Programare orientată pe obiecte

Breviar - Laboratorul 8

Mihai Nan
mihai.nan.cti@gmail.com

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

1 Clase interne

1.1 Introducere

Definitia unei clase poate contine definitia unei alte clase, iar aceste clase, definite in interiorul unei alte clase, se numesc ***imbricate*** sau ***interne***. Tipul acesta de clase reprezinta o functionalitate importanta a limbajului Java, deoarece, prin acest mod, este permisa gruparea claselor care sunt legate logic si exista un control al vizibilitatii bine definit.

Clasele interne sunt de mai multe tipuri, in functie de modul in care se instantiaza si de relatia lor cu clasa exterioara care le contine, ele se impart in:

- clase interne normale (***regular inner classes***);
- clase interne statice (***static nested classes***);
- clase anonime (***anonymous inner classes***);
- clase interne metodelor (***method-local inner classes***) sau blocurilor.

⚠ IMPORTANT !

⚠ Unul din avantajele majore al claselor interne este reprezentat de comportamentul acestora ca un membru al clasei. Astfel, o clasa interna poate avea acces la toti membrii clasei in care este definita (***outer class***), inclusiv la cei declarati folosind modificatorul ***private***.

Observatie

O clasa interna poate avea modificatorii specifici claselor (***abstract***, ***final***) si o serie de modificatori specifici membrilor unei clase (***public***, ***protected***, ***private***, ***static***). Acesti modificatori atribuie clasei interne aceleasi proprietati ca in cazul oricarui membru al unei clase.

1.2 Clase interne normale (*regular inner classes*)

La compilarea unei clase care contine declaratii de clase, se vor obtine fisiere .class pentru fiecare clasa, in cazul clasei interne numele unui astfel de fisier fiind de forma $NumeClasaExterna\$NumeClasaInterna.class$.

O clasa interna poate fi accesata doar printr-o instanta a clasei externe in care este definita (precum metodele si variabile nesatatice ale unei clase).

In continuare, se va prezenta un exemplu care contine o instantiere a unui clase interne.

Cod sursa Java

```
class Outer {
  private int x;

  class Inner {
    private int y;

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

    public int getX() {
      return Outer.this.x;
    }

    public int getY() {
      return this.y;
    }
  }

  public Inner getInstance(int x, int y) {
    Inner obj = new Inner(x, y);
    return obj;
  }
}

class Test {
  public static void main(String args[]) {
    Outer out = new Outer();
    Outer.Inner in1 = out.new Inner(10, 20);
    System.out.println(in1.getX() + " " + in1.getY());
    Outer.Inner in2 = out.getInstance(20, 30);
    System.out.println(in2.getX() + " " + in2.getY());
  }
}
```

In codul de mai sus, sunt uzitate doua modalitati de a obtine o instanta a clasei ***Inner***:

- folosind o metoda, definita in clasa ***Outer***, care instantiaza un obiect de tip ***Inner*** si il returneaza;
- folosind un obiect de tip ***Outer*** pentru a instantia unul de tip ***Inner***.

⚠ IMPORTANT !

⚠ De asemenea, se poate observa in codul de mai sus ca putem accesa referinta la clasa externa folosind numele acesteia si cuvantul cheie ***this***.

```
NumeClasaExterna.this;
```

1.3 Clase interne statice (*static nested classes*)

O clasa interna statica este definita cu un modificator de acces ***static*** si nu are acces direct la membrii clasei externe care nu sunt statici. Trebuie ca, mai intai, sa se declare o instanta a clasei externe si apoi sa se foloseasca instanta creata pentru accesul la membrii clasei externe.

Pentru a creea o instanta de clasa statica se utilizeaza sintaxa:

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

1.4 Clase interne anonime (*anonymous inner classes*)

Clasele interne anonime sunt clase interne dar care nu au un nume. In mod uzual aceste tipuri de clase sunt definite ca si subclase ale unei clase sau ca si implementari ale unor interfete.

Cod sursa Java

```
1  class SuperClass {
2      public void doIt() {
3          System.out.println("SuperClass doIt()");
4      }
5  }
6
7  public class AnonymousTest {
8      public static void main(String[] args) {
9          SuperClass instance = new SuperClass() {
10             public void doIt() {
11                 System.out.println("Anonymous class doIt()");
12             }
13         };
14         instance.doIt();
15     }
16 }
```

Observatie

In mod similar, o clasa anonima poate implementa o interfata, iar pentru intelegerea acestui aspect se recomanda analizarea codului de mai jos.

