Αντικειμενοστρεφής Προγραμματισμός

Παναγιώτης Αδαμίδης
adamidis@it.teithe.gr

"Αρχικοποίηση, More Upcasting, Binding"

Αρχικοποίηση

- Ο μεταγλωττισμένος κώδικας για κάθε κλάση υπάρχει σε διαφορετικό αρχείο.
- Το αρχείο φορτώνεται όταν χρησιμοποιείται για πρώτη φορά. Αυτό συμβαίνει όταν δημιουργείται το πρώτο αντικείμενο της κλάσης ή κατά την πρόσβαση σε ένα "static" μέλος της κλάσης.
- Εάν υπάρχει υπερκλάση τότε πρώτα θα αρχικοποιηθούν τα static μέλη αυτής.
- Πρώτα αρχικοποιούνται οι βασικοί τύποι στις προκαθορισμένες τιμές τους και μετά οι αναφορές σε null. Μετά καλείται ο δομητής της βασικής κλάσης (αυτόματα ή με την χρήση του super) και ακολουθούν οι μεταβλητές του αντικειμένου (instance variables).
- Τελειώνει με την εκτέλεση του υπόλοιπου κώδικα του δομητή

Αρχικοποίηση - Παράδειγμα

```
class Insect {
    int i = 9;
    int j;
    Insect() {
        prt("i = " + i + ", j = " + j);
        j = 39;
    }
    static int x1 = prt("static Insect.x1 initialized");
    static int prt(String s) {
        System.out.println(s);
        return 47;
    }
}
```

συνεχίζεται

Αρχικοποίηση - Παράδειγμα

```
public class Beetle extends Insect {
   int k = prt("Beetle.k initialized");
   Beetle() {
      prt("k = " + k);
      prt("j = " + j);
   }
   static int x2 = prt("static Beetle.x2 initialized");
   public static void main(String[] args) {
      prt("Beetle constructor");
      Beetle b = new Beetle();
   }
}
```

Αρχικοποίηση - Παράδειγμα

ΕΞΟΔΟΣ:

```
static Insect.x1 initialized
static Beetle.x2 initialized
Beetle constructor
i = 9, j = 0
Beetle.k initialized
k = 47
j = 39
```

Upcasting - Επανεξέταση

```
class Note {
   private int value;
   private Note(int val) { value = val; }
   public static final Note
      MIDDLE_C = new Note(0),
      C_SHARP  = new Note(1),
      B_FLAT   = new Note(2);
}

class Instrument {
   public void play(Note n) {
      System.out.println("Instrument.play()");
   }
}
```

συνεχίζεται

Upcasting - Επανεξέταση

```
class Wind extends Instrument {
   // Redefine interface method:
   public void play(Note n) {
      System.out.println("Wind.play()");
   }
}

public class Music {
   public static void tune(Instrument i) {
      // ...
      i.play(Note.MIDDLE_C);
   }
   public static void main(String[] args) {
      Wind flute = new Wind();
      tune(flute); // Upcasting
   }
}
```

No Upcasting; More programming (1)

```
class Note {
   private int value;
   private Note(int val) { value = val; }
   public static final Note  MIDDLE_C = new Note(0),
         C_SHARP = new Note(1),  B_FLAT = new Note(2);
}
class Instrument {
   public void play(Note n) { System.out.println("Instrument.play()");}
}
class Wind extends Instrument {
   public void play(Note n) { System.out.println("Wind.play()"); }
}
class Stringed extends Instrument {
  public void play(Note n) { System.out.println("Stringed.play()"); }
}
```

No Upcasting; More programming (2)

