8. Výčtový typ

V této kapitole si ukážeme, jak implementovat v Javě „statické“ seznamy konstant (hodnot). Příkladem mohou být dny v týdnu, měsíce v roce, planety obíhající kolem slunce či přípustné parametry na příkazové řádce. Jedná se o seznamy, které se nemění za běhu aplikace – pokud se seznam změní (např. přibude další parametr příkazové řádky), musí se program znovu přeložit. Nebudeme se zde zabývat seznamy, kterým se za běhu programu počet prvků mění (např. seznamy studentů, seznamy místností, seznamy věcí).
Nejdříve si ukážeme implementaci seznamu pomocí celočíselných pojmenovaných konstant, většina kapitoly je věnována výčtovému typu, který je novinkou ve verzi Javy 5.0.

8.1. Pojmenované konstanty

Pro vyjádření seznamu přípustných hodnot lze použít celočíselné pojmenované konstanty.
U pojmenovaných konstant se uvádějí modifikátory **final** (hodnotu konstanty nelze změnit), **static** (konstanty jsou nezávislé na instancích) a obvykle i **public** – konstanty bývají veřejně dostupné, což opět souvisí s jejich neměnitelností. Ukážeme si je na příkladu se dny v týdnu:

```
    public static final int DEN_NEDELE=0;
    public static final int DEN_PONDELI=1;
    public static final int DEN_UTERY=2;
    public static final int DEN_STREDA=3;
    public static final int DEN_CTVRTEK=4;
    public static final int DEN_PATEK=5;
    public static final int DEN_SOBOTA=6;
```

Tyto konstanty použijeme při vkládání termínů do rozvrhu. Hlavička metody by mohla vypadat takto:

```
    public void pridejTermin(int den, String odKdy, String doKdy,
                                 String popis) {
        // obsah metody
    }
```

Vlastní přidávání termínů může vypadat takto:

```
    rozvrh.pridejTermin(DEN_PONDELI, "14:30", "16:00",
                                        "konzultační hodiny");
    rozvrh.pridejTermin(DEN_UTERY, "12:45", "14:15",
                                        "konzultační hodiny");
```

Použití pojmenovaných konstant pro vyjádření seznamu hodnot má ale několik nevýhod:

- řešení není typově bezpečné – bez upozornění se přeloží i následující kód:

```
        rozvrh.pridejTermin(999, "14:30", "16:00",
                                            "konzultační hodiny");
```

- řešení nepodporuje vkládání konstant – pokud se do seznamu konstant vloží nová konstanta či pokud se změní pořadí konstant, musí se znovu přeložit všechny třídy, které konstanty používají,
- při tisku mají malou vypovídací hodnotu – v našem příkladu se vytiskne číslo a ne např. „pondělí“ či „DEN_PONDELI“.

Pojmenované konstanty se používají i v jiných situacích, např. pro vyjádření matematických konstant. V těchto případech mají své opodstatnění a samozřejmě nelze hovořit o výše uvedených nevýhodách.

8.2. Výčtový typ (enum)

V Javě 5.0 byl zaveden výčtový typ – **enum**, který řeší většinu nevýhod pojmenovaných konstant pro vytvoření uzavřené množiny neměnných hodnot. Základní deklarace výčtového typu je velmi jednoduchá:

```
[public] enum název {
         hodnota1,
         hodnota2,
         ...
     }
```

Výčtový typ se obvykle deklaruje v samostatném souboru podobně jako třídy či rozhraní. Soubor se musí jmenovat stejně jako výčtový typ. Jméno výčtového typu by mělo začínat velkým písmenem, jména vlastních hodnot by měly být velkými písmeny obdobně jako pojmenované konstanty.
V deklaraci výčtového typu se též obvykle používá i modifikátor přístupu **public**. Následuje příklad deklarace výčtového typu pro jednotlivé dny v týdnu:

```
public enum DenVTydnu {
     NEDELE,
     PONDELI,
     UTERY,
     STREDA,
     CTVRTEK,
     PATEK,
     SOBOTA
}
```

