{
    public static void main(String s){
      System.out.println(s);
    }

    public static void main(String args[]){
      System.out.println("klisi tis methodou main()");
      main("Java");
    }
}
```



Υπέρβαση/Υπερκάλυψη Μεθόδων *(Method Overriding)* *(1/6)*

Η υπέρβαση μεθόδων βασίζεται στον **Πολυμορφισμό** της Αντικειμενοστρέφειας *(Πολυμορφική ιδιότητα των αντικειμένων)*, επιτρέποντας τον προγραμματιστή να δημιουργήσει δύο μεθόδους με το ίδιο όνομα και υπογραφή, με αποτέλεσμα να υλοποιούνται ***διαφορετικά κατά τον χρόνο εκτέλεσης*** *(run time)* ***ανάλογα με τον τύπο του αντικειμένου***.

- Η υπέρβαση μεθόδων επιτρέπει να γράφουμε ***επεκτάσιμο*** και ***ευέλικτο*** κώδικα, γιατί παρέχουμε νέα λειτουργικότητα με ελάχιστες αλλαγές στον κώδικα.
- ***Υπέρβαση ή υπερκάλυψη μεθόδου*** *είναι η διαδικασία όπου μία υποκλάση επανα-υλοποιεί μία μέθοδο που κληρονόμησε από την υπερκλάση.*

Υπέρβαση/Υπερκάλυψη Μεθόδων *(Method Overriding)* *(2/6)*

- Στην υποκλάση γράφουμε την μέθοδο με την ίδια υπογραφή και στο σώμα της μεθόδου τον κώδικα που υλοποιεί τη νέα λειτουργικότητα.
- Κλήση της μεθόδου με αντικείμενο της υποκλάσης συνεπάγεται κλήση της μεθόδου της υποκλάσης με την νέα λειτουργικότητα, ενώ κλήση με αντικείμενο της υπερκλάσης, θα κληθεί η μέθοδος της υπερκλάσης με την αρχική λειτουργικότητα.

Κανόνες:

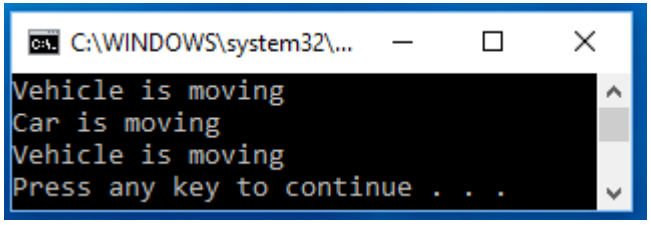
- ***Εφαρμόζεται μόνο σε μεθόδους που κληρονομούνται.***
- ***Δεν αλλάζουμε την υπογραφή ή τον τύπο του επιστρεφόμενου αποτελέσματος της μεθόδου που υπερβαίνουμε.***

Υπέρβαση/Υπερκάλυψη Μεθόδων *(Method Overriding)* *(3/6)*

- Μπορούμε να αυξήσουμε την προσβασιμότητα στη μέθοδο (π.χ. από protected σε public).
- Δεν μπορούμε να υπερβούμε τελικές (final) μεθόδους.
- Δεν μπορούμε να υπερβούμε στατικές (static) μεθόδους, αλλά να τις επανα-ορίσουμε.
- Μια υποκλάση στο ίδιο πακέτο με την υπερκλάση μπορεί να υπερβεί οποιαδήποτε μέθοδο της υπερκλάσης που δεν είναι private ή final.
- Μια υποκλάση σε διαφορετικό πακέτο μπορεί να υπερβεί τις μη τελικές μεθόδους που είναι public ή protected.
- Μπορούμε να μειώσουμε ή να διαγράψουμε δηλωμένες εξαιρέσεις *(exceptions)*, όχι όμως να προσθέσουμε νέες.
- Δεν μπορούμε να υπερβούμε τους δομητές.

Υπέρβαση/Υπερκάλυψη Μεθόδων *(Method Overriding)* *(4/6)*

