Αντικειμενοστρεφής Προγραμματισμός

Παναγιώτης Αδαμίδης
adamidis@it.teithe.gr

"Abstract classes, δομητές και πολυμορφισμός, finalize, downcasting"

Αφηρημένες (abstract) κλάσεις

- Είναι πάντα βασικές κλάσεις μιας ιεραρχίας κλάσεων.
- Στόχος των αφηρημένων κλάσεων είναι να χειριστούμε ένα σύνολο κλάσεων μέσω μίας κοινής διασύνδεσης.
  - Στα προηγούμενα παραδείγματα η κλάση Instrument χρησιμοποιείται για να δημιουργήσει την κοινή διασύνδεση.
- Δηλώνουμε μία υπερκλάση που να καθορίζει μόνο την γενική μορφή η οποία θα χρησιμοποιείται από κοινού από όλες τις υποκλάσεις, αφήνοντας σε κάθε υποκλάση την δυνατότητα να συμπληρώνει από μόνη της τις επιμέρους λεπτομέρειες.
- Δεν είναι δυνατό να δημιουργηθούν αντικείμενα αφηρημένων κλάσεων.

Αφηρημένες (abstract) μέθοδοι 1/2

- Οι αφηρημένες μέθοδοι δεν είναι πλήρεις.
- Σύνταξη:

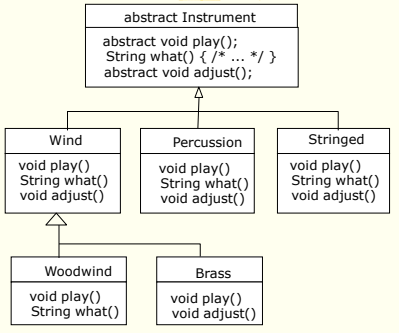
```
abstract <τύπος>
         <όνομα>([<λίστα_παραμέτρων>]);
```

- Εάν μία κλάση περιέχει τουλάχιστον μία αφηρημένη μέθοδο τότε η κλάση πρέπει να δηλωθεί ως "abstract".
- Ο τροποποιητής abstract δεν μπορεί να εφαρμοστεί σε μεθόδους static ή σε δομητές (constructors). *(Γιατί;)*

Αφηρημένες (abstract) μέθοδοι 2/2

- Εάν κάποια κλάση κληρονομεί μία αφηρημένη κλάση τότε σε αυτή θα πρέπει οπωσδήποτε να οριστούν πλήρως οι αφηρημένες μέθοδοι. Εάν δεν οριστούν τότε και η υποκλάση θα είναι αφηρημένη και φυσικά δεν μπορούμε να δημιουργήσουμε αντικείμενα της υποκλάσης.
- ***Προσοχή:*** Δεν είναι απαραίτητο να οριστούν ως "abstract" όλες οι μέθοδοι μίας "abstract" κλάσης. Αντιθέτως, είναι δυνατό μία abstract κλάση να μην έχει μεθόδους abstract.

Αφηρημένες κλάσεις και μέθοδοι Παράδειγμα



Abstract classes/methods Παράδειγμα (1)

```
abstract class Instrument {
   int i; // storage allocated for each
   public abstract void play();
   public String what() { return "Instrument"; }
   public abstract void adjust();
}
class Wind extends Instrument {
   public void play() {  System.out.println("Wind.play()"); }
   public String what() { return "Wind"; }
   public void adjust() {}
}
class Percussion extends Instrument {
   public void play() { System.out.println("Percussion.play()"); }
   public String what() { return "Percussion"; }
   public void adjust() {}
}
```

Abstract classes/methods Παράδειγμα (2)

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

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"; }
}
```

Abstract classes/methods Παράδειγμα (3)

```
public class Music4 {
    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;
        orchestra[i++] = new Wind();
        orchestra[i++] = new Percussion();
        orchestra[i++] = new Stringed();
        orchestra[i++] = new Brass();
        orchestra[i++] = new Woodwind();
        tuneAll(orchestra);
    }
}
```

Abstract classes/methods Παράδειγμα (4)

- ΕΞΟΔΟΣ:
  - Wind.play()
  - Percussion.play()
  - Stringed.play()
  - Brass.play()
  - Woodwind.play()

Δομητές και Πολυμορφισμός: Σειρά κλήσης

- Η μνήμη που ανατίθεται στο αντικείμενο αρχικοποιείται στο δυαδικό μηδέν.
- Κλήση του δομητή της βασικής κλάσης. Το βήμα επαναλαμβάνεται μέχρι την ρίζα της ιεραρχίας.
- Αρχικοποίηση μελών με την σειρά που δηλώνονται
- Κλήση του δομητή της παραγόμενης κλάσης

Δομητές και Πολυμορφισμός Παράδειγμα (1)

