Αντικειμενοστρεφής Προγραμματισμός

Παναγιώτης Αδαμίδης
adamidis@it.teithe.gr

Διασυνδέσεις/Διεπαφές (Interfaces)

Γενικά περί διασυνδέσεων (1/2)

"An interface is a named collection of method declarations (without definitions). An interface can also declare constants."
– *The Java Tutorial*, Sun Microsystems

- Σκοπός: Καθορισμός ενός συνόλου κανόνων συμπεριφοράς που μπορεί να παρέχει κάποια κλάση.
- Οι διασυνδέσεις παρέχουν ένα πιο επιτηδευμένο τρόπο οργάνωσης και ελέγχου των αντικειμένων παρέχοντας έναν τρόπο για να καθορίσουμε τι θα κάνει μία κλάση αλλά όχι πως θα το κάνει.
- Η χρήση διασυνδέσεων μας επιτρέπει να ορίσουμε την μορφή μίας κλάσης (ονόματα μεθόδων, παράμετροι, τύπος επιστροφής) αλλά όχι την υλοποίηση.

Γενικά περί διασυνδέσεων (2/2)

- Μπορούμε να θεωρήσουμε μία διασύνδεση ως μία αμιγώς αφηρημένη (abstract) κλάση, με παρόμοια σύνταξη.
- Αν και μοιάζει με μία "abstract" κλάση, έχει σημαντικές διαφορές:
  - Οι διασυνδέσεις διαθέτουν μόνο δηλώσεις μεθόδων και όχι υλοποίησή τους.
  - Μία κλάση μπορεί να υλοποιήσει πολλές διασυνδέσεις, αλλά μπορεί να κληρονομήσει μόνο μία κλάση.
  - Οι διασυνδέσεις δεν αποτελούν μέρος μιας ιεραρχίας κλάσεων, έτσι κλάσεις που δεν έχουν κάποια σχέση μεταξύ τους μπορούν να υλοποιούν την ίδια διασύνδεση.
- Οποιαδήποτε κλάση υλοποιεί (implements) μία διασύνδεση έχει συμβατική υποχρέωση να παρέχει υλοποιήσεις όλων των μεθόδων της διασύνδεσης αλλιώς πρέπει να δηλωθεί ως "abstract".
- Εισάγει την δυνατότητα χρήσης πολλαπλής κληρονομικότητας και ενός κομψού τρόπου δήλωσης σταθερών.

Μορφή Δήλωσης

```
[public] interface  <όνομα> {
       <τύπος> <όνομα_μεθόδου>([<λίστα_παραμέτρων>]);
            :                :                 :
       <τύπος> <όνομα_μεταβλητής> = <τιμή>;
            :                :                 :
 }
```

- Εάν η διασύνδεση είναι "public" θα πρέπει να βρίσκεται σε αρχείο με το ίδιο όνομα και μπορεί να υλοποιηθεί σε οποιοδήποτε πακέτο
- Οι μεταβλητές πρέπει να αρχικοποιούνται και είναι (έμμεσα) static και final. Όλα τα μέλη είναι (έμμεσα) public.
- Η υλοποίηση γίνεται σε κάποια κλάση. Μπορούν να υπάρξουν διαφορετικές υλοποιήσεις της ίδιας διασύνδεσης.
- Μία κλάση μπορεί να υλοποιήσει πολλές διασυνδέσεις.

Μορφή Δήλωσης-Παράδειγμα

- Καθορισμός διασύνδεσης σε μία κλάση που παράγει μία σειρά αριθμών. Δηλώνεται "public" για να μπορεί να υλοποιηθεί από κλάσεις σε οποιοδήποτε πακέτο.

```
public interface  Series {
     int getNext(); // επιστροφή επόμενου αριθμού
     void reset(); // επανεκκίνηση
     void setStart (int x); // ορισμός τιμής έναρξης
}
```

Υλοποίηση

```
[public] class <όνομα> [extends <υπερκλάση>]
                    implements <διασύνδεση>
                    [, <διασύνδεση> [,...]] {
       // σώμα κλάσης
}
```

- Οι μέθοδοι που υλοποιούν τις μεθόδους μίας διασύνδεσης πρέπει να δηλώνονται ως "public".
- Εάν παραλείψουμε την υλοποίηση κάποιας μεθόδου τότε η κλάση πρέπει να δηλωθεί ως "abstract".