Při použití tohoto výčtového typu v rozvrhu musíme upravit hlavičku a částečně asi i obsah metody *`pridejTermin()`*. V hlavičce metody bude první parametr výčtového typu *`DenVTydnu`*:

```
     public void pridejTermin(DenVTydnu den, String odKdy,
                                  String doKdy, String popis) {
          // obsah metody
     }
```

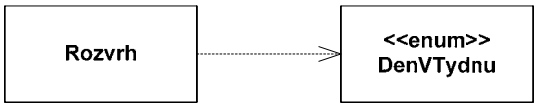
Vlastní přidávání termínů bude nyní vypadat takto:

```
     rozvrh.pridejTermin(DenVTydnu.PONDELI, "14:30", "16:00",
                                          "konzultační hodiny");
     rozvrh.pridejTermin(DenVTydnu.UTERY, "12:45", "14:15",
                                          "konzultační hodiny");
```

Řešení s výčtovým typem:

- je typově bezpečné – nelze za den v týdnu doplnit hodnotu, která není ve výčtovém typu,
- je odolnější vůči změnám výčtového typu – pokud se změní pořadí hodnot ve výčtovém typu či se dovnitř seznamu vloží další hodnota, není potřeba překládat třídy, které výčtový typ používají.
- při tisku se vytiskne smysluplnější hodnota, vytiskne se vlastní konstanta (např. „PONDELI“).

V diagramu tříd se vyznačí výčtový typ s použitím stereotypu:



Obrázek 8.1 Diagram tříd s výčtovým typem

8.3. Metody výčtového typu

Výčtový typ patří mezi referenční typy, konkrétně je speciálním potomkem třídy `Object`, hodně se podobá třídám. Na rozdíl od tříd a rozhraní však nepodporuje dědičnost – nelze vytvořit výčtový typ, který by byl potomkem jiného výčtového typu.
Jako potomek třídy `Object` obsahuje výčtový typ též všechny metody třídy `Object` – metody `toString()`, `equals()`, `hashCode()` a další.
Výčtový typ má několik dalších metod. Statická metoda **values()** vrací výčet všech hodnot výčtového typu. Používá se pro procházení výčtového typu pomocí cyklu `for`. Následující kód vypíše všechny hodnoty výčtového typu `DenVTydnu` (vypíše všechny dny):

```
for (DenVTydnu den : DenVTydnu.values() ) {
     System.out.println(den);
}
```

U výčtových typů lze používat statickou metodu **valueOf()**, která vrátí prvek výčtového typu odpovídající zadanému řetězci. Pokud se nenajde prvek výčtového typu pro zadaný řetězec, vznikne výjimka `IllegalArgumentException`. Při vyhledávání prvku se kontroluje velikost písmen.
V následujícím kódu vznikne na druhém řádku výjimka:

```
DenVTydnu sobota = DenVTydnu.valueOf("SOBOTA");
DenVTydnu nedele = DenVTydnu.valueOf("nedele"); // výjimka !!!
```

Výčtový typ též implementuje rozhraní `Comparable` a tudíž obsahuje metodu `compareTo()` – dle výčtového typu je možné třídit. Prvky se řadí dle pořadí, v jakém jsou uvedeny ve výčtovém typu.
V našem příkladu se dny by se rozvrh řadil od neděle do soboty.

