Zaawansowane programowanie obiektowe i funkcyjne

Wykład 2: Tryb wyliczeniowy

dr inż. Marcin Luckner
mluckner@mini.pw.edu.pl

Wydział Matematyki i Nauk Informacyjnych

Wersja 1.2
19 października 2021

Fundusze Europejskie Wiedza Edukacja Rozwój
Rzeczpospolita Polska
Politechnika Warszawska
Unia Europejska Europejski Fundusz Społeczny

Projekt „NERW 2 PW. Nauka – Edukacja – Rozwój – Współpraca” współfinansowany jest ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

Zadanie 10 pn. „Modyfikacja programów studiów na kierunkach prowadzonych przez Wydział Matematyki i Nauk Informacyjnych”, realizowane w ramach projektu „NERW 2 PW. Nauka – Edukacja – Rozwój – Współpraca”, współfinansowanego jest ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

Wyliczenie przy pomocy `int`

- Jeżeli mamy stały zbiór wartości, możemy chcieć je zadeklarować w czytelny sposób umożliwiający ich wielokrotne i wygodne użycie.
- Standardowym podejściem jest stworzenie grupy wartości `int`.

```
1  public static final int SEASON_WINTER = 0;
2  public static final int SEASON_SPRING = 1;
3  public static final int SEASON_SUMMER = 2;
4  public static final int SEASON_AUTUMN = 3;
```

- Jest to podejście lepsze niż przekazywanie samych wartości `int`, pozwala także na nadanie kolejności parametrom.
- Jednakże to rozwiązanie ma też szereg wad.

Wyliczenie przy pomocy `int` - wady

```
1  public static final int SEASON_WINTER = 0;
2  public static final int SEASON_SPRING = 1;
3  public static final int SEASON_SUMMER = 2;
4  public static final int SEASON_AUTUMN = 3;
```

- Wartości są interpretowane jako liczby, co może prowadzić do nieporozumień.

```
1  SEASON_AUTUMN - SEASON_SPRING == SEASON_SUMMER;
```

- Znaczenie wartości jest zakodowane tylko w nazwie zmiennej. Sama wartość jest prezentowana jako liczba całkowita.

```
1  System.out.println(SEASON_SUMMER);
2  //seasonToString(2);
```

- Nie mamy możliwości kontrolowania zakresu zmiennych. Nie możemy utworzyć funkcji, która będzie przyjmowała jako argument tylko wybrany podzbiór `int`.

```
1  String seasonToString(int season);
2  //seasonToString(5);
3  //seasonToString(MONTH_FEBRUARY);
```

Typ wyliczeniowy enum

- Java wprowadza typ wyliczeniowy enum, aby wyeliminować wymienione problemy.
- Typ wyliczeniowy zawiera stałe wyliczeniowe, określone jako statyczne i finalne pola.

```
1  public enum Season {
2      WINTER, SPRING, SUMMER, AUTUMN
3  }
```

- Nie jest już interpretowany jako int i możemy stosować kontrolę typu.

```
1  public static String seasonToString(Season season){
2    return season.toString();
3  }
```

- Jest wyświetlany użytkownikowi w przyjazny sposób.

```
1   System.out.println("The current season is
        "+seasonToString(Season.AUTUMN));
2   //The current season is AUTUMN
```

Typ wyliczeniowy - pola i metody

- Typ wyliczeniowy pozwala na dodawanie pól do elementów wyliczenia.
- Pozwala też na dopisywanie metod i implementację interfejsów.
- Wartości pól mogą być przekazywane przy pomocy konstruktora elementów wyliczenia.

