Programare avansata pe obiecte – laborator 3

Diana Maftei
diana.maftei@endava.com
https://github.com/DianaMaftei/pao_lab_2022

Mostenirea

- Modeleaza relatia “is-a”, indica faptul ca o clasa este derivata dintr-o clasa de baza
- Mai exista si relatii de tip “has-a”, indica faptul ca o clasa container are o referinta la alta clasa continuta in ea

Polimorfism

Se refera la proprietatea obiectelor de a avea mai multe forme.

Limbajul Java permite definerea a doua tipuri de polimorfism:

1. Supraincarcare (tipul parametric) - overloading - putem define o metoda cu acelasi nume dar care difera prin nr/tipul de parametri
    a. Acest tip de polimorfism se mai numeste si compile time polymorphism sau static binding sau early binding
2. Supradefinire/suprascriere – overriding - clasele derivate definesc o metoda din clasa de baza (cu aceeasi semnatura!) careia ii schimba comportamentul
    a. Acest tip de polimorfism se mai numeste si runtime polymorphism sau dynamic binding sau late binding
    b. Nu putem supradefini o metoda static/final sau privata
    c. Metoda suprascrisa nu poate un avea un modificator de access care sa ii permita o vizibilitate mai mica. De ex daca in clasa parinte avem o metoda protected in copil aceasta poate deveni publica, nu privata.

Abstract

- Cuvantul cheie **abstract**
- O clasa care contine cel putin o metoda abstracta trebuie sa fie abstracta
- O clasa abstracta nu poate fi instantiata
- Orice clasa care va extinde clasa abstracta trebuie sa ofere implementari pentru metodele abstracte

Imutabilitate

- Odata ce un obiect este creat, nu ii mai putem schimba continutul
- Toate clasele wrapper sunt imutabile
- Reguli de urmat pentru a obtine asta:
    1. Clasa trebuie sa fie final, pentru a nu mai fi extinsa
    2. Membrii clasei sunt final (initializati doar in constructor)
    3. Constructor cu parametrii folosit pentru initializarea membrilor clasei

4. Getteri pentru toate field-urile
5. Fara setter (ca sa nu oferim posibilitatea schimbarii field-urilor)
6. In orice metoda a clasei care modifica starea obiectelor, trebuie returnata o noua instanta a acestora (inclusiv in constructor)

String

- Immutable
- De fiecare data cand facem o schimbare un alt String e creat
- StringBuilder si StringBuffer (alternativa thread safe)

Exercitii – predict the output

```
class Test {
    protected int x, y;
}

class Main {
    public static void main(String args[]) {
        Test t = new Test();
        System.out.println(t.x + " " + t.y);
    }
}
```

```
class Test {
    public static void main(String[] args) {
        for(int i = 0; 1; i++) {
            System.out.println("Hello");
            break;
        }
    }
}
```

```
class Main {
    public static void main(String args[]) {
        System.out.println(fun());
    }
    int fun() {
        return 20;
    }
}
```

```
class Test {
   public static void main(String args[]) {
       System.out.println(fun());
   }
   static int fun() {
       static int x= 0;
       return ++x;
   }
}
```

```
package ro.unibuc.pao;

class Point {
```

```
    protected int x, y;

    public Point(int x, int y)
    {
        this.x = x;
        this.y = y;
    }
}

public class Main {
    public static void main(String args[])
    {
        Point p = new Point();
        System.out.println("x = " + p.x + ", y = " + p.y);
    }
}
```

```
class Base {
    protected void foo() {}
}
class Derived extends Base {
    void foo() {}
}
public class Example {
    public static void main(String args[]) {
        Derived d = new Derived(); d.foo();

    }
}
```

```
class Complex {
    private   double re, im;
    public String toString() {
        return "(" + re + " + " + im + "i)";
    }
    Complex(Complex c) {
        re = c.re;
        im = c.im;
    }
}

public class Example {
    public static void main(String[] args) {
        Complex c1 = new Complex();
        Complex c2 = new Complex(c1);
        System.out.println(c2);
    }
}
```

```
public class Calculator
{
    int  num = 100;
    public void  calc(int num) { this.num = num * 10;  }
    public void  printNum()    { System.out.println(num); }

    public static void main(String[] args)
    {
        Calculator obj = new Calculator();
```

```
        obj.calc(2);
        obj.printNum();
    }
}
```

```
public class MyStuff
{
    String name;

    MyStuff(String n) {  name = n;  }

    public static void main(String[] args)
    {
        MyStuff m1 = new MyStuff("guitar");
        MyStuff m2 = new MyStuff("tv");
        System.out.println(m2.equals(m1));
    }

    @Override
    public boolean equals(Object obj)
    {
        MyStuff m = (MyStuff) obj;
        if (m.name != null)  { return true;  }
        return false;
    }
}
```

```
class Alpha
{
    public String type = "a ";
    public Alpha() {  System.out.print("alpha "); }
}

public class Beta extends Alpha
{
    public Beta()  {  System.out.print("beta ");  }

    void go()
    {
        type = "b ";
        System.out.print(this.type + super.type);
    }

    public static void main(String[] args)
    {
        new Beta().go();
    }
}
```

```
class Gfg
{
    static int num;
    static String mystr;

    // constructor
    Gfg()
    {
```

```
        num = 100;
        mystr = "Constructor";
    }

    // First Static block
    static
    {
        System.out.println("Static Block 1");
        num = 68;
        mystr = "Block1";
    }

    // Second static block
    static
    {
        System.out.println("Static Block 2");
        num = 98;
        mystr = "Block2";
    }

    public static void main(String args[])
    {
        Gfg a = new Gfg();
        System.out.println("Value of num = " + a.num);
        System.out.println("Value of mystr = " + a.mystr);
    }
}
```