Cod sursa Java

```
class Test {
  public static void main(String args[]) {
    TreeSet<String> set=new TreeSet<String>(new Comparator(){
      @Override
      public int compare(Object arg0, Object arg1) {
        String s1 = (String) arg0;
        String s2 = (String) arg1;
        return s1.length() - s2.length();
      }

    });
    set.add("Clase interne");
    set.add("Clase anonime");
    set.add("Laborator");
    System.out.println(set);
  }
}
```

1.5 Clase interne metodelor (*method-local inner classes*)

Daca dorim ca vizibilitatea unui clase sa se restranga doar la nivelul unei singure metode, putem opta pentru definirea acestei clase in cadrul metodei. Singurii modificatori care pot fi aplicati acestor clase sunt ***abstract*** sau ***final***.

Cod sursa Java

```
class Outer {
    public void printText() {
        class Local {

        }
        Local local = new Local();
    }
}
```

Cod sursa Java

```java
interface Number<T> {
  public T getValue();
}

class Outer<T> {
  public Number<T> getInnerInstance() {
    class Numar<T> implements Number<T> {
      T value;

      @Override
      public T getValue() {
        return value;
      }
    }
    return new Numar();
  }
}

class Test {
  public static void main(String args[]) {
    Outer<Integer> out = new Outer<Integer>();
    //Eroare la compilare
    Outer.Numar in1 = out.getInnerInstance();
    Number<Integer> in2 = out.getInnerInstance();
  }
}
```

Observatie

Clasele interne, declarate in cadrul metodelor, **NU** pot uzita obiectele declarate in metoda respectiva si nici parametrii metodei. Pentru a le putea accesa, acestea trebuie sa fie declarate ***final***. Aceasta restrictie se datoreaza faptului ca variabilele si parametrii metodelor se afla pe segmentul de stiva (zona de memorie) creat pentru metoda respectiva, ceea ce face ca ele sa nu aiba o perioada de viata egala cu cea a clasei interne. Daca variabila este declarata de tip ***final***, atunci, la rulare, se va stoca o copie a acesteia ca un camp al clasei interne, in acest mod putand fi accesata si dupa executia metodei.

1.6 Clase interne in blocuri

⚠ IMPORTANT !

⚠ La compilare, clasa va fi creata, indiferent daca la rulare nu se executa blocul in care este definita.

Cod sursa Java

```
class Outer {
  public Comparator getInnerInstance(int nr) {
    if(nr % 2 == 0) {
      class MyComparator implements Comparator {
        @Override
        public int compare(Object o1, Object o2) {
          Integer obj1 = (Integer) o1;
          Integer obj2 = (Integer) o2;
          return obj1 - obj2;
        }

      }
      MyComparator comp = new MyComparator();
      return comp;
    } else {
      return null;
    }
  }
}
```

2 Mostenirea claselor interne

Observatie

Deoarece pentru a instantia un obiect al carui tip este reprezentat de o clasa interna este nevoie de o instanta a clasei externe, in care este definita clasa interna, pentru a se atasa constructorul clasei interne, mostenirea unei clase interne nu este tocmai intuitiva, fiind considerata mai complicata decat cea obisnuita. Astfel, pentru intelegerea acesteia, se propune analizarea urmatorului exemplu.

Cod sursa Java

```
class Egg2 {
  protected class Yolk {
    public Yolk() {
      System.out.println("Egg2.Yolk()");
    }

    public void f() {
      System.out.println("Egg2.Yolk.f()");
    }
  }

  private Yolk y = new Yolk();

  public Egg2() {
    System.out.println("New Egg2()");
  }

  public void insertYolk(Yolk yy) {
    y = yy;
  }

  public void g() {
    y.f();
  }
}

class BigEgg1 extends Egg2 {
  public class Yolk extends Egg2.Yolk {
    public Yolk() {
      System.out.println("BigEgg1.Yolk()");
    }

    public void f() {
        System.out.println("BigEgg1.Yolk.f()");
    }
  }

  public BigEgg1() {
    insertYolk(new Yolk());
  }

  public static void main(String[] args) {
    Egg2 e2 = new BigEgg1();
    e2.g();
  }
}
```

3 Poate fi o clasa interna suprascrisa?

In aceasta sectiune vom analiza urmatorul scenariu: avem o clasa care contine definitia unei clase interne si o alta clasa care mosteneste clasa initiala si contine definitia unei clase interne avand acelasi nume cu cea definita in superclasa. Ne dorim sa aflam daca se intampla acelasi lucru ca in cazul metodelor care pot fi suprascrise, iar pentru a afla raspunsul se va propune analizarea urmatorului exemplu.

Cod sursa Java

```
class Egg {
  private Yolk y;
  protected class Yolk {
    public Yolk() {
      System.out.println("Egg.Yolk()");
    }
  }

  public Egg() {
    System.out.println("New Egg()");
    y = new Yolk();
  }
}

class BigEgg2 extends Egg {
  public class Yolk {
    public Yolk() {
      System.out.println("BigEgg2.Yolk()");
    }
  }

  public static void main(String[] args) {
    new BigEgg2();
  }
}
```