Υλοποίηση - Παράδειγμα_1 (1)

```
class ByTwos  implements  Series {
    int start;
    int val;
    ByTwos() { start=0;  val=0; }
    public int getNext() {
        val+=2;
        return val;
    }
    public void reset() { start=0;  val=0; }
    public void setStart(int x) { start=x;   val=x; }
}
```

Υλοποίηση - Παράδειγμα_1 (2)

```
class SeriesDemo {
    public static void main (String args[]) {
        ByTwos ob = new ByTwos();
        for (int i=0; i<5; i++)
            System.out.println("Next value is "+ob.getNext());
        System.out.println("\nResetting");
        ob.reset();
        for (int i=0; i<5; i++)
            System.out.println("Next value is "+ob.getNext());
        System.out.println("\nStarting at 100");
        ob.setStart(100);
        for (int i=0; i<5; i++)
            System.out.println("Next value is "+ob.getNext());
    }
}
```

Άλλη χρήση!!!

Ένας κομψός τρόπος παροχής σταθερών

Δήλωση σταθερών με τις μέχρι τώρα γνώσεις μας

```
class Constants {
  public final static int FEETPERMILE = 5280;
  public final static String PROMPT =
                                "Enter a number";
  public final static double PI = 3.141592;
}
```

- **Χρήση:**

```
feet = miles * Constants.FEETPERMILE;
```

Καλύτερος τρόπος δήλωσης σταθερών

```
interface Constants {
     int FPA_PERCENT = 19;
     String PROMPT = "Enter a price";
     double PI = 3.141592;
}
```

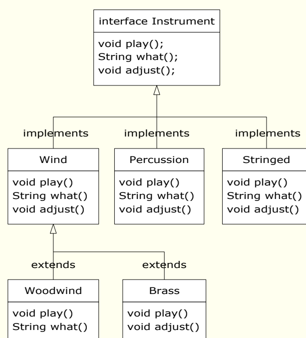
- Υλοποίηση της διασύνδεσης :

```
public class Box implements Constants
{
   // lots of code omitted
   cycleArea = radius * PI;
```

Άλλη χρήση!!!

Ομαδοποίηση διαφορετικού τύπου αντικειμένων σε μία δομή

Παράδειγμα μουσικών οργάνων (1)



Παράδειγμα μουσικών οργάνων (2)

```
interface Instrument {
   // Compile-time constant:
   int i = 5; // static & final
   // Cannot have method definitions:
   void play(); // Automatically public
   String what();
   void adjust();
}

class Wind implements Instrument {
   public void play() {
      System.out.println("Wind.play()");
   }
   public String what() { return "Wind"; }
   public void adjust() {}
}
```

Παράδειγμα μουσικών οργάνων (3)

```
class Percussion implements Instrument {
   public void play() {
      System.out.println("Percussion.play()");
   }
   public String what() { return "Percussion"; }
   public void adjust() {}
}

class Stringed implements Instrument {
   public void play() {
      System.out.println("Stringed.play()");
   }
   public String what() { return "Stringed"; }
   public void adjust() {}
}
```

Παράδειγμα μουσικών οργάνων (4)

```
class Brass extends Wind {
   public void play() {
      System.out.println("Brass.play()");
   }
   public void adjust() {
      System.out.println("Brass.adjust()");
   }
}

class Woodwind extends Wind {
   public void play() {
      System.out.println("Woodwind.play()");
   }
   public String what() { return "Woodwind"; }
}
```

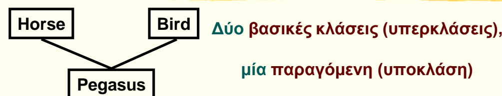
Παράδειγμα μουσικών οργάνων (5)

```
public class Music5 {
   static void tune(Instrument i) { i.play(); }
   static void tuneAll(Instrument[] e) {
      for(int i = 0; i < e.length; i++)  tune(e[i]);
   }
   public static void main(String[] args) {
      Instrument[] orchestra = new Instrument[5];
      int i = 0;
      // Upcasting during addition to the array:
      orchestra[i++] = new Wind();
      orchestra[i++] = new Percussion();
      orchestra[i++] = new Stringed();
      orchestra[i++] = new Brass();
      orchestra[i++] = new Woodwind();
      tuneAll(orchestra);
   }
}
```

Άλλη χρήση!!!

