Objektovo orientované programovanie

Učiteľ:
Ing. Jozef Wagner PhD.

Učebnica:
https://oop.wagjo.com/

OPG
Teória 14

1. Enumeračné typy
2. Pomocou Stringov a celých čísel
3. Pomocou tried a dedičnosti
4. Trieda enum

Enumeračné typy

Nečíselné dátové typy s konečným počtom hodnôt

- Stav objednávky: nová, zpracovávaná, expedovaná, doručená
- Dni v týždni: pondelok, utorok, streda, štvrtok, piatok, sobota, nedeľa
- Pohlavie: muž, žena
- Svetové strany: sever, juh, východ, západ
- Veľkosť trička: XS, S, M, L, XL, XXL

Ako takéto hodnoty čo najlepšie v Jave reprezentovať?

Enumeračné typy pomocou Stringov

Reťazce vieme použiť v týchto prípadoch:

- hodnôt je pomerne veľa (cca viac ako 30)
- hodnoty nevieme zoradiť
- v budúcnosti môžu pribudnúť nové hodnoty
- hodnoty sú rovnocenné, teda nemáme niektoré hodnoty, ktoré majú špeciálnu logiku alebo dáta

Nevýhody:

- pri zadávaní môže nastať preklep
- nemáme žiadnu typovú kontrolu

String ako enumeračný typ

```
class Person {
    //private String name;
    private String country;

    public Person(String country) {
        this.country = country;
    }
    public String getCountry() {
        return country;
    }
}

Person p = new Person("SK");
System.out.println(p.getCountry());
```

Príklady:

- krajina
- platobná mena
- kategórie výrobkov
- značky tovarov
- kategórie z databáz, nad ktorými nemáme kontrolu.

Pomocou číselných typov

Číselné typy vieme použiť v týchto prípadoch:

- hodnoty vieme zoradiť
- hodnôt je veľké množstvo (viac ako 30)
- hodnoty majú matematický význam
- potrebujeme veľmi efektívne spracovanie (rýchlosť, málo miesta v pamäti)
- hodnoty majú číselný štandard (HTTP kódy)

Nevýhody:

- musíme štandardizovať číselnú hodnotu (Pondelok bude 0 alebo 1?, Najvyššia priorita je 1 alebo 10?)
- nemáme žiadnu typovú kontrolu

Celé číslo ako enumeračný typ

```
class Response {
    int code;
    String body;

    Response(int code, String body) {
        this.code = code;
        this.body = body;
    }

    boolean isSuccess() { return code >= 200 && code < 300; }
    boolean isNotFound() { return code == 404; }
}

// Použitie
Response r = new Response(404, "Stránka nenájdená");
```

Pomocou tried a dedičnosti

Triedy a dedičnosť typy vieme použiť ak:

- hodnôt nie je veľa
- k hodnotám je potrebné priradiť dodatočnú logiku alebo stav/dáta

Nevýhody:

- Existuje viacero inštancií jednej hodnoty
- Príliš veľa kódu, veľká zložitosť

```
public class Objednavka {
    private Tovar[] tovar;
    private double celkovaSuma;
    private Adresa adresa;
    private Platba sposobPlatby;
}

public abstract class Platba {
    public abstract boolean mozeOdoslat();
}

public class Dobierka extends Platba {
    public boolean mozeOdoslat() {
        return true;
    }
}
```

```
public class PlatbaKartou extends Platba {
    private boolean jeZaplatene;

    public boolean mozeOdoslat() {
        return jeZaplatene;
    }

    public void otvorPlatobnuBranu() {
        // ...
    }
}
```

```
public class PlatbaPrevodom extends Platba {
    private String variabilnySymbol;

    public boolean mozeOdoslat() {
        // ... skontroluj ucet
    }

    public String getVariabilnySymbol () {
        return variabilnySymbol;
    }

}
```

Pomocou enum

Enum (enumeration) je špeciálny typ triedy v Jave, ktorý reprezentuje pevný, nemenný a obmedzený zoznam hodnôt

```
public enum Pohlavie { MUZ, ZENA }
```

```
public enum svetovaStrana {
    SEVER, VYCHOD, JUH, ZAPAD
}
```

```
public class Osoba {
   private String meno;
   private String priezvisko;
   private Pohlavie pohlavie;

   public Osoba(String meno, String priezvisko, Pohlavie pohlavie) {
      this.meno = meno;
      this.priezvisko = priezvisko;
      this.pohlavie = pohlavie;
   }
}

Osoba o = new Osoba("Ján", "Novák", Pohlavie.MUZ);

if(o.getPohlavie() == Pohlavie.MUZ) { // použitie v podmienke
      // ...
}
```

Pomocou enum

Vlastnosti:

- Hodnoty sú statické konštanty, preto názvy sú veľkými písmenami
- Zakázané vytvárať vlastné objekty enumu
- Všetky enumy dedia od `java.lang.Enum`
- `valueOf()` - vytváranie enumu z reťazca
- `name()` - názov enumu vo forme reťazca

Pomocou enum

Enum je vhodný pre nasledovné situácie:

- Existuje pevná množina hodnôt
- Počet hodnôt nie je veľký
- Hodnoty nepotrebujú špeciálne dáta alebo správanie
- Potrebujeme silnú typovú kontrolu

Nevýhody enumu:

- Zoznam hodnôt je pevne daný a nedá sa rozšíriť
- Je pomalý ak máme veľké množstvo hodnôt (viac ako 1000)

Enum vieme použiť v podmienkach `switch`

```
public enum OrderStatus {
    NEW, PAID, SHIPPED, DELIVERED
}

public static void sendNotification(OrderStatus status) {
    switch (status) {
        case NEW:
            System.out.println("Nová objednávka");
            break;
        case PAID:
            sendEmail("Platba prijatá");
            break;
        // ... Vo switchi musia byť všetky hodnoty enumu!
    }
}
```

Enum môže obsahovať dodatočné atribúty a metódy.

```
enum Planet {
  MERCURY(3.303e+23, 2.4397e6),
  EARTH(5.976e+24, 6.37814e6);

  private final double mass;
  private final double radius;

  Planet(double mass, double radius) {
    this.mass = mass;
    this.radius = radius;
  }
```

```
double getMass() {
  return mass;
}

double surfaceGravity() {
  final double G = 6.6E-11;
  return G * mass / (radius *
radius);
}


double g =
Planet.EARTH.surfaceGravity();
```

Enum v Java API

Java API obsahuje niekoľko `enum` tried, ktoré sa často používajú:

- java.time.Month - mesiace v roku - `JANUARY`, `MARCH`
- java.nio.file.StandardOpenOption - režim otvárania súboru - `READ`, `WRITE`
- java.util.concurrent.TimeUnit - Jednotka času - `DAYS`, `SECONDS`
- java.math.RoundingMode - spôsob zaokrúhľovania - `FLOOR`, `CEILING`