Objektovo orientované programovanie

(enumeračné typy)

8. prednáška (3. časť)

Vladislav Novák
FEI STU v Bratislave
4.11.2014
(úprava 10.11. na str. 1, 3, 5)

Enumeračné typy (Vymenované typy)

Enumeračný typ je typ ktorý obsahuje fixnú množinu konštánt daného typu (napríklad dni v týždni typu deň). Názov každej konštanty sa podľa konvencie píše veľkými písmenami. Enumeračný typ je vhodné použiť v prípade, keď potrebujeme reprezentovať pevnú množinu konštánt (známu v čase kompilácie, keď predpokladáme že sa nebude meniť napr. dni v týždni).
Enumeračný typ sa definuje kľúčovým slovom `enum`.

príklad:
jednoduchá definícia enumeračného typu:

```
public enum Den {
    PONDELOK, UTOROK, STREDA, STVRTOK, PIATOK, SOBOTA, NEDELA
}
```

Enumeračný typ `Den` definuje zoznam enumeračných konštánt (`PONDELOK`, ..., `NEDELA`). Podľa zoznamu enumeračných konštánt kompilátor automatický vytvorí inštancie typu `Den`. Nie je možné vytvoriť ďalšie inštancie typu `Den`.

použitie enumeračného typu:

```
public class PouzitieEnum {
    private static void vytlacInfo(Den den) {
         //vo switch s enumeračným typom v častiach case nepíšeme typ (napr.: ~~Den.~~PONDELOK)
         switch(den) {
            case PONDELOK:
            case UTOROK:
            case STREDA:
            case STVRTOK:
            case PIATOK:
              System.out.println("den " + den + " je pracovny");
              break;
            case SOBOTA:
            case NEDELA:
              System.out.println("den " + den + " je vikendovy");
              break;
          }
    }

    public static void main(String[] args){
         Den dnes = Den.PONDELOK;

         System.out.println("dnes je " + dnes); //dnes.toString()

         if( dnes == Den.PONDELOK ) {
             System.out.println("dnes je pondelok");
         }
         else {
             System.out.println("dnes nie je pondelok");
         }

         vytlacInfo(dnes);

         vytlacInfo(Den.SOBOTA);
    }
}
```

výstup:

```
dnes je PONDELOK
dnes je pondelok
den PONDELOK je pracovny
den SOBOTA je vikendovy
```

Deklarácia `enum` definuje triedu, nazývanú *enumeračný typ*. Jej telo môže obsahovať premenné a metódy. Niektoré sú kompilátorom pridané automaticky.

Enumeračný typ implicitne obsahuje členské premenné, deklarované ako `public static final`, pomenované podľa deklarovaných enumeračných konštánt. Každá takáto členská premenná je inicializovaná na enumeračnú konštantu, ktorá s ňou korešponduje.

Príkladom automaticky doplnenej metódy je napríklad statická metóda `values`, ktorá vracia pole obsahujúce všetky enumeračné konštanty daného typu, v takom poradí v ktorom boli definované.

Ďalším príkladom je automaticky doplnená metóda `ordinal`, ktorá vráti číslo vyjadrujúce poradie enumeračnej konštanty v zozname. Prvá enumeračná konštanta má číslo 0.

príklad:

```
public enum Den {
    PONDELOK, UTOROK, STREDA, STVRTOK, PIATOK, SOBOTA, NEDELA;
}
```

```
public static void main(String[] args) {
    for (Den den : Den.values()) {
            System.out.println(den + " " + den.ordinal());
    }
}
```

výstup:

```
PONDELOK 0
UTOROK 1
STREDA 2
STVRTOK 3
PIATOK 4
SOBOTA 5
NEDELA 6
```

Metóda `compareTo` slúži na porovnávanie poradia enumeračných konštánt. Vráti záporne celé číslo, nulu, alebo kladné celé číslo, podľa výsledku porovnania poradia enumeračných konštánt. Je definovaná rozhraním `Comparable<T>`, ktoré enumeračné typy implementujú.

príklad:

```
public enum Den {
    PONDELOK, UTOROK, STREDA, STVRTOK, PIATOK, SOBOTA, NEDELA;
}
```

```
public static void main(String[] args) {
    System.out.println(Den.PONDELOK.compareTo(Den.UTOROK)); // -1
                                                // (PONDELOK < UTOROK)

    System.out.println(Den.UTOROK.compareTo(Den.PONDELOK)); //  1
                                                // (UTOROK > PONDELOK)

    System.out.println(Den.PONDELOK.compareTo(Den.PONDELOK));// 0
                                               // (PONDELOK = PONDELOK)
}
```

výstup:

```
-1
1
0
```

Všetky enumeračné typy implicitne dedia od triedy `java.lang.Enum`. Pretože Java nepodporuje viacnásobnú dedičnosť, enumeračný typ nemôže dediť od inej triedy.