```
class Vehicle {
   public void move () {
        System.out.println ("Vehicle is moving");}
}

class Car extends Vehicle {
   public void move () {
     super.move ();          // kalei tin move() tis yperklasis
     System.out.println ("Car is moving");}
}

public class TestCar {
   public static void main (String args []){
        Vehicle c = new Car (); // Vehicle reference se antikeimeno Car
        c.move ();              //kalei tin methodo stin klasi Car
        Vehicle v = new Vehicle();
        v.move(); } }
```



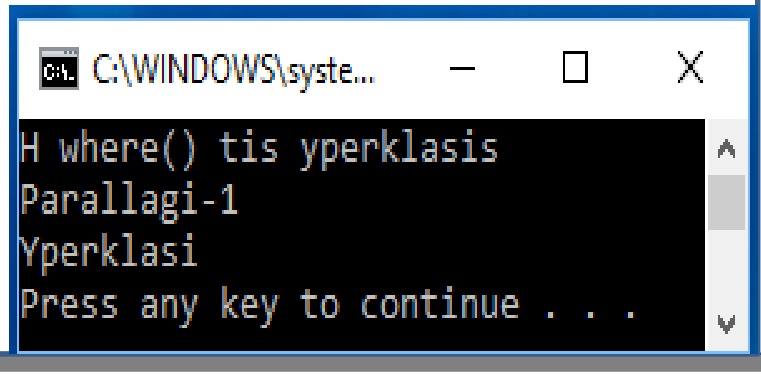
Υπέρβαση/Υπερκάλυψη Μεθόδων *(Method Overriding)* *(5/6)*

- **Δεν μπορούμε να υπερβούμε final – static και private μεθόδους.**
- Στο παρακάτω παράδειγμα θα πάρουμε *compilation error* αν προσπαθήσουμε να υπερβούμε την final μέθοδο, ενώ δεν θα μπορέσουμε να υπερβούμε επίσης τις static και private μεθόδους.

```
class Yperklasi{
  public final String version(){
       where();  //klisi tis methodou where() tis yperklasis
       return "Parallagi-1"; }

  public static String name(){
       return "Yperklasi";}

  private void where(){
       System.out.println("H where() tis yperklasis"); }
}
```

Υπέρβαση/Υπερκάλυψη Μεθόδων *(Method Overriding)* *(6/6)*

```
class Ypoklasi extends Yperklasi{
  // Compilation Error : Final method can't be overridden in Java
  // public final String version(){
  //     return "Parallagi-2";
  // }

 public static String name(){return "Ypoklasi";}

    //Idia ypografi, alla krimeni methodos
   private void where(){
       System.out.println("H where() tis ypoklasis");}

 class FinalStaticPrivateOverridingTest {
   public static void main(String args[]) {
     Yperklasi y = new Ypoklasi();
     System.out.println(y.version());
     System.out.println(y.name());}}
```

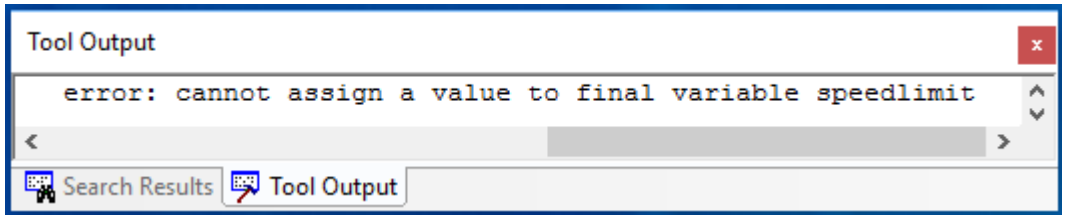


Τελικές μεταβλητές, πεδία, παράμετροι, κλάσεις και μέθοδοι *(1/3)*

Τελικές μεταβλητές - πεδία – παράμετροι *(final variables – fields - parameters)*

- Τελικά πεδία, παράμετροι και τοπικές μεταβλητές δεν μπορούν να αλλάξουν τιμή, όταν η τιμή έχει προκαθοριστεί. Στο παράδειγμα θα πάρουμε μήνυμα λάθους επειδή πάμε να τροποποιήσουμε την τιμή του final πεδίου speedlimit.

```
class Car{
  final int speedlimit=170; //final variable
  void run(){
    speedlimit=220;
  }
  public static void main(String args[]){
    Car myCar=new Car();
    myCar.run();
  }
}
```



Τελικές μεταβλητές, πεδία, παράμετροι, κλάσεις και μέθοδοι *(2/3)*

Τελικές κλάσεις *(final classes)*

- Δεν μπορούν να κληρονομηθούν, άρα δεν μπορούν επεκταθούν *(extended)* από υποκλάσεις.
- Χρησιμοποιούμε την λέξη κλειδί **final**. Οι κλάσεις Math, String, Wrapper-κλάσεις (Double, Integer, κλπ.) είναι final.

Σύνταξη:

