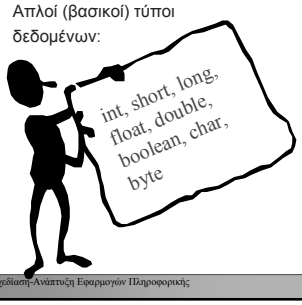
Σχεδίαση-Ανάπτυξη Εφαρμογών Πληροφορικής - Εβδομάδα 3

Εβδομάδα 3: Κλάσεις συσκευαστές, αφηρημένες κλάσεις και διαπροσωπείες

Αντικείμενα και μη-αντικείμενα

- Η Java παρέχει τύπους αντικειμένων και απλούς τύπους δεδομένων

Απλοί (βασικοί) τύποι δεδομένων:



Αντικείμενα:

Οτιδήποτε άλλο! (συμπεριλαμβανομένων των Strings!)

Αντικείμενα και μη-αντικείμενα (2)

- Ισχύουν διαφορετικοί κανόνες για τα τους τύπους αντικείμενων και τους βασικούς τύπους
  - Τα αντικείμενα κληρονομούν (έστω και έμμεσα) από την κλάση **`Object`**
  - Αυτό δεν ισχύει για τους βασικούς τύπους

Ποια είναι η συνέπεια;

⇨ Οι βασικοί τύποι δεν είναι υπο-τύποι της κλάσης **`Object`**!

Βασικοί τύποι και συλλογές δεδομένων

Πώς εισάγουμε έναν ακέραιο σε μια συλλογή δεδομένων τύπου **`ArrayList`**;

- Τα στοιχεία της **`ArrayList`** είναι τύπου **`Object`**
- Ο βασικός τύπος **`int`** δεν είναι υπο-τύπος του **`Object`**
- Ένας "**`int`**" δεν μπορεί να εισαχθεί σε μία "**`ArrayList`**"

Η λύση: κλάσεις συσκευαστές [wrapper Classes]

- Κάθε βασικός τύπος έχει μία αντίστοιχη κλάση η οποία μπορεί να χρησιμοποιηθεί όταν απαιτείται ένα πραγματικό αντικείμενο.
  Για παράδειγμα:

Βασικός τύπος	τύπος κλάσης
int	**Integer**
char	**Character**
boolean	**Boolean**

Μία λίστα από "`Integers`"

Συσκευασία του "**`int`**" σε αντικείμενο **`Integer`** πριν την προσθήκη του στην **`ArrayList`**

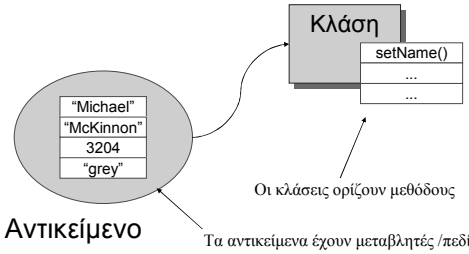
```
ArrayList intList = new ArrayList();
int anInt = 42;
intList.add(new Integer(anInt));
...
Integer anInteger;
anInteger = (Integer)intList.get(2);
int myInt = anInteger.intValue();
```

Μετά την εξαγωγή του "**`Integer`**" από την **`ArrayList`**, λαμβάνουμε τον "**`int`**" από τον **`Integer`**.

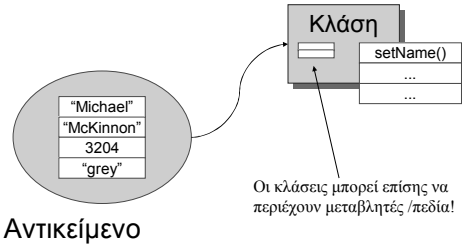
Σχεδίαση-Ανάπτυξη Εφαρμογών Πληροφορικής - Εβδομάδα 3

Μεταβλητές /πεδία κλάσης [class variables]

Τα αντικείμενα “ανήκουν” σε κλάσεις...



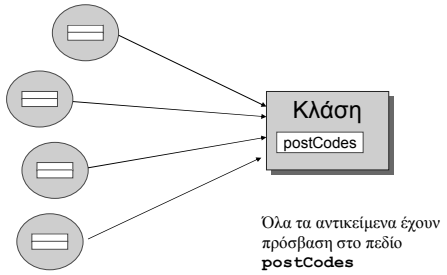
Μεταβλητές /πεδία κλάσης (2)



Μεταβλητές /πεδία κλάσης (3)

