Objektovo orientované programovanie

(vnorené typy – 1. časť (pomenované typy))

5. prednáška

Vladislav Novák
FEI STU v Bratislave
14.10.2013

Obsah

Vnorené triedy

Vnorená trieda (*nested class*) – trieda definovaná v rámci inej triedy.

```
class VonkajsiaTrieda {
      class VnorenaTrieda {
            //.....
      }
      // .....
}
```

Vnorená trieda je členom triedy v ktorej je umiestnená. Preto má prístup ku všetkým členom vonkajšej triedy aj keď sú označené ako `private`.

Modifikátory

Vnorenú triedu možno označiť jedným z modifikátorov: `public`, `protected`, `private`, alebo nechať bez týchto modifikátorov (package private).
Pre prístupové práva platia rovnaké pravidlá ako pre členské premenné a metódy.

Vnorené triedy sa delia do dvoch kategórii:

- *statické vnorené triedy* (deklarované so slovom `static`)
- *vnútorné triedy* (*inner class*) = nestatické vnorené triedy

```
public class VonkajsiaTrieda {
  private String instancna = "premenna instancie";
  private static String staticka = "premenna triedy";

  class VnutornaTrieda {
    VnutornaTrieda() {
      System.out.println(instancna);
      System.out.println(staticka);
    }
  }

  static class StatickaVnorenaTrieda  {
    StatickaVnorenaTrieda() {
      System.out.println(staticka);
    }
  }

  public static void main(String[] args) {
    VonkajsiaTrieda vonkajsiObjekt = new VonkajsiaTrieda();

    VonkajsiaTrieda.VnutornaTrieda vnutornyObjekt =
                          vonkajsiObjekt.new VnutornaTrieda();

    VonkajsiaTrieda.StatickaVnorenaTrieda statickaVnorena =
                  new VonkajsiaTrieda.StatickaVnorenaTrieda();
  }
}
```

Použitie:

- Logické zoskupenie tried: Ak je niektorá trieda užitočná iba pre jednu triedu, môže byť vhodné ju vložiť do príslušnej triedy
- Lepšie zapuzdrenie: ak jedna trieda pristupuje k členom inej triedy ktoré by mali byť `private`, môžeme ju vložiť ako vnútornú.
- Môže zlepšovať čitateľnosť a udržovateľnosť kódu

Často sa využíva pri tvorbe GUI.

Statická vnorená trieda (static nested class)

Statická vnorená trieda nemôže priamo pristupovať ku (nestatickým) členom inštancie vonkajšej triedy (podobne ako statická metóda vonkajšej triedy). Ku statickým členom triedy môže pristupovať.

Prístup:

```
VonkajsiaTrieda.StatickaVnutornaTrieda
```

Vytvorenie inštancie statickej vnorenej triedy:

```
VonkajsiaTrieda.StatickaVnorenaTrieda vnorenyObjekt =
                    new VonkajsiaTrieda.StatickaVnorenaTrieda();
```

Vnútorná trieda (inner [non-static] class)

Je pridružená k inštancii vonkajšej triedy. Má prístup aj k (nestatickým) členským premenným a metódam danej inštancie.

Vnútorná trieda nemôže obsahovať statické členy (výnimkou sú konštantné premenné).

Pred vytvorením inštancie vnútornej triedy musí byť vytvorená inštancia vonkajšej triedy.

Vytvorenie inštancie vnútornej triedy:

```
VonkajsiaTrieda.VnútornáTrieda vnutornyObjekt =
                            vonkajsiObjekt.new VnutornaTrieda();
```

Referencia `this` na inštanciu vonkajšiu triedy z vnútornej triedy

Niekedy je potrebné v kóde vnútornej triedy získať referenciu na inštanciu vonkajšej triedy. Kľúčovým slovom `this` vo vnútornej triede získame referenciu na inštanciu vnútornej triedy. Ak potrebujeme získať `this` inštancie vonkajšej triedy, je potrebné pred `this` napísať názov vonkajšej triedy.

príklad:

```
public class VonkajsiaTrieda {
  private int atribut;

  private class VnutornaTrieda {
    private int atribut;

    public void metoda() {
      this.atribut = 10; //atribút inštancie vnútornej triedy
      VonkajsiaTrieda.this.atribut = 20; //atribút inštancie
                                         //vonkajšej triedy
    }
  }
}
```

