Programare orientată pe obiecte

Breviar - Laboratorul 6

Mihai Nan
mihai.nan.cti@gmail.com

Facultatea de Automatică şi Calculatoare
Universitatea Politehnica din Bucureşti
Anul universitar 2015 - 2016

1 Clase abstracte si interfete

1.1 Clase si metode abstracte

Uneori, in proiectarea unei aplicatii, este necesar sa reprezentam cu ajutorul claselor concepte abstracte care sa nu poata fi instantiate si care sa foloseasca doar la dezvoltarea ulterioara a unor clase ce descriu obiecte concrete. De exemplu, in pachetul ***java.lang*** exista clasa abstracta ***Number*** care modeleaza conceptul generic de „numar". Intr-un program nu avem insa nevoie de numere generice, ci de numere cu un anumit tip: intregi, reale, etc. Clasa ***Number*** serveste ca superclasa pentru clasele concrete ***Byte***, ***Double***, ***Float***, ***Integer***, ***Long*** si ***Short***, ce implementeaza obiecte pentru descrierea numerelor de un anumit tip.

⚠ IMPORTANT !

⚠ O clasa abstracta nu poate fi instantiata. Astfel, vom uzita, in schimb, subclase ale sale care nu sunt abstracte.

Declaram o clasa, ca fiind abstracta, folosind cuvantul cheie ***abstract***. O clasa abstracta poate avea modificatorul ***public***, accesul implicit fiind la nivel de pachet, dar nu poate specifica modificatorul ***final***, combinatia ***abstract final*** fiind semnalata ca **eroare de compilare**.

O clasa abstracta poate contine aceleasi elemente membre ca o clasa obisnuita, la care se adauga declaratii de metode abstracte - fara o implementare efectiva.

Cod sursa Java

```
1  public abstract class Patrulater {
2      class Point{
3          int x, y;
4
5          public Point(int x, int y) {
6              this.x = x;
7              this.y = y;
8          }
9      }
10     Point p1, p2, p3, p4;
11     //Metode abstracte
12     abstract int calculPerimetru();
13     abstract double calculArie();
14 }
```

Spre deosebire de clasele obisnuite, care trebuie sa furnizeze implementari pentru toate metodele declarate, o clasa abstracta poate contine metode fara o implementare efectiva. Astfel de metode se numesc **metode abstracte** si pot aparea doar in clase abstracte. De asemenea, in declararea unei metode abstracte de foloseste cuvantul-cheie ***abstract***.

Paralelogramul, dreptunghiul, patratul si rombul sunt figuri geometrice ce pot fi caracterizate prin patru puncte din plan. Pentru fiecare astfel de figura geometrica avem o formula prin care ii putem determina aria, stiind doar coordonatele celor patru puncte ce definesc figura sau si alte date suplimentare. Prin urmare, clasele care modeleaza aceste figuri geometrice ar putea fi derivate dintr-o clasa abstracta ***Patrulater***. Mai mult, intrucat pentru fiecare astfel de clasa avem o formula diferita pentru calculul ariei, este normal ca ***Patrulater*** sa contina o metoda ***calculArie***. Insa, se pune problema implementarii pe care o va avea aceasta metoda in clasa de baza, deoarece fiecare patrulater concret are un mod propriu de calcul al ariei.

Intr-o astfel de situatia, varianta corecta de implementare este cea in care metoda ***calculArie*** este declarata ca fiind abstracta, ea neavand o implementare in clasa de baza ***Patrulater***. Implicit, aceasta clasa va deveni una abstracta, deoarece contine o metoda abstracta.

Observatie

O clasa poate fi abstracta chiar daca aceasta nu contine o metoda abstracta. Daca o clasa contine cel putin o metoda abstracta, atunci respectiva clasa va trebui si ea declarata ca fiind abstracta, in caz contrar, semnalandu-se o eroare la compilare.
O subclasa a unei clase abstracte trebuie sa implementeze toate metodele abstracte ale superclasei sale, in caz contrar, fiind necesar ca si subclasa sa fie abstracta. Acest fapt este normal, fiindca nu pot fi instantiate clase care contin metode neimplementate!