- Τα πεδία κλάσης τα κατέχει η («αποθηκεύονται στην») κλάση
- Τα πεδία κλάσης είναι προσπελάσιμα από αντικείμενα
- Υπάρχει μόνο ένα αντίγραφο των πεδίων κλάσης στο σύστημα (αποθηκεύονται «στην κλάση»)
- Όλα τα αντικείμενα μοιράζονται τα πεδία
- Τα πεδία κλάσης μπορεί να χρησιμοποιηθούν για την ανταλλαγή πληροφοριών μεταξύ αντικειμένων (μια όχι και τόσο καλή πρακτική)

Μεταβλητές /πεδία κλάσης (4)



Μεταβλητές /πεδία κλάσης (συντακτικό)

```
class Person
{
      private static int numberOfPeople = 0;

      Person()
      {
            numberOfPeople++;
      }
      ...
}
```


Μέθοδοι κλάσης [class methods]

- Οι μέθοδοι μπορεί να ανήκουν σε μία κλάση (αντί σε ένα αντικείμενο)
- Καλούνται μέσω της χρήσης του ονόματος της κλάσης (και όχι του αντικειμένου)

Στην κλάση **Integer**:

```
public static int parseInt(String s)
```

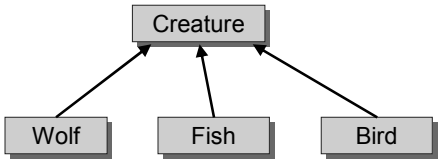
Χρήση:

```
String input;
input = ...  // get input from user
int players = Integer.parseInt(input);
```

Σχεδίαση-Ανάπτυξη Εφαρμογών Πληροφορικής - Εβδομάδα 3

Κλάσεις αφηρημένου τύπου [abstract classes]

- Μερικές φορές δεν είναι δυνατόν να υλοποιηθούν όλες οι μέθοδοι!
- Παράδειγμα: κλάση **`creature`**, method **`move`**



Κλάσεις αφηρημένου τύπου (2)

- Στις κλάσεις αφηρημένου τύπου, δεν υλοποιούνται όλες οι μέθοδοι
- Μερικές μέθοδοι δηλώνονται ως «αφηρημένες» [abstract]
- Εάν μία κλάση περιέχει «αφηρημένες» μεθόδους, τότε πρέπει να δηλωθεί ως **`"abstract"`**
- Δεν είναι δυνατό να δημιουργήσουμε αντικείμενα με βάση μια κλάση αφηρημένου τύπου

Κλάσεις αφηρημένου τύπου (3)

Παράδειγμα:

```
abstract class Creature
{
        private age;

        public int getAge()
        {
                return age;
        }

        public abstract void move();
}
```

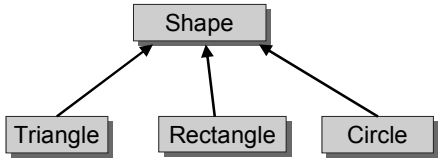
Μερικές μέθοδοι μπορεί να έχουν υλοποιηθεί

Άλλες, όχι

«Απτές» κλάσεις [concrete classes]

- Concrete = απτή, χειροπιαστή
- Ως απτές [concrete] κλάσεις αναφέρονται όσες κλάσεις δεν είναι αφηρημένου τύπου
- Αντικείμενα μπορεί να δημιουργηθούν μόνο με βάση τις απτές κλάσεις

Παράδειγμα: κλάση **`Shapes`**




```
abstract class Shape
{
   public abstract void circumference();
}
```


Λόγος ύπαρξης των «αφηρημένων» κλάσεων

- Εάν δεν μπορούμε να δημιουργήσουμε αντικείμενα με βάση τις κλάσεις αφηρημένου τύπου, ποια η χρησιμότητα τους;

Απάντηση:

- Πολυμορφισμός!
- Μία «αφηρημένη» κλάση είναι ένας τύπος δεδομένων
- Μπορούμε να δηλώσουμε μεταβλητές αυτού του τύπου
- Υπο-τύποι μπορεί να δημιουργηθούν

Σχεδίαση-Ανάπτυξη Εφαρμογών Πληροφορικής - Εβδομάδα 3

Προσομοίωση διακριτών γεγονότων

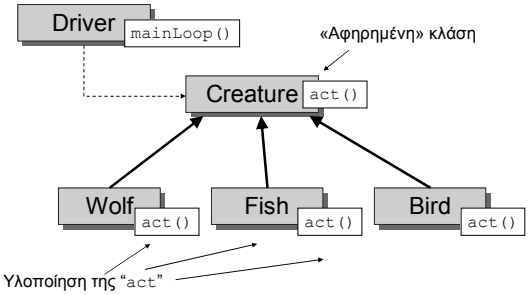
[Discrete Event Simulation]

- προσομοίωση συστήματος
- το σύστημα αποτελείται από ένα σύνολο ενεργών στοιχείων [actors]
- Βασίζεται σε ένα επαναλαμβανόμενο σύνολο εντολών [main loop] το οποίο δίνει τη δυνατότητα στα ενεργά στοιχεία να δράσουν

Παραδείγματα:

- Προσομοίωση περιβαντολογικών συστημάτων, αεροδρομίων, συγκοινωνιακών συστημάτων, πρόγνωση καιρού, πυρηνικών εκρήξεων, κλπ.