Príklad použitia vnútornej triedy

```
public class PoleDouble {

    private double[] udaje;

    public PoleDouble(double ... vstup) {
        udaje = new double[vstup.length];
        for (int i = 0; i < udaje.length; i++) {
            udaje[i] = vstup[i];
        }
    }

    private boolean jePlatny(int index) {
        return index >= 0 && index < udaje.length;
    }

    public double dajHodnotu(int index) {
        return jePlatny(index) ? udaje[index] : 0;
    }

    public void nastav(int index, double hodnota) {
        if( jePlatny(index)) {
            udaje[index] = hodnota;
        }
    }

    public double dajDlzku() {
        return udaje.length;
    }

    public Iterator dajIterator() {
        return this.new Iterator();
    }

    public class Iterator {

        private int index; //index pred dalsi prvok

        public Iterator() {
            index = -1;
        }

        public boolean existujeDalsiPrvok() {
            return index + 1 < udaje.length;
        }

        //posunie sa na dalsi prvok a vrati jeho hodnotu
        public double dajDalsiPrvok() {
            index++;
            return udaje[index];
        }
    }
}
```

```
public static void main(String[] args) {
    PoleDouble pole = new PoleDouble(1,2,3,4,5);

    //iterátor súvisí s inštanciou pola,
    //môžeme vytvoriť viacero iterátorov súvisiacich s jednou
    //inštanciou pola
    PoleDouble.Iterator iterator = pole.dajIterator();
    while( iterator.existujeDalsiPrvok()) {
        System.out.println(iterator.dajDalsiPrvok());
    }
}
```

výstup:

```
1.0
2.0
3.0
4.0
5.0
```

Lokálne vnútorné triedy

Existujú dva špeciálne typy vnorených tried. Možno ich deklarovať v rámci tela metódy:

- *lokálne vnútorné triedy* (*local class*) – trieda deklarovaná v rámci tela metódy
- *anonymné vnútorné triedy* (*anonymous class*) – trieda deklarovaná v rámci tela metódy, ale nie je pomenovaná. Budeme sa nimi zaoberať neskôr.

príklad:

súbor Uloha.java:

```
public interface Uloha {
	void pocitaj();
}
```

súbor Spustac.java:

```
public class Spustac {
	private Uloha uloha;

	public void nastav(Uloha novaUloha ) {
		uloha = novaUloha;
	}

	public void spusti()  {
		if( uloha != null ) {
			uloha.pocitaj();
		}
	}
}
```

súbor PrikladPreLokalnuTriedu.java:

```
public class PrikladPreLokalnuTriedu {
	public static void main(String[] args) {

		//lokalna trieda
		class Uloha1 implements Uloha{
			public void pocitaj() {
				System.out.println("lokalna trieda");
			}
		}
		Spustac spustac = new Spustac();
		spustac.nastav(new Uloha1());


		//niekde dalej:
		spustac.spusti();
	}
}
```

Vnorené rozhrania

príklad:

```
public class PoleDouble {
    private double[] udaje;

    public PoleDouble(double ... vstup) {
        udaje = new double[vstup.length];
        for (int i = 0; i < udaje.length; i++) {
            udaje[i] = vstup[i];
        }
    }

    @Override
    public String toString() {
        return Arrays.toString(udaje);
    }

    private interface Funkcia {
        double vykonaj(double prvok, double argument);
    }

    private class Pripocitavac implements Funkcia {
        @Override
        public double vykonaj(double prvok, double argument) {
            return prvok + argument;
        }
    }

    private class Nasobic implements Funkcia {
        @Override
        public double vykonaj(double prvok, double argument) {
            return prvok * argument;
        }
    }

    private void vykonajCyklus(double argument, Funkcia funkcia) {
        for(int i = 0; i < udaje.length; i ++) {
            udaje[i] = funkcia.vykonaj(udaje[i], argument);
        }
    }

    public void pripocitajKuVsetkymPrvkom(double scitanec) {
        //for(int i = 0; i < udaje.length; i++) {
        //    udaje[i] += scitanec;
        //}
        vykonajCyklus(scitanec, new Pripocitavac());
    }

    public void vynasomKazkyPrvok(double nasobitel) {
        //for(int i = 0; i < udaje.length; i++) {
        //    udaje[i] *= nasobitel;
        //}
        vykonajCyklus(nasobitel, new Nasobic());
    }
}
```

```
public static void main(String[] args) {
    PoleDouble pole = new PoleDouble(1,2,3,4,5);

    System.out.println(pole); // [1.0, 2.0, 3.0, 4.0, 5.0]

    pole.pripocitajKuVsetkymPrvkom(2);
    System.out.println(pole); // [3.0, 4.0, 5.0, 6.0, 7.0]

    pole.vynasomKazkyPrvok(10);
    System.out.println(pole); // [30.0, 40.0, 50.0, 60.0, 70.0]
}
```

výstup:

```
[1.0, 2.0, 3.0, 4.0, 5.0]
[3.0, 4.0, 5.0, 6.0, 7.0]
[30.0, 40.0, 50.0, 60.0, 70.0]
```