```
public final class Teliki
{
  <σώμα κλάσης>
 }
```

Αν γράψουμε: **class test extends Teliki {…}** κατά την μεταγλώττιση θα λάβουμε το μήνυμα: **//compilation error: cannot inherit from final class**

Τελικές μεταβλητές, πεδία, παράμετροι, κλάσεις και μέθοδοι *(3/3)*

Τελικές μέθοδοι (final methods)

- Δεν μπορούν να επαναοριστούν (overridden) στις υποκλάσεις.
- Στατικές μέθοδοι και μέθοδοι με private πρόσβαση είναι έμμεσα final.
- Οι κλήσεις σε final μεθόδους υλοποιούνται κατά τη μεταγλώττιση *(static binding).*

```
class Loan{
 public final String getLonType() {
    return “Home loan";
 }
}
class MyLoan extends Loan{
   @Override
   public final String getLoanType() {
      return “my personal loan";   //compilation error: overridden method is final
    }
 }
```

Upcasting – Downcasting *(1/9)*

- **Upcasting:** Η ανάθεση ενός αντικειμένου της υποκλάσης σε αναφορά της υπερκλάσης *(casting a subtype to supertype)*.
- **Διαφορετικά:** Η java επιτρέπει, μέσω της κληρονομικότητας, σε ένα αντικείμενο της υποκλάσης να αντιμετωπίζεται σαν αντικείμενο της υπερκλάσης.
- Το upcasting γίνεται αυτόματα, δεν χρειάζεται κάποιο προσδιοριστικό, γιατί η υποκλάση είναι μια εξειδίκευση της υπερκλάσης.

- **Downcasting:** Η ανάθεση ενός αντικειμένου της υπερκλάσης σε αναφορά της υποκλάσης (casting a supertype to subtype).
- Γίνεται ρητά από τον προγραμματιστή.
- Μπορεί να αποτύχει αν το πραγματικό αντικείμενο είναι άλλου τύπου.

```
Animal anim = new Cat();
Dog dog = (Dog) anim; //error- ClassCastException, γιατί το πραγματικό αντικείμενο είναι Cat
```

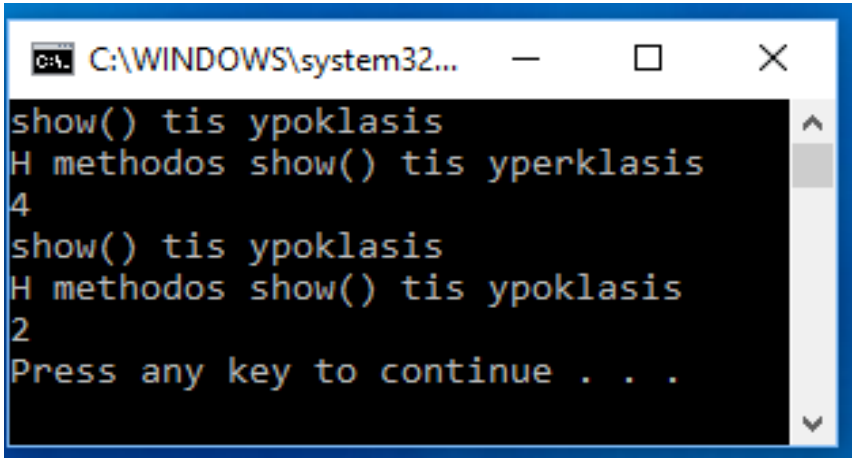
Upcasting – Downcasting *(2/9)*

Παράδειγμα Upcasting – Downcasting:

```
class Yperklasi{
  int x=4;
  void show(){
    System.out.println("show() tis yperklasis");}

  void MonoYperklasiShow(){
    System.out.println("H methodos show() tis yperklasis");}
}