Enumeračný typ môže okrem zoznamu enumeračných konštánt obsahovať atribúty a metódy. Atribúty a metódy musia byť uvedené až za zoznamom enumeračných konštánt.
Ak sú v enumeračnom type uvedené atribúty a metódy, musí byť zoznam enumeračných konštánt ukončený bodkočiarkou.

Konštruktory enumeračného typu musia byť typu *package-private*, alebo `private`. Samotné enumeračné konštanty uvedené na začiatku definície enumeračného typu sa vytvárajú automaticky (s využitím konštruktora). Konštruktor nemožno vyvolať „ručne“.

```
public enum Den {
    //zoznam enumeračných konštánt je uvedený ako prvý
    PONDELOK, UTOROK, STREDA, STVRTOK, PIATOK, SOBOTA, NEDELA;

    //za zoznamom enumeračných konštant môžu nasledovať ďalšie členy
    private int premenna;
    public void metoda () { ..... }
    Den() { .....} //konstruktor
}
```

príklad:

```
public enum Den {
      PONDELOK("pondelok", true),
      UTOROK  ("utorok"  , true),
      STREDA  ("streda"  , true),
      STVRTOK ("stvrtok" , true),
      PIATOK  ("piatok"  , true),
      SOBOTA  ("sobota"  , false),
      NEDELA  ("nedela"  , false);

      private final String nazov;
      private final boolean beznePracovny;
      private final String skratka;

      Den(String nazov, boolean pracovny) {
            this.nazov = nazov;
            this.beznePracovny = pracovny;
            this.skratka = nazov.substring(0, 2);
      }
      public int dajPoradie(){
            return ordinal() + 1;//ordinal() vráti por. č. enum konštanty
      }
      public String dajNazov(){
            return nazov;
      }
      public boolean dajPracovnost(){
            return beznePracovny;
      }
      public String dajSkratku(){
            return skratka;
      }
}
```

```
public static void main(String[] args) {
    for( Den den : Den.values() ) {
         System.out.format("%d: %-8s (%s) %s%n",
                    den.dajPoradie(),
                    den.dajNazov(),
                    den.dajSkratku(),
                    (den.dajPracovnost() ? "pracovny" : "volny")
          );
    }
}
```

výstup:

```
1: pondelok (po) pracovny
2: utorok   (ut) pracovny
3: streda   (st) pracovny
4: stvrtok  (st) pracovny
5: piatok   (pi) pracovny
6: sobota   (so) volny
7: nedela   (ne) volny
```

Nasledujúci kód približne naznačuje spôsob automatického doplnenia členských premenných podľa zoznamu enumeračných konštánt (v predchádzajúcom príklade). Kompilátor pridáva aj ďalšie parametre pri volaní konštruktora obsahujúce názov a poradie. Avšak každý kompilátor môže tieto parametre pridať iným spôsobom.

```
public enum Den {
 public static final Den PONDELOK = new Den("pondelok",true, "PONDELOK",0);
 public static final Den UTOROK   = new Den("utorok"  ,true, "UTOROK",  1);
 public static final Den STREDA   = new Den("streda"  ,true, "STREDA",  2);
 public static final Den STVRTOK  = new Den("stvrtok" ,true, "STVRTOK", 3);
 public static final Den PIATOK   = new Den("piatok"  ,true, "PIATOK",  4);
 public static final Den SOBOTA   = new Den("sobota"  ,false,"SOBOTA",  5);
 public static final Den NEDELA   = new Den("nedela"  ,false,"NEDELA",  6);
 // ďalšie členy
}
```

Enumeračný typ môže deklarovať abstraktné metódy, ktoré musia enumeračné konštanty implementovať. Príklad:

```
public enum Funkcia {
    PLUS ('+'){ //parameter konštruktora
        @Override
        public int vyhodnot(int a, int b) {
            return a + b;
        }
    },
    MINUS ('-'){ //parameter konštruktora
        @Override
        public int vyhodnot(int a, int b) {
            return a - b;
        }
    };

    private char znak;

    Funkcia(char znak) {
        this.znak = znak;
    }

    //enumeračné konštanty musia definovať implementaciu abstraktnej metódy
    public abstract int vyhodnot(int a, int b);

    @Override
    public String toString() { //príklad prekrytia metódy toString
        return "" + znak;
    }
}
```

```
public static void main(String[] args) {
int a = 10;
int b = 5;

Funkcia funkcia = Funkcia.PLUS;
System.out.println(a+" "+funkcia+" "+b+" = "+funkcia.vyhodnot(a,b));

funkcia = Funkcia.MINUS;
System.out.println(a+" "+funkcia+" "+b+" = "+funkcia.vyhodnot(a,b));
}
```

výstup:

```
10 + 5 = 15
10 - 5 = 5
```