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

public class Music2 {
  public static void tune(Wind i) {  i.play(Note.MIDDLE_C);  }
  public static void tune(Stringed i) {  i.play(Note.MIDDLE_C); }
  public static void tune(Brass i) {  i.play(Note.MIDDLE_C);  }
  public static void main(String[] args) {
    Wind flute = new Wind();
    Stringed violin = new Stringed();
    Brass frenchHorn = new Brass();
    tune(flute); // Upcasting ??
    tune(violin);
    tune(frenchHorn);
  }
}
```

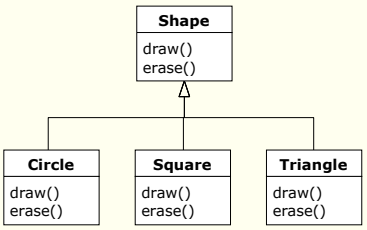
No Upcasting; More programming (3)

- ΕΞΟΔΟΣ:
  Wind.play()
  Stringed.play()
  Brass.play()
- Προβλήματα
  - Μέθοδοι για κάθε νέο όργανο (νέα κλάση)
  - Εάν ξεχάσετε να υπερφορτώσετε κάποια μέθοδο ο compiler δεν θα δώσει μήνυμα λάθους.
- Στόχος: κώδικας προσανατολισμένος στην βασική κλάση (όρισμα της βασικής κλάσης)

Σύνδεση μεθόδων (Binding)

- Early binding: από τον compiler ή τον linker
- Late/dynamic/run-time binding: σύνδεση κατά την εκτέλεση βασισμένη στον τύπο του αντικειμένου. Αν και διαφέρει από γλώσσα σε γλώσσα, τα αντικείμενα πρέπει να συνοδεύονται από κάποια πληροφορία.
- Όλες οι συνδέσεις μεθόδων στην Java γίνονται κατά την εκτέλεση εκτός αν η μέθοδος έχει δηλωθεί "final".

Παράδειγμα δυναμικής σύνδεσης



- Shape s = new Circle(); // σωστό ή λάθος
- s.draw();

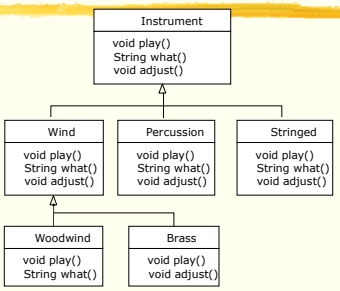
Παράδειγμα δυναμικής σύνδεσης (1)

```
class Shape {
    void draw() {}
    void erase() {}
}
class Circle extends Shape {
    void draw() { System.out.println("Circle.draw()"); }
    void erase() { System.out.println("Circle.erase()"); }
}
class Square extends Shape {
    void draw() { System.out.println("Square.draw()"); }
    void erase() { System.out.println("Square.erase()"); }
}
class Triangle extends Shape {
    void draw() { System.out.println("Triangle.draw()"); }
    void erase() { System.out.println("Triangle.erase()"); }
}
```

Παράδειγμα δυναμικής σύνδεσης (2)

```
public class Shapes {
    public static Shape randShape() {
        switch((int)(Math.random() * 3)) {
            default:
            case 0: return new Circle();
            case 1: return new Square();
            case 2: return new Triangle();
        }
    }
    public static void main(String[] args) {
        Shape[] s = new Shape[9];
        // Fill up the array with shapes:
        for(int i = 0; i < s.length; i++)
            s[i] = randShape();
        // Make polymorphic method calls:
        for(int i = 0; i < s.length; i++)
            s[i].draw();
    }
}
```

Επεκτασιμότητα



- Επεκτασιμότητα: Δυνατότητα πρόσθεσης νέων λειτουργιών κληρονομώντας νέους τύπους δεδομένων από μία κοινή βασική κλάση. Οι μέθοδοι της βασικής κλάσης δεν είναι ανάγκη να μεταβληθούν για να διευθετηθούν οι νέες κλάσεις

Επεκτασιμότητα - Παράδειγμα (1)

```
import java.util.*;

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

Επεκτασιμότητα - 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"; }
}
```

Επεκτασιμότητα - Παράδειγμα (3)

```
public class Music3 {
    // New types added to the system still work right:
    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);
    }
```

Επεκτασιμότητα - Παράδειγμα (4)

- ΕΞΟΔΟΣ:

```
Wind.play()
Percussion.play()
Stringed.play()
Brass.play()
Woodwind.play()
```