class Ypoklasi extends Yperklasi{
  int x=2;
  void show(){
      System.out.println("show() tis ypoklasis");}
  void MonoYpoklasiShow(){
      System.out.println("H methodos show() tis ypoklasis");}}
```

Upcasting – Downcasting *(3/9)*

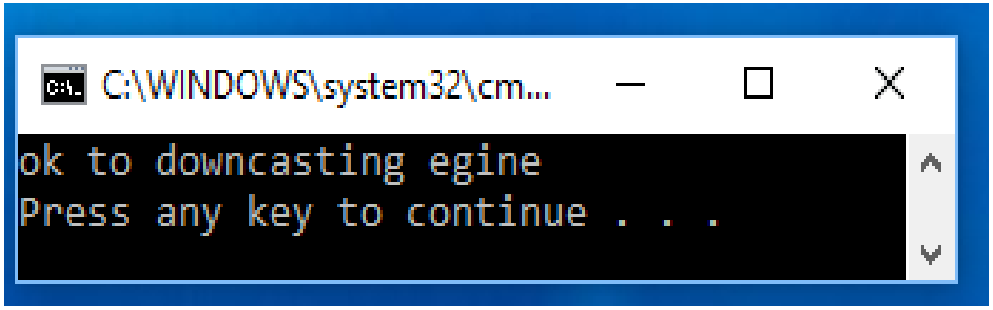
```
public class UpcastingDowncastingEx {
  public static void main(String[] args) {
    Yperklasi yper = new Ypoklasi(); //upcasting
    yper.show();
    yper.MonoYperklasiShow();    //klironomeitai kai ekteleitai
    System.out.println(yper.x); //an kai ypervasi tou x,
                                //emfanizetai to 4 tis yperklasis
    Ypoklasi ypo=(Ypoklasi)yper; //downcasting
    ypo.show();
    ypo.MonoYpoklasiShow();
    System.out.println(ypo.x);
  }
}
```



Upcasting – Downcasting *(4/9)*

Παράδειγμα: Σωστό Downcasting με την χρήση της instanceof

```
class Animal{ }
class Dog extends Animal  {
 static void Amethod(Animal animal) {
      if (animal instanceof Dog) {
          Dog dog=(Dog)animal;  //downcasting
          System.out.println("ok to downcasting egine");}
}

public static void main (String [] args) {
 Animal animal=new Dog();
 Dog.Amethod(animal); }
}
```



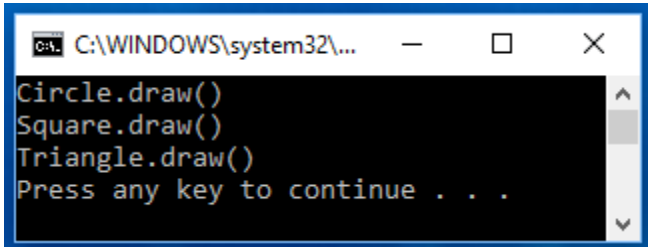
Upcasting – Downcasting *(5/9)*

Παράδειγμα: Upcasting και Πολυμορφισμός

```
import java.util.*;
abstract class Shape {
  void draw() { System.out.println(this + ".draw()"); }
  abstract public String toString();}
 class Circle extends Shape {
   public String toString() {return "Circle";}}
 class Square extends Shape {
   public String toString() {return "Square";}}
 class Triangle extends Shape {
   public String toString() {return "Triangle";}}
 class Shapes {
  public static void main(String[] args) {
   List<Shape> shapeList = Arrays.asList(new Circle(), new Square(),
                    new Triangle());
 for (Shape shape: shapeList)
  shape.draw();}}
```

Upcasting

Πολυμορφισμός



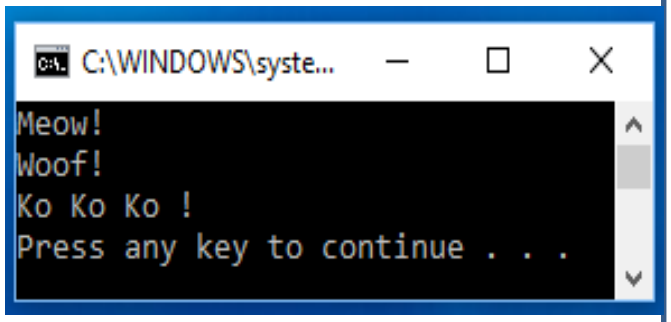
Αντί Downcasting χρήση Πολυμορφισμού και dynamic binding *(6/9)*

Παράδειγμα – 1ο : Η άσκηση με downcasting/instanceof και μετά με χρήση Πολυμορφισμού και δυναμικής δέσμευσης

```
class Animal{ }
class Dog extends Animal {
  public void woof() {
    System.out.println("Woof!");}}

 class Cat extends Animal {
   public void meow() {
    System.out.println("Meow!");}}

 class Chicken extends Animal {
   public void kikiriko() {
    System.out.println("Ko Ko Ko !"); } }
```

Αντί Downcasting χρήση Πολυμορφισμού και dynamic binding *(7/9)*