8.4. Rozšíření příkazu switch

V souvislosti s výčtovým typem byl též rozšířen příkaz **switch** – lze v něm uvést vedle celočíselných primitivních typů a typu `char` i výčtový typ. I u výčtového typu je vhodné vždy uvádět větev **default** a to i v případě, že vyjmenujeme všechny přípustné hodnoty – je to určitá ochrana pro případ přidání další hodnoty do výčtového typu.
V následujícím příkazu `switch` se odlišuje víkend od pracovních dní (je to i jedna z mála situací, kdy ve větvích `case` není vždy uveden příkaz `brek`):

```
// proměnná den je typu DenVTydnu

switch (den) {
     case NEDELE:
     case SOBOTA: System.out.println("víkend"); break;
     case PONDELI:
     case UTERY:
     case STREDA:
     case CTVRTEK:
     case PATEK: System.out.println("pracovní den"); break;
     default : System.out.println("takový den neznám");
}
```

Všimněte si, že za slovem `case` se píší konkrétní konstanty výčtového typu bez odkazu na vlastní výčtový typ.

8.5. Výčtový typ uvnitř třídy

V některých situacích potřebujeme seznam přípustných hodnot pouze uvnitř nějaké třídy – nechceme, aby výčtový typ byl veřejně dostupný. Řešením je definice výčtového typu uvnitř třídy. Ukážeme si to na příkladu se znaménky pro matematické operace:

```
 1 public class Vypocet {
 2
 3   private enum Operace { PLUS, MINUS, NASOBENO, DELENO }
 4
 5   public double vypocet (Operace operace, double prvni,
 6                                       double druhy) {
 7   switch (operace) {
 8      case PLUS: return prvni + druhy;
 9      case MINUS: return prvni - druhy;
10      case NASOBENO: return prvni * druhy;
11      case DELENO: return prvni/druhy;
12      default : return 0;
13   }
14   }
15   public static void main (String ... args) {
16   Vypocet spocti = new Vypocet();
17   System.out.println("3 PLUS 5 = " +
18                           spocti.vypocet(Operace.PLUS, 3, 5));
19   }
20 }
```

Pokud je výčtový typ definován uvnitř třídy, je možné v deklaraci uvést všechny modifikátory přístupu (tedy i *`private`* a *`protected`*) – v tomto případě se chovají stejně, jako u datových atributů. V závislosti na modifikátoru lze deklarovaný výčtový typ používat i v potomcích hlavní třídy (v našem případě v potomcích třídy *`Vypocet`*).
Pokud chceme, aby výčtový typ byl veřejně dostupný, měl by být definovaný v rámci samostatného souboru a ne uvnitř nějaké třídy.

8.6. Výčtový typ a datové struktury

Pokud chceme ukládat výčtový typ do množin či ho používat jako klíč v mapách (viz kapitola 10), poskytuje Java v balíčku *`java.util`* speciální rychlé implementace množiny a mapy – třídy *`EnumSet`* a *`EnumMap`*.
Třída *`EnumSet`* má navíc metodu *`range()`*, kterou lze použít pro procházení pouze části seznamu prvků ve výčtovém typu. V následujícím příkladu se vypíší pouze pracovní dny:

```
    for (Den den : EnumSet.range(Den.PONDELI, Den.PATEK)) {
        System.out.println(den);
    }
```

Třída *`EnumSet`* dále podporuje některé základní operace s množinami výčtových typů.

8.7. Rozšiřování výčtového typu

Výčtové typy nemusí být tak jednoduché, jak jsme si doposud ukazovali. K hodnotám ve výčtovém typu lze přiřazovat další parametry (např. k planetám lze přiřadit vzdálenost od slunce, k názvům barev lze doplnit jejich označení pomocí RGB), výčtový typ lze rozšiřovat o další metody či překrývat již existující metody. Tato problematika je však již poměrně komplikovaná a proto doporučujeme ji nastudovat z dokumentace u firmy Sun. My si zde ukážeme pouze jednoduchý příklad, ve kterém překryjeme metodu *`toString()`*, aby vypisovala dny v týdnu malými písmeny:

```
public enum Den {
     NEDELE,
     PONDELI,
     UTERY,
     STREDA,
     CTRVTEK,
     PATEK,
     SOBOTA;

     public String toString() {
          return name().toLowerCase();
     }
}
```