Exercitii

1. Proiectati o clasa CandyBox, care va continue campurile: flavor (String), origin (String) cu modificatorii de access corespunzatori. Clasa va avea de asemenea:
    a. Constructor fara parametri
    b. Constructor care va initializa toate campurile
    c. O metoda abstracta getVolume()
    d. Suprascrierea metodei toString()

    Din ea derivati clasele Merci, Lindt, Milka. Pentru un design diferit, cutiile au diverse forme:
    a. Lindt va continue length, width si height
    b. Milka va fi un cilindru cu campurile radius si height
    c. Merci va fi un cub si va contine campul length

    Clasele vor avea de asemenea:

    a. Constructor fara parametri
    b. Constructor care initializeaza membrii claselor
    c. Suprascrierea metodei getVolume()
    d. Suprascrierea metodei toString() care sa returneze un mesaj de genul: “The ” + origin + “ “ + flavor + “ has volume “ + volume.

Verificati egalitatea acestor obiecte create (din fiecare tip) si adaugati metoda equals() dupa cum este nevoie. Justificati criteriul de echivalenta ales.

Creati o clasa CandyBag care va contine un array cu cutii din fiecare tip.

Creati clasa Area care va continue un obiect de tip CandyBag, un camp number (int) si unul street (String). Clasa va contine:

a. Constructor fara parametri
b. Constructor care initializeaza atributele
c. O metoda printAddress() care va afisa adresa completa si continutul CandyBag (parcurgeti array-ul si apelati metoda toString() pentru elementele sale)

2. Se citeste un sir de caractere de la tastatura, verificati daca este un palindrom.
3. Scrieti un program care verifica daca doua siruri de caractere sunt anagrame. (ex: ramo, mora si roma sunt anagrame)
4. Sa se implementeze o clasa PasswordMaker ce genereaza o parola pornind de la datele unei persoane. Aceasta clasa o sa contina urmatoarele:
   a. o constanta MAGIC_NUMBER, care ia orice valoare doriti
   b. un String constant MAGIC_STRING, lung de minim 20 caractere, generat random (puteti crea o metoda care da un caracter random)
   c. un constructor care primeste un String, numit name
   d. o metoda getPassword() care va returna parola
      Parola se construieste concatenand urmatoarele:
      - un sir random de lungime MAGIC_NUMBER
      - 10 caractere din MAGIC_STRING
      - lungimea atributului name ca si String
      - un numar intreg generat random din intervalul [0,100] folosind clasa Random (https://docs.oracle.com/javase/8/docs/api/java/util/Random.html)

Modificati clasa PasswordMaker astfel incat sa respecte conceptul de Singleton.