```
class Example2 {
 public static void main(String[] args) {
   makeItTalk(new Cat());
   makeItTalk(new Dog());
   makeItTalk(new Chicken()); }

 public static void makeItTalk(Animal animal) {
  if (animal instanceof Cat) {
    Cat cat = (Cat) animal;
    cat.meow();}
  else if (animal instanceof Dog) {
    Dog dog = (Dog) animal;
    dog.woof();}
  else if (animal instanceof Chicken) {
    Chicken chicken = (Chicken)animal;
    chicken.kikiriko();}
 } }
```



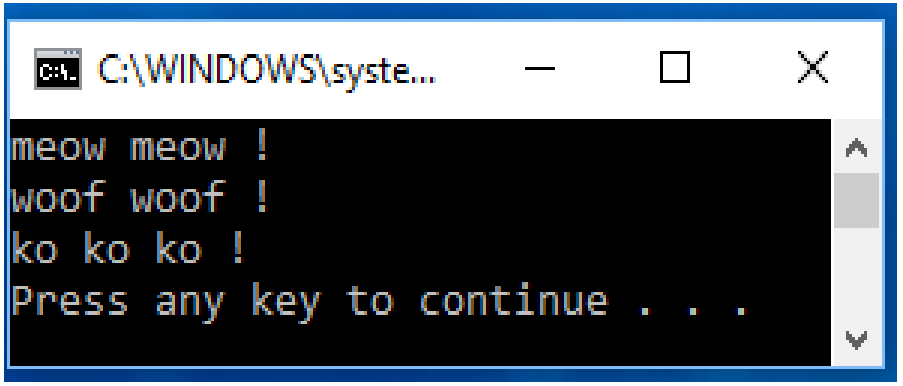
Αντί Downcasting χρήση Πολυμορφισμού και dynamic binding *(8/9)*

Παράδειγμα – 2ο : Η άσκηση με χρήση Πολυμορφισμού και δυναμικής δέσμευσης

```
abstract class Animal {
  public abstract void talk();}

class Dog extends Animal {
  public void talk() {
   System.out.println("woof woof !");}}

class Cat extends Animal {
  public void talk() {
  System.out.println("meow meow !");}}

 class Chicken extends Animal {
   public void talk() {
    System.out.println("ko ko ko !");}}
```

Αντί Downcasting χρήση Πολυμορφισμού και dynamic binding *(9/9)*

```
class Example1 {
  public static void main(String[] args) {
    makeItTalk(new Cat());
    makeItTalk(new Dog());
    makeItTalk(new Chicken());
  }

  public static void makeItTalk(Animal animal) {
    animal.talk(); }
}
```



Άσκηση

Σε ένα Τμήμα υπάρχουν μαθήματα **θεωρητικά** (μόνο θεωρία), μόνο **εργαστηριακά** και **μεικτού τύπου**(θεωρία + εργαστήριο). Ο τελικός βαθμός για τα μαθήματα υπολογίζεται:

- Θεωρητικό: 70% βαθμός εξέτασης, 30% βαθμός εργασίας.
- Εργαστηριακό: 100% βαθμός εξέτασης.
- Μεικτό: 60% βαθμός εξέτασης θεωρίας, 40% βαθμός εξέτασης εργαστηρίου.

1η έκδοση-λύση:

(1) Να γίνει το πρόγραμμα που υλοποιεί μέσω **πολυμορφισμού** και **dynamic binding** ένα Ν-πλήθος μαθημάτων. Η κλάση **Course**, θα είναι αφηρημένη, τα πεδία και οι μέθοδοι θα καθοριστούν από εσάς, εκτός από την **αφηρημένη μέθοδο TelikosVathmos()** που θα υλοποιείται διαφορετικά στις 3 υποκλάσεις (**Theoritiko, Ergastiriako, Meikto**). Τα πεδία και οι λοιπές μέθοδοι των υποκλάσεων θα αποφασιστούν επίσης από εσάς.

2η έκδοση – παραλλαγή:

(2) Να γίνει επέκταση του προγράμματος με την προσθήκη ενός νέου μαθήματος **MeiktoMeErgasia,** του οποίου ο **TelikosVathmos()** υπολογίζεται ως:

- Μεικτό με εργασία: 40% βαθμός εξέτασης θεωρίας, 40% βαθμός εξέτασης εργαστηρίου και 20% βαθμός εργασίας. Το πρόγραμμα θα υλοποιεί ένα Ν-πλήθος μαθημάτων όλων των τύπων.