Ingineria Sistemelor de Programare

Java – Concepte OOP

mihai.hulea@aut.utcluj.ro

Cuprins

- Enum
- Clase abstracte
- Interfete
- Clase interne

Tipul enum

- Introdus in Java 1.5
- Simplifica declarare constantelor
- Nu poate fi instantiat

```
public enum Currency {
    PENNY(1), NICKLE(5), DIME(10),
QUARTER(25);
    private int x;

    private Currency(int x) {
        this.x = x;
    }
}
```

```
public enum Day {
   SUNDAY, MONDAY, TUESDAY, WEDNESDAY,
   THURSDAY, FRIDAY, SATURDAY;
}
```

Exemplu – Tipul enum

```
package control.isp.tests;

enum Day {
    MONDAY,SUNDAY;
}


public class Main {
    static void checkDay(Day d){
        switch(d){
            case MONDAY:{
                System.out.println(d.name()+" it's working day!");
                break;
            }
            case SUNDAY:{
                System.out.println(d.name()+" it's free day!");
                break;
            }
        }
    }

    public static void main(String[] args) {
        checkDay(Day.MONDAY);
        checkDay(Day.SUNDAY);
    }
}
```

Tipul enum

- Nu poate fi instantiat
- Constructorul trebuie sa aiba acces *default* sau *private* si nu poate fi accesat in mod direct
- Pentru tipul enum se poate utiliza '==' pentru a compar doua tipuri enum
- Metoda values() este adaugata automat la tipul enum => returneaza vector cu constantele definite
- Metoda valueOf(String v) este adaugata automat al tipul enum => returneaza constanta cu valoarea data ca argument

Clase abstracte si interfete

- Clase abstracte
  - Nu pot fi instantiate
  - Contin metode fara implementare
- Interfete
  - Contin doar metode fara implementare
  - Specifica un contract intre interfata si clasa care o implementeaza
  - O clasa poate implementa mai multe interfete => simuleaza mostenirea multipla

Exemplu – Clase abstracte

```
abstract class GraphicObject {
                    int x, y;
                    void moveTo(int newX, int newY) { //metoda normala
                      System.out.println("Move graphic object to position"+x+":"+y);
                    }
                    abstract void draw();                          //metoda abstracta
}

class Circle extends GraphicObject {
                    void draw() {
                       System.out.println("Draw circle");
                    }
          }
```

Exemplu - Interfete

```
interface Instrument {
    void play();
}

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

```
public class Muzica { //clasa principala
    static void play(Instrument i) {
        i.play();
    }
    static void playAll(Instrument[] e) {
        for (int i = 0; i < e.length; i++) {
            play(e[i]);
        }
    }
    public static void main(String[] args) {
        Instrument[] orchestra = new
Instrument[2];
        int i = 0;
        orchestra[i++] = new Pian();
        orchestra[i++] = new Vioara();
        playAll(orchestra);
    }
}
```

Interfete functionale

- Concept introdus in Java 1.8
- Se utilizeaza adnotatia @FunctinalInterface dar nu este obligatorie
- Contin o singura metoda
- Faciliteaza implementarea expresiilor lambda

Exemplu – Interfata functionala

```
@FunctionalInterface
interface Shape{
   public void draw();
}

class ShapeImpl implements Shape{

   @Override
   public void draw() {
      System.out.println("Drawing shape!");
   }

}
```

Interfete si clase abstracte cu implementari default

- Implemantera unui comportament implicit pentru metode
- Exemplu de utilizare: aduagare de metoda intr-o interfata fara a modifica implementarile existente

Exemplu – Implementare metoda default

```
interface X{
    void m1();
    default void m2(){
        System.out.println("Do something!");
    }
}

interface Y extends X{
    default void m2(){
        System.out.println("Do something else
1!");
    }
}
```

Metode statice in interfete

- Introduse in Java 8
- Faciliteaza eliminarea claselor utilitare
- Sunt utilizate pentru implementarea metodelor utilitare

- Demo

Exemplu – Metode statice in interfete

```
interface Instrument{
    void play();
    static public Nota GenereazaNota(int nota){
        return new Nota(nota);
    }
}

class Pian implements Instrument{
    @Override
    public void play() {
        System.out.println("Pianul canta nota
"+Instrument.GenereazaNota(1));
    }
}

class Nota{
  ....
}
```

Clase interne

- Statice
- De instanta
- Locale
- Anonime

```
class OuterClass {
    …
    class NestedClass {
        …
    }
}
```

Exemplu – Clasa interna

```
//clasa interna statica
class Outer {
    static class Nested {
    }
}

Outer.Nested instance = new Outer.Nested();
```

```
//clasa interna de instanta
public class Outer {
    private String text = "I am private!";
    public class Inner {
        public void printText() {
            System.out.println(text);
        }
    }
}

Outer x= new Outer();
Outer.Inner inner = x.new Inner();
inner.printText();
```

Exemplu – Clasa interna

```
//clasa locala
class Outer {
    public void printText() {
        class Local {
        }
        Local local = new Local();
    }
}
```

```
//ascunderea membrului clasei externe
public class Outer {
    private String text = "I am Outer private!";
    public class Inner {
        private String text = "I am Inner private";
        public void printText() {
            System.out.println(text);
            System.out.println(Outer.this.text);
        }
    }
}
```

Exemplu – Clasa interna anonima

```
//clase interne anonime
public interface MyInterface{
  public void doIt();
}

....
MyInterface instance = new MyInterface(){
    public void doIt() {
        System.out.println("Anonymous class
doIt()");
    }
};

instance.doIt();
```