Πολλαπλή κληρονομικότητα

Περί πολλαπλής κληρονομικότητας



Δύο βασικές κλάσεις (υπερκλάσεις),

μία παραγόμενη (υποκλάση)

- Πολλαπλή κληρονομικότητα είναι η δυνατότητα δημιουργίας νέας κλάσης συνδυάζοντας τις ιδιότητες πολλών κλάσεων.
- Στην Java μπορούμε να κληρονομήσουμε μόνο μία κλάση.
  - Μη ύπαρξη πολλαπλής κληρονομικότητας
  - Η διασύνδεση μας προσφέρει έμμεσα την λειτουργικότητα της πολλαπλής κληρονομικότητας
- Η πολλαπλή κληρονομικότητα μπορεί να υλοποιηθεί κληρονομώντας μία μόνο κλάση και υλοποιώντας ένα πλήθος διασυνδέσεων.

Πολλαπλή Κληρονομικότητα

- Η πολλαπλή κληρονομικότητα οδηγεί σε πιθανή σύγχυση ονομάτων (πχ. Bird.doYourThing() - Horse.doYourThing())
- Η υλοποίηση μιας διασύνδεσης έχει την ίδια συμπεριφορά με την κληρονομικότητα μιας υπερκλάσης, αλλά
  - πρέπει να υλοποιήσει όλες τις μεθόδους
  - δεν υπάρχουν προβλήματα σύγχυσης ονομάτων
- Επειδή με την διασύνδεση δεν γίνεται δέσμευση μνήμης υπάρχει η δυνατότητα συνδυασμού πολλών διασυνδέσεων σε μία κλάση, χωρίς τα προβλήματα της πολλαπλής κληρονομικότητας.
- Κατά γενική ομολογία, τα καλά της πολλαπλής κληρονομικότητας δεν αξίζουν τα προβλήματα που προκαλεί

Πολλαπλή Κληρονομικότητα Παράδειγμα (1)

```
interface CanFight { void fight(); }
interface CanSwim { void swim(); }
interface CanFly { void fly(); }
class ActionCharacter {
    public void fight() {
        System.out.println("ActionCharacter.fight()");}
}
class Hero extends ActionCharacter
                implements CanFight, CanSwim, CanFly {
    public void swim() {
        System.out.println("Hero.swim()");
    }
    public void fly() {System.out.println("Hero.fly()");}
}
```

Πολλαπλή Κληρονομικότητα Παράδειγμα (2)

```
public class Adventure {
    static void s(Hero x) { x.fight(); }
    static void t(CanFight x) { x.fight(); }
    static void u(CanSwim x) { x.swim(); }
    static void v(CanFly x) { x.fly(); }
    static void w(ActionCharacter x) { x.fight(); }
    public static void main(String[] args) {
        Hero h = new Hero();
        s(h);
        t(h); // Treat it as a CanFight
        u(h); // Treat it as a CanSwim
        v(h); // Treat it as a CanFly
        w(h); // Treat it as an ActionCharacter
    }
}
```

Πολλαπλή Κληρονομικότητα Παράδειγμα (3)

Έξοδος:

```
ActionCharacter.fight()
ActionCharacter.fight()
Hero.swim()
Hero.fly()
ActionCharacter.fight()
```

Κληρονομικότητα διασύνδεσης Παράδειγμα (1)

```
interface Monster { void menace(); }
interface DangerousMonster extends Monster {
    void destroy();
}
interface Lethal { void kill(); }
class DragonZilla implements DangerousMonster {
    public void menace() {
        System.out.println("DragonZilla.menace()");
    }
    public void destroy() {
        System.out.println("DragonZilla.destroy()");
    }
}
```

Κληρονομικότητα διασύνδεσης Παράδειγμα (2)

```
interface Vampire  extends DangerousMonster,
                                          Lethal {
    void drinkBlood();
}
class HorrorShow {
    static void u(Monster b) { b.menace(); }
    static void v(DangerousMonster d) {
        d.menace();
        d.destroy();
    }
    public static void main(String[] args) {
        DragonZilla if2 = new DragonZilla();
        u(if2);
        v(if2);
    }
}
```

Κληρονομικότητα διασύνδεσης Παράδειγμα (3)

