Budapesti Műszaki és Gazdaságtudományi Egyetem

Java technológia

Enumerációk

Sipos Róbert
siposr@hit.bme.hu

2014. 02. 27.

Helyettesítő megoldások I.

- A Javában az 5.0-ás verzió előtt nem volt felsorolási típus.
- Mivel gyakran van szükség ezen funkcionalitás megvalósítására, több megoldás (idióma) is született.
- A legegyszerűbb helyettesítő eszköz integer konstansok deklarálása:

```
public class Dragon {

   public static final int COPPER = 0;
   public static final int SILVER = 1;
   public static final int GOLD = 2;
}
```

- Előnye, hogy csak egy "integernyi" helyet foglal a memóriában, és használható `switch`-ben:

Helyettesítő megoldások II.

```
int dragon;
...
switch (dragon) {
   case Dragon.COPPER : ...
   case Dragon.SILVER : ...
   case Dragon.GOLD : ...
}
```

- Hátránya viszont, hogy nincs típusellenőrzés, kiiratva nem informatív, nem lehet kényelmesen `Collection`be tenni, és a konstanst felhasználó kódba **értékként fordul be**, ami az értékek megváltozásakor bináris inkompatibilitást eredményez.

```
int dragon = 5;                                    // Ilyen sárkány nincs...
```

```
int dragon = Dragon.GOLD;
...
System.out.println (dragon);                       // 2. Ez vajon mit jelent?
```

Helyettesítő megoldások III.

- A másik, gyakran használt módszer a `String` konstansok használata:

```
public class Dragon {

   public static final String COPPER = "copper";
   public static final String SILVER = "silver";
   public static final String GOLD = "gold";
}
```

- Ez már jobb, de több helyet foglal a memóriában, még mindig nincs típusellenőrzés, és referencia lévén `switch`ben sem használható (Java7 előtt).

Helyettesítő megoldások IV.

- A jó megoldás egy kívülről nem példányosítható osztály példányainak használata:

```
public class Dragon {

   private Dragon (String name) { this.name = name; }
   public String toString () { return name; }

   public static final Dragon COPPER = new Dragon ("copper");
   public static final Dragon SILVER = new Dragon ("silver");
   public static final Dragon GOLD = new Dragon ("gold");

   private final String name;
}
```

- Ez eddig a legjobb megoldás, de `switch`ben ez sem használható.

Helyettesítő megoldások V.

- Probléma akkor lehet, ha az értékeket szerializálni akarjuk, ilyenkor a `Dragon` osztály implementálja a `Serializable` interfészt.
- Ha azonban több forrásból töltök be `Dragon` értékeket, akkor egy-egy értékből több példány is létrejöhet, ezt ki kell védeni egy kanonizáló lépésben:

```
public class Dragon implements Serializable {

   ...
   public Object readResolve () {
      if (COPPER.name.equals (name)) return COPPER;
      else if (SILVER.name.equals (name)) return SILVER;
      else return GOLD;
   }
   ...
}
```

Enumerációk definiálása

- Az új enumeráció típus a fentieket valósítja meg, sőt `switch`ben is használható.
- Enumerációt az **`enum`** kulcsszó segítségével definiálhatunk:

```
public enum Dragon { COPPER,SILVER,GOLD };
```

- A `Dragon`t **enum típus**nak, vagy **osztály**nak hívják (enum type, vagy enum class), az értékeket pedig enum konstansok (enum constant).
- Az enum típusok önálló osztályok, a `java.lang.Enum` osztályból származnak, ebből következően az osztályokra és interfészekre vonatkozó láthatósági szabályok érvényesek rájuk.
- Az enum konstansok a megfelelő enum osztály példányai.

Enumerációk használata

- Az enum konstansokra az enum osztály és a konstans nevével hivatkozhatunk, mert **a konstans az enum osztály egy mezője**.

```
Dragon d = Dragon.SILVER;
```

- A statikus import segítségével az enumok használata is egyszerűsödik:

```
import static monster.Dragon.*;

Dragon d = SILVER;
```

- Az enumok switch-ben is használhatók:

```
switch (dragon) {
   case GOLD : runLikeHell ();
   ...
}
```

Az Enum osztály tulajdonságai I.

- Minden enum típusnak van értelmes toString, hashCode és equals metódusa, mindegyik Serializable, Comparable, és effektíve final (a privát konstruktor miatt).
- Egy enum konstanstól lekérdezhetjük, melyik enum osztályban deklarálták:

```
Dragon.SILVER.getDeclaringClass ();                    // Dragon.class
```

- Lekérdezhetjük az enum konstans sorszámát (0-tól kezdődően, deklarálási sorrendben), illetve a nevét:

```
Dragon.GOLD.ordinal ();                                          // 2
Dragon.COPPER.name ();                                  // "COPPER"
```

Az Enum osztály tulajdonságai II.

- Az Enum osztály statikus valueOf metódusa segítségével megkaphatjuk egy megadott enum osztály megadott nevű konstansát.

```
Enum.valueOf (Dragon.class,"SILVER");                    // Dragon.SILVER
```

- A Class osztály getEnumConstants metódusával megkaphatjuk az adott enum osztály konstansait:

```
Dragon[] dragons = Dragon.class.getEnumConstants ();
```

Enum osztályok kiegészítése I.

- Az enum típust kiegészíthetjük tetszés szerinti metódusokkal, konstruktorokkal és mezőkkel:

```
public enum Dragon {
   COPPER (5),SILVER (10),GOLD (20);

   Dragon (int weight) { this.weight = weight; }
   public int getWeight () { return weight; }

   private int weight;
}
```

Enum osztályok kiegészítése II.

- Az egyes enum konstansokhoz viselkedést is rendelhetünk:

```
public abstract enum Dragon {
   COPPER {
      void action () { eatTheVirgin (); }
   },
   SILVER {
      void action () { killTheKnight (); }
   },
   GOLD {
      void action () { burnTheCity (); }
   };

   abstract void action ();

   ...
}
```

EnumSet, EnumMap I.

- A Collections Framework tartalmaz egy speciálisan enum konstansok tárolására szolgáló Set, és egy enum kulcsokkal használható Map megvalósítást.
- Az EnumSet és az EnumMap enum konstansok tárolására lényegesen hatékonyabb, mint a többi Set és Map implementáció.
- Az EnumSet-nek nincs publikus konstruktora, a Set-et halmazműveletekkel lehet létrehozni:

```
EnumSet.allOf (Dragon.class);                          // minden sárkány
EnumSet.noneOf (Dragon.class);                       // egyik sárkány sem
EnumSet.of (Dragon.SILVER,Dragon.COPPER);      // a silver és a copper
EnumSet.range (Dragon.COPPER,Dragon.GOLD);             // minden sárkány
EnumSet.complementOf (EnumSet.of (Dragon.GOLD));    // csak a gold nem
```

EnumSet, EnumMap II.

- Az EnumMap létrehozásakor a kulcsok típusát meg kell adnunk:

```
EnumMap<Dragon,Integer> knightsEaten = new EnumMap (Dragon.class);
```

- Az értékeket a szokásos módon adhatunk hozzá és kérdezhetünk le:

```
public void eatKnight (Dragon d) {
   Integer ke = knightsEaten.get (d);
   if (ke == null)
      ke = 0;
   knightsEaten.put (d,ke + 1);
}
```