1.2 Interfete

Interfetele duc conceptul de clasa abstracta cu un pas inainte, prin eliminarea oricaror implementari de metode, punand in practica unul din conceptele programarii orientate pe obiecte: separarea modelului unui obiect (interfata) de implementarea sa. Asadar, o interfata poate fi privita ca un ***protocol de comunicare*** intre obiecte.

O interfata Java defineste un set de metode, dar nu specifica nicio implementare pentru ele. O clasa care implementeaza o interfata trebuie, obligatoriu, sa specifice implementari pentru toate metodele interfetei, supunandu-se, asadar, unui anumit comportament.

⚠ IMPORTANT !

⚠ O interfata este o colectie de metode fara implementare si declaratii de constante.

Observatie

O interfata poate avea un singur modificator, iar acesta este ***public***. O interfata publica este accesibila tuturor claselor, indiferent de pachetul din care fac parte, implicit nivelul de acces fiind doar la nivelul pachetului din care face parte interfata.
O interfata poate extinde oricate interfete, acestea numindu-se **superinterfete**.

Cod sursa Java

```
interface Inf1 extends Comparable {
    public void metohd1();
}

interface Inf2 extends Inf1 {
    public void method2();
}

public class Test implements Inf2 {
    public void method2() {
        //Metoda din interfata Inf2
    }
    public void metohd1() {
        //Metoda din interfata Inf1
    }
    public int compareTo(Object o) {
        return 0; //Metoda din interfata Comparable
    }
}
```

2 Compararea elementelor

Sa presupunem ca avem o lista, de tip ArrayList, cu elemente de tip ***Student*** si dorim sa sortam elementele din aceasta lista dupa media anilor, presupunand ca ***Student*** este o clasa ce contine printre membrii sai o variabila ce retine media anilor. Pentru realizarea acestui lucru, exista doua posibilitati, ce vor fi detaliate in continuare, folosind ***Comparable*** si ***Comparator***.

2.1 Interfata Comparable

Pentru a putea compara direct o instanta a unui tip (clasa definita de utilizator) cu o alta, este necesar ca tipul respectiv sa implementeze interfata ***Comparable***. Aceasta interfata contine o singura metoda care intoarce:

- *un numar pozitiv* - daca instanta curenta este **mai mare** decat cea primita ca parametru;
- *0* - daca instantele sunt **egale**;
- *un numar negativ* - daca instanta curenta este **mai mica** decat cea primita ca parametru.

Cod sursa Java

```
class Student implements Comparable {
    private String nume;
    private double medie;

    public Student(String nume, double medie) {
        this.nume = nume;
        this.medie = medie;
    }

    @Override
    public int compareTo(Object o) {
        Student obj = (Student) o;
        return this.medie - obj.medie;
    }

    public String toString() {
        String result = nume + " " + medie;
        return result;
    }

    public static void main(String args[]) {
        Vector v = new Vector();
        v.add(new Student("Popescu", 10));
        v.add(new Student("Ionescu", 9.75));
        v.add(new Student("Popa", 9.80));
        Collections.sort(v);
        System.out.println(v);
    }
}
```

2.2 Interfata Comparator

Cod sursa Java

```java
class StudentComparator implements Comparator {
    @Override
    public int compare(Object o1, Object o2) {
        Student s1 = (Student) o1;
        Student s2 = (Student) o2;
        if(s1.getMedie() > s2.getMedie()) {
            return 1;
        } else if(s1.getMedie() == s2.getMedie()) {
            return 0;
        } else {
            return -1;
        }
    }
}

class Student {
    private String nume;
    private double medie;

    public Student(String nume, double medie) {
        this.nume = nume;
        this.medie = medie;
    }

    public String toString() {
        String result = nume + " " + medie;
        return result;
    }

    public double getMedie() {
        return this.medie;
    }

    public static void main(String args[]) {
        Vector v = new Vector();
        v.add(new Student("Popescu", 10));
        v.add(new Student("Ionescu", 9.75));
        v.add(new Student("Popa", 9.80));
        Collections.sort(v, new StudentComparator());
        System.out.println(v);
    }
}
```