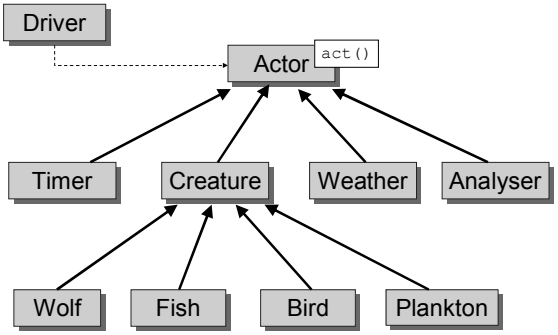
Προσομοίωση διακριτών γεγονότων



Προσομοίωση – βασικός βρόγχος

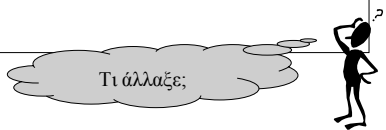
```
private void mainLoop()
{
   while(!endOfSimulation())
   {
      // let every actor do what they want to do
      for(int i=0; i < actors.size(); i++)
      {
         Actor actor = (Actor)actors.get(i);
         actor.act();
      }
   }
}
```


Πιο περίπλοκο παράδειγμα



Βασικός βρόγχος

```
private void mainLoop()
{
   while(!endOfSimulation())
   {
      // let every actor do what they want to do
      for(int i=0; i < actors.size(); i++)
      {
         Actor actor = (Actor)actors.get(i);
         actor.act();
      }
   }
}
```



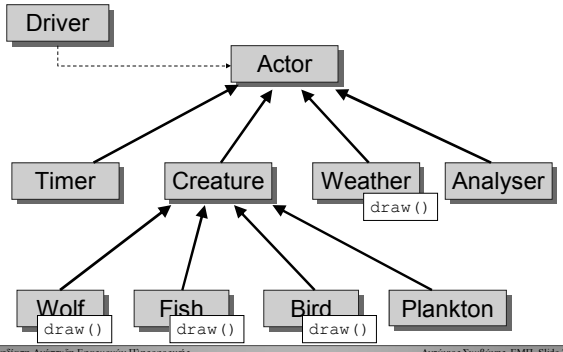
Μία ακόμα προσθήκη…

- Να προστεθεί κώδικα που εμφανίζει την προσομοίωση στην οθόνη

```
private void mainLoop()
{
   while(!endOfSimulation())
   {
      // let every actor do what they want to do
      for(int i=0; i < actors.size(); i++)
      {
         Actor actor = (Actor)actors.get(i);
         actor.act();
      }
      ... // draw all visible actors on screen
   }
}
```

Σχεδίαση-Ανάπτυξη Εφαρμογών Πληροφορικής - Εβδομάδα 3

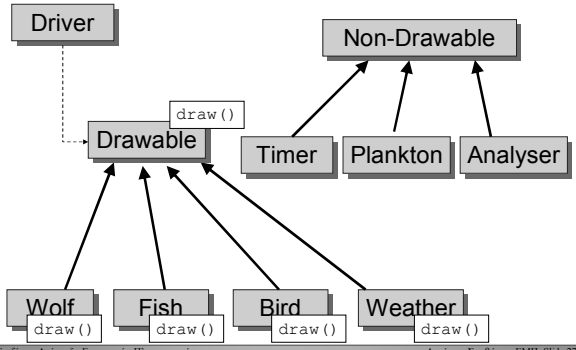
Ένα πιο ενδιαφέρον παράδειγμα



Θα επιθυμούσαμε να…

```
private void mainLoop()
{
  while(!endOfSimulation())
  {
    // let every actor do what they want to do
    for(int i=0; i < actors.size(); i++)
    {
      Actor actor = (Actor)actors.get(i);
      actor.act();
    }
    for(int i=0; i < drawables.size(); i++)
    {
      Drawable drawable = (Drawable)drawables.get(i);
      drawable.draw();
    }
  }
}
```


Μια διαφορετική ιεραρχία



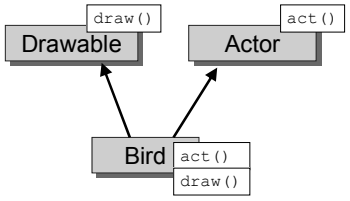
Το πρόβλημα

```
private void mainLoop()
{
  while(!endOfSimulation())
  {
    // let every actor do what they want to do
    for(int i=0; i < actors.size(); i++)
    {
      Actor actor = (Actor)actors.get(i);
      actor.act();
    }
    for(int i=0; i < drawables.size(); i++)
    {
      Drawable drawable = (Drawable)drawables.get(i);
      drawable.draw();
    }
  }
}
```

Έπαψε να είναι σωστός κώδικας.

Πολλαπλή κληρονομικότητα

[Multiple Inheritance]




```
Bird myBird = new Bird();
actors.add(myBird);
drawables.add(myBird);
```


Το επόμενο πρόβλημα…

- Η Java δεν επιτρέπει να κληρονομήσουμε από περισσότερες της μίας κλάσης!
- Η πολλαπλή κληρονομικότητα είναι χρήσιμη αλλά προκαλεί προβλήματα (κάνει πιο πολύπλοκο) στον ορισμό της γλώσσας
- Για το λόγο αυτό, μερικές γλώσσες δεν επιτρέπουν την πολλαπλή κληρονομικότητα (πχ. Java)
- Πως αντιμετωπίζουμε το πρόβλημα;

Σχεδίαση-Ανάπτυξη Εφαρμογών Πληροφορικής - Εβδομάδα 3

Η λύση: διαπροσωπείες [interfaces]

- Μια διαπροσωπεία μοιάζει με μία πλήρως «αφηρημένη» κλάση
- Οι διαπροσωπείες [interfaces]:
  - περιέχουν μόνο δηλώσεις μεθόδων [method signatures]
  - περιέχουν μόνο ορισμούς σταθερών [constants]
  - δεν περιέχουν πεδία
  - δεν περιέχουν υλοποιήσεις μεθόδων

Διαπροσωπείες και κληρονομικότητα

- Μία κλάση μπορεί να κληρονομήσει μόνο από μία άλλη κλάση, αλλά…
- …μπορεί να «κληρονομήσει» από περισσότερες από μία διαπροσωπείες!

Ορολογία (για να ξεχωρίζουμε τις δυο αυτές μορφές κληρονομικότητας):

- Μία κλάση **επεκτείνει** [extends] μια υπερ-κλάση.
- Μία κλάση **υλοποιεί** [implements] μία διαπροσωπεία.

Παράδειγμα

```
interface Drawable
{
   /**
    * Draw this entity on screen.
    */
   void draw();
}
```

Οι μέθοδοι των διαπροσωπειών είναι εξ' ορισμού "`public`" – δεν χρειάζεται μετατροπέας πρόσβασης

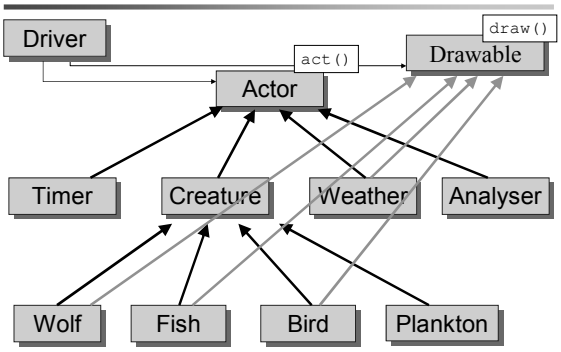
Η υπο-κλάση

```
class Bird extends Creature
   implements Drawable
{
   ...

   /**
    * Act. Here that means fly around and search
    * for bird food.
    */
   public void act()
   { ... }

   /**
    * Draw this bird on screen.
    */
   public void draw()
   { ... }
}
```


Η λύση



Πολλαπλές διαπροσωπείες

```
class Bird extends Actor
   implements Drawable, Storable
{
   ...
}
```

- Το πολύ μία υπερ-κλάση
- Απεριόριστος αριθμός διαπροσωπειών

Πολλαπλές διαπροσωπείες (2)

```
class Bird
   implements Actor, Drawable, Storable
{
   ...
}
```

- Η κλάση Actor θα μπορούσε να ήταν μία διαπροσωπεία.

Οι διαπροσωπείες ως τύποι δεδομένων

- Οι διαπροσωπείες ορίζουν τύπους δεδομένων (όπως και οι κλάσεις).
- Όπως ισχύει και για τις κλάσεις αφηρημένου τύπου, δεν μπορούμε να δημιουργήσουμε ένα στιγμιότυπο [instance] από μία διαπροσωπεία, αλλά μπορούμε να ορίζουμε μεταβλητές
- Στιγμιότυπα των κλάσεων που υλοποιούν τις διαπροσωπείες μπορεί να καταχωρηθούν σε μεταβλητές τύπου διαπροσωπειών

```
Drawable nextToDraw;
Bird myBird = new Bird();

nextToDraw = myBird;
```