Układ słoneczny

```
1  public enum Planet {
2    MERCURY(3.302e+23, 2.439e6),
3    VENUS (4.869e+24, 6.052e6),
4    EARTH (5.975e+24, 6.378e6),
5    MARS (6.419e+23, 3.393e6),
6    JUPITER(1.899e+27, 7.149e7),
7    SATURN (5.685e+26, 6.027e7),
8    URANUS (8.683e+25, 2.556e7),
9    NEPTUNE(1.024e+26, 2.477e7);
10
11   private final double mass; // kilo
12   private final double radius; // meters
13   private final double surfaceGravity; // [m / s^2]
14
15   private static final double G = 6.67300E-11;// universal gravity
         constant [m^3 / kg s^2]
16
17   Planet(double mass, double radius) { // constructor
18     this.mass = mass;
19     this.radius = radius;
20     surfaceGravity = G * mass / (radius * radius);
21   }
22
23   public double mass() { return mass; }
24   public double radius() { return radius; }
25   public double surfaceGravity() { return surfaceGravity; }
26   public double surfaceWeight(double mass) {
27     return mass * surfaceGravity; // F = ma
28   }
29 }
```

Układ słoneczny - zastosowanie

- Możemy wywoływać metody z trybu wyliczeniowego jak metody statyczne.

```
1  double weight = new Person("Neil Armstrong"){
2    public double getWeightinSpacesuit(){
3      return 163.3; //360lb
4    }
5  }.getWeightinSpacesuit();
6
7   for (Planet planet:Planet.values()) {
8    System.out.format("Neil Armstrong's surface weight on %s is
         %5.2fN%n",planet.toString(),planet.surfaceWeight(weight));
9  }
```

Wynik

```
Neil Armstrong's surface weight on MERCURY is 604,87N
Neil Armstrong's surface weight on VENUS is 1448,60N
Neil Armstrong's surface weight on EARTH is 1600,58N
Neil Armstrong's surface weight on MARS is 607,58N
Neil Armstrong's surface weight on JUPITER is 4048,94N
Neil Armstrong's surface weight on SATURN is 1705,44N
Neil Armstrong's surface weight on URANUS is 1448,29N
Neil Armstrong's surface weight on NEPTUNE is 1818,68N
```

Indeksowanie i wyszukiwanie wartości

- Do trybu wyliczeniowego możemy odwoływać się korzystając z indeksów.

```
1   System.out.println(Planet.values()[2]);
2   //EARTH
```

- Możemy także odwowyłać się do nazwy.

```
1   System.out.println(Planet.valueOf("MARS"));
2   //MARS
```

Działania matematyczne

- Możemy symulować działanie metod dynamicznych w zależności od elementu wyliczenia.

```
1  public enum Operation {PLUS, MINUS, TIMES, DIVIDE;
2
3    public double apply(double x, double y) {
4      switch(this) {
5        case PLUS: return x + y;
6        case MINUS: return x - y;
7        case TIMES: return x * y;
8        case DIVIDE: return x / y;
9      }
10   throw new AssertionError("Nieznana operacja: " + this);
11   }
12 }
```

- Lepszym podejściem jest zastosowanie abstrakcji.

```
1  public enum Operation {
2    PLUS { public double apply(double x, double y){return x + y;} },
3    MINUS { public double apply(double x, double y){return x - y;} },
4    TIMES { public double apply(double x, double y){return x * y;} },
5    DIVIDE { public double apply(double x, double y){return x / y;} };
6    public abstract double apply(double x, double y);
7  }
```

Działania matematyczne - zastosowanie

- Wywołanie dynamicznych operacji.

```
1  double x = 2;
2  double y = 5;
3
4  for (Operation operation:Operation.values()) {
5    System.out.format("%5.2f %s %5.2f =
         %5.2f%n",x,operation,y,operation.apply(x,y));
6  }
```

Wynik

```
2.00 PLUS  5.00 =  7.00
2.00 MINUS  5.00 = -3.00
2.00 TIMES  5.00 = 10.00
2.00 DIVIDE  5.00 =  0.40
```

Metoda ordinal

- Metoda `ordinal` zwraca liczbę porządkową.

```
1  public enum Ensemble {
2    SOLO, DUET, TRIO, QUARTET, QUINTET, SEXTET, SEPTET, OCTET, NONET,
         DECTET;
3
4    public int numberOfMusicians() { return ordinal() + 1; }
5  }
```