```
class Meal {
    Meal() { System.out.println("Meal()"); }
}
class Bread {
    Bread() { System.out.println("Bread()"); }
}
class Cheese {
    Cheese() { System.out.println("Cheese()"); }
}
class Lettuce {
    Lettuce() { System.out.println("Lettuce()"); }
}
class Lunch extends Meal {
    Lunch() { System.out.println("Lunch()");}
}
```

Δομητές και Πολυμορφισμός Παράδειγμα (2)

```
class PortableLunch extends Lunch {
    PortableLunch() {
        System.out.println("PortableLunch()");
  }
}
class Sandwich extends PortableLunch {
    Bread b = new Bread();
    Cheese c = new Cheese();
    Lettuce l = new Lettuce();
    Sandwich() {
        System.out.println("Sandwich()");
    }
    public static void main(String[] args) {
        new Sandwich();
    }
}
```

Δομητές και Πολυμορφισμός Παράδειγμα (3)

- ΕΞΟΔΟΣ:

Meal()
Lunch()
PortableLunch()
Bread()
Cheese()
Lettuce()
Sandwich()

Κληρονομικότητα και finalize() Παράδειγμα (1)

- Κατά την υπέρβαση της finalize() πρέπει να καλέσουμε την finalize() της βασικής κλάσης

```
class DoBaseFinalization {
   public static boolean flag = false;
}
class Characteristic {
   String s;
   Characteristic(String c) {
      s = c;
      System.out.println("Creating Characteristic " + s);
   }
   protected void finalize() {
      System.out.println("finalizing Characteristic " + s);
   }
}
```

Κληρονομικότητα και finalize() Παράδειγμα (1)

- Κατά την υπέρβαση της finalize() πρέπει να καλέσουμε την finalize() της βασικής κλάσης

```
class DoBaseFinalization {
   public static boolean flag = false;
}
class Characteristic {
   String s;
   Characteristic(String c) {
      s = c;
      System.out.println("Creating Characteristic " + s);
   }
   protected void finalize() {
      System.out.println("finalizing Characteristic " + s);
   }
}
```

Κληρονομικότητα και finalize() Παράδειγμα (2)

```
class LivingCreature {
   Characteristic p = new Characteristic("is alive");
   LivingCreature() {System.out.println("LivingCreature()"); }
   protected void finalize() throws Throwable {
         System.out.println("LivingCreature finalize");
         // Call base-class version LAST!
         if (DoBaseFinalization.flag)  super.finalize();
   }
}
class Animal extends LivingCreature {
   Characteristic p = new Characteristic("has heart");
   Animal() { System.out.println("Animal()"); }
   protected void finalize() throws Throwable {
         System.out.println("Animal finalize");
         if (DoBaseFinalization.flag) super.finalize();
   }
}
```

Κληρονομικότητα και finalize() Παράδειγμα (3)

```
class Amphibian extends Animal {
   Characteristic p =
         new Characteristic("can live in water");
   Amphibian() {
         System.out.println("Amphibian()");
   }
   protected void finalize() throws Throwable {
         System.out.println("Amphibian finalize");
         if (DoBaseFinalization.flag)
               super.finalize();
   }
}
```

Κληρονομικότητα και finalize() Παράδειγμα (4)

```
public class Frog extends Amphibian {
   Frog() { System.out.println("Frog()"); }
   protected void finalize() throws Throwable {
         System.out.println("Frog finalize");
         if (DoBaseFinalization.flag) super.finalize();
   }
   public static void main(String[] args) {
         if (args.length != 0 && args[0].equals("finalize"))
               DoBaseFinalization.flag = true;
         else
               System.out.println("Not finalizing bases");
         new Frog(); // Instantly becomes garbage
         System.out.println("Bye!");
         // Force finalizers to be called:
         System.gc();
   }
}
```

Κληρονομικότητα και finalize()
Παράδειγμα - Έξοδος

➢java Frog

```
Not finalizing bases
Creating Characteristic is alive
LivingCreature()
Creating Characteristic has heart
Animal()
Creating Characteristic can live in water
Amphibian()
Frog()
Bye!
Frog finalize
finalizing Characteristic is alive
finalizing Characteristic has heart
finalizing Characteristic can live in water
```

Κληρονομικότητα και finalize()
Παράδειγμα - Έξοδος

➢java Frog finalize

```
Creating Characteristic is alive
LivingCreature()
Creating Characteristic has heart
Animal()
Creating Characteristic can live in water
Amphibian()
Frog()
Bye!
Frog finalize
Amphibian finalize
Animal finalize
LivingCreature finalize
finalizing Characteristic is alive
finalizing Characteristic has heart
finalizing Characteristic can live in water
```

Κλήση πολυμορφικών μεθόδων από δομητές
Παράδειγμα (1)

