# 1

# 1 Перечисления

1. Перечислимый тип `enum`
2. Статический импорт

# «МАГИЧЕСКИЕ ЧИСЛА» И КЛАССИЧЕСКОЕ ИСПОЛЬЗОВАНИЯ КОНСТАНТ

Практически в любой программе возникает необходимость использования числовых величин. Если эти значения использовать в явном виде, то возникает проблема «магических чисел» - т.е. чисел, назначение которых неясно из