- Należy jednak uważać z utożsamianiem jej z wartością wyliczenia.

```
1  public enum Ensemble {
2    SOLO(1), DUET(2), TRIO(3), TERCET(3), QUARTET(4), QUINTET(5),
         SEXTET(6), SEPTET(7), OCTET(8), NONET(9), DECTET(10)
3
4    private final int numberOfMusicians;
5
6    Ensemble(int size) { this.numberOfMusicians = size; }
7
8    public int numberOfMusicians() { return numberOfMusicians; }
9  }
```

Pola bitowe

- Czasami chcemy przekazać więcej informacji w jednym parametrze.
- Możemy uzyskać to stosując sumę wartości binarnych.

```
1  public static final int STYLE_BOLD = 1 << 0; // 1
2  public static final int STYLE_ITALIC = 1 << 1; // 2
3  public static final int STYLE_UNDERLINE = 1 << 2; // 4
4  public static final int STYLE_STRIKETHROUGH = 1 << 3; // 8
5
6  public void applyStyles(int styles){
7  }
```

- Suma bitowa pozwala przekazać więcej wartości naraz `applyStyles(STYLE_BOLD | STYLE_ITALIC)`.

Pola bitowe - ograniczenia

- Metoda zachowuje wszystkie wady wyliczania przy pomocy `int`.
- Przy większej liczbie możliwych wartości używanie pola bitowego staje się kłopotliwe.
  - musimy przewidzieć maksymalną wartość parametru, aby dobrać typ zmiennej.
  - nie ma prostego sposobu iterowania wartości.

Typ EnumSet

- Alternatywnym rozwiązaniem jest zastosowanie typu wyliczeniowego i zdefiniowanie jego podzbioru.
- Podzbiór definiujemy używając typu `EnumSet`.

```
1  public enum Style { BOLD, ITALIC, UNDERLINE, STRIKETHROUGH }
2
3  public void applyStyles(Set<Style> styles){
4  }
```

- Wartości przekazujemy tworząc zbiór wyliczeń `applyStyles(EnumSet.of(Style.BOLD, Style.ITALIC))`.

Podzbiory

- Podzbiór możemy tworzyć poprzez wyliczenie elementów.

```
1  EnumSet<Planet> gasGiant = EnumSet.of(JUPITER, SATURN, URANUS,
       NEPTUNE);
2  //[JUPITER, SATURN, URANUS, NEPTUNE]
```

- Możemy także określić przedział.

```
1  EnumSet<Planet> innerPlanets = EnumSet.range(MERCURY,MARS);
2  //[MERCURY, VENUS, EARTH, MARS]
```

- Możemy sprawdzić przynależnosć elementu do podzbioru `innerPlanets.contains(MARS)`.

Wyliczanie wypłaty

```
1  public enum PayrollDay {
2          MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY,
3          SATURDAY, SUNDAY;
4
5      private static final EnumSet<PayrollDay> weekend =
           EnumSet.of(SATURDAY, SUNDAY);
6      private static final EnumSet<PayrollDay> workingDays =
           EnumSet.range(MONDAY, FRIDAY);
7      private static final double workingHours = 8;
8
9      double payout(double hours, double wage){
10         wage *= weekend.contains(this)?2:1; // double wage at
               weekends
11         hours+=workingDays.contains(this) &&
               hours>workingHours?hours-workingHours:0; // double hour
               of overtime
12         return hours*wage;
13     }
14 }
```

Projekt „NERW 2 PW. Nauka – Edukacja – Rozwój – Współpraca” współfinansowany jest ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

Zadanie 10 pn. „Modyfikacja programów studiów na kierunkach prowadzonych przez Wydział Matematyki i Nauk Informacyjnych”, realizowane w ramach projektu „NERW 2 PW. Nauka – Edukacja – Rozwój – Współpraca”, współfinansowanego jest ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.