```
abstract class Glyph {
    abstract void draw();
    Glyph() {
        System.out.println("Glyph() before draw()");
        draw();
        System.out.println("Glyph() after draw()");
    }
}
class RoundGlyph extends Glyph {
    int radius = 1;
    RoundGlyph(int r) {
        radius = r;
        System.out.println(
          "RoundGlyph.RoundGlyph(), radius = " + radius);
    }
```

Κλήση πολυμορφικών μεθόδων από δομητές
Παράδειγμα (2)

```
    void draw() {
        System.out.println(
            "RoundGlyph.draw(), radius = " + radius);
    }
}

public class PolyConstructors {
    public static void main(String[] args) {
        new RoundGlyph(5);
    }
}
```

➢ java PolyConstructors

```
Glyph() before draw()
RoundGlyph.draw(), radius = 0
Glyph() after draw()
RoundGlyph.RoundGlyph(), radius = 5
```

Δομητές και Πολυμορφισμός:
Συμβουλές

➢Κατά τον πολυμορφισμό:

- Οι δομητές να επιφορτιστούν μόνο με την ικανοποιητική αρχικοποίηση των αντικειμένων και όχι με περισσότερες λειτουργίες.
- Αποφύγετε την κλήση μεθόδων από δομητές. Οι μόνες ασφαλείς μέθοδοι είναι οι μέθοδοι final και οι private (εμμέσως final)

Downcasting (1)

➢ Η βασική κλάση μπορεί να δεχθεί μηνύματα από κάποια υποκλάση της αφού έχουν την ίδια βασική διασύνδεση/ συμπεριφορά. Έτσι το "upcasting" είναι μία απλή υπόθεση.

➢ Η υποκλάση έχει επιπλέον στοιχεία τα οποία απαιτούν την υλοποίηση επιπλέον μεθόδων. Από την βασική κλάση δεν έχουμε πρόσβαση στις επιπλέον μεθόδους. Έτσι μετά το "upcast" δεν μπορούμε να καλέσουμε τις νέες μεθόδους. Πρέπει να βρούμε τον ακριβή τύπο του αντικειμένου για να μπορέσουμε να χρησιμοποιήσουμε τις επιπλέον μεθόδους. Η μετακίνηση προς τα κάτω της ιεραρχίας των κλάσεων (η μετατροπή σε αντικείμενο τύπου υποκλάσης) ονομάζεται "downcast".

Downcasting (2)

- Το "upcast" είναι ασφαλές. Για το "downcast" ο compiler απαιτεί να δηλωθεί ο τύπος μετατροπής. Για παράδειγμα ο κύκλος είναι σχήμα, αλλά ένα σχήμα μπορεί να είναι κύκλος, τρίγωνο ή κάτι άλλο.
- Η Java ελέγχει όλες τις μετατροπές τύπων κατά την εκτέλεση του προγράμματος (Run-time type identification - RTTI) και επιβεβαιώνει την σωστή μετατροπή. Εάν ο τύπος είναι λάθος έχουμε "ClassCastException".

Downcasting - Παράδειγμα

```
class Useful {
    public void f() {System.out.println("Useful.f()");}
    public void g() {System.out.println("Useful.g()");}
}
class MoreUseful extends Useful {
    public void f() {System.out.println("MoreUseful.f()");}
    public void g() {System.out.println("MoreUseful.g()");}
    public void u() {System.out.println("MoreUseful.u()");}
    public void v() {System.out.println("MoreUseful.v()");}
    public void w() {System.out.println("MoreUseful.w()");}
}
public class RTTI {
    public static void main(String[] args) {
        Useful[] x = {  new Useful(),  new MoreUseful() };
        x[0].f();
        x[1].g();
        //! x[1].u();
        if (x[1] instanceof MoreUseful) ((MoreUseful)x[1]).u();
        ((MoreUseful)x[0]).u(); // Exception thrown
    }
}
```

Downcasting
Παράδειγμα (instanceof)

```
class Useful {
    public void f() {System.out.println("Useful.f()");}
    public void g() {System.out.println("Useful.g()");}
}
class MoreUseful extends Useful {
    public void f() {System.out.println("MoreUseful.f()");}
    public void g() {System.out.println("MoreUseful.g()");}
    public void u() {System.out.println("MoreUseful.u()");}
    public void v() {System.out.println("MoreUseful.v()");}
    public void w() {System.out.println("MoreUseful.w()");}
}
public class RTTI {
    public static void main(String[] args) {
        Useful[] x = {  new Useful(),  new MoreUseful() };
        x[0].f();
        x[1].g();
        //! x[1].u();
        if (x[1] instanceof MoreUseful) ((MoreUseful)x[1]).u();
        if (x[0] instanceof MoreUseful) ((MoreUseful)x[0]).u();
    }
}
```