Έξοδος:

```
DragonZilla.menace()
DragonZilla.menace()
DragonZilla.destroy()
```

API Interfaces

Comparable
Cloneable

Interface Comparable

- Θέλουμε μία περιέχουσα δομή (container) η οποία μπορεί να μπορεί να έχει ταξινομημένα τα αντικείμενα που περιλαμβάνει.
- Για να μπουν δύο αντικείμενα σε κάποια σειρά θα πρέπει να υπάρχει μέθοδος σύγκρισής τους.
  - Ποια μέθοδο θα χρησιμοποιήσουμε για να συγκρίνουμε δύο αντικείμενα;
  - Θα παρέχουμε εμείς μία τέτοια μέθοδο;
- Όλα τα αντικείμενα που θα υπάρχουν σε αυτή την ταξινομημένη περιέχουσα δομή θα πρέπει υποχρεωτικά να παρέχουν μία τέτοια μέθοδο σύγκρισης. Επίσης αυτή η μέθοδος θα πρέπει να έχει μία γνωστή "υπογραφή".
- Πως το επιτυγχάνουμε;
- Με χρήση της διασύνδεσης Comparable

Interface java.lang.Comparable

- *Comparable* είναι μία προκαθορισμένη διασύνδεση του API της Java.

Από το API: "This interface imposes a total ordering on the objects of each class that implements it."

```
public interface Comparable {
     public int compareTo(Object o)
}
```

- Επιστρέφει: Ένα αρνητικό ακέραιο, μηδέν ή θετικό ακέραιο εάν το καλών αντικείμενο είναι μικρότερο από, ίσο με, ή μεγαλύτερο από το αντικείμενο που δίνεται ως παράμετρος.

Παράδειγμα ...

- Θέλουμε να αποθηκεύσουμε αντικείμενα "Box" τα οποία συγκρίνονται ΄με βάση τον όγκο τους.

Καθορισμός της κλάσης Box

```
public class Box {
  int length;
  int width;
  int height;

   public void setLength (int newLen)
   {
     length = newLen;
   }  // of setLength

   public int getLength ( ) {
      return (length);
   } // of getLength

   public void setWidth (int newWid)
   {
     width = newWid;
   }  // of setWidth

   public int getWidth ( ) {
     return (width);
   } // of getWidth

public void setHeight (int newHt)
{
    height = newHt;
}  // of setHeight

public int getHeight ( ) {
    return (height);
}  // of getHeight

public int getVolume ( ) {
  return ( getLength( ) *
           getWidth( )  *
           getHeight( ) );

  }  // of getVolume

}  // of class Box
```

Προσθήκη επιπλέον δυνατοτήτων

```
public class Box implements Comparable {
  // Όλα τα άλλα όπως προηγουμένως
  public int compareTo(Object o) {
    int retval = -1;
    if(o instanceOf Box) {
      retVal = getVolume() - ((Box)o).getVolume();
      if(retVal == 0)
        retVal = 0;
      else if(retVal > 0)
        retVal = 1;
      else
        retval = -1;
    } // if
    return retval;
  }
}
```

Εκτέλεση κώδικα ...

```
Box a = new Box(10, 20, 30);
Box b = new Box(2, 4, 6);
Box c = new Box(20, 10, 30);
System.out.println(a.compareTo(b)); ===> 1
System.out.println(a.compareTo(c)); ===> 0
System.out.println(b.compareTo(c)); ===> -1
System.out.println(a.compareTo("Hello")); ===> -1;
```

Εν κατακλείδι!

- Αν πούμε ότι κάποιο αντικείμενο πρέπει να είναι συγκρίσιμο ("Comparable") είναι σαν να λέμε:
  - Πρέπει να είναι αντικείμενο (δηλ. να έχει μία σχέση «*είναι-ένα*» με τη κλάση *java.lang.Object*. It pretty much will satisfy that by default.), ΚΑΙ
  - Πρέπει να έχει υλοποιημένη μία μέθοδο *compareTo*. (Απλά συμπεριλάβετε μία μέθοδο η οποία έχει την ίδια υπογραφή με αυτή που βρίσκεται στο interface java.lang.Comparable)
- Υλοποιώντας την διασύνδεση Comparable σε μία κλάση είναι σαν να την υποχρεώνουμε να παρέχει την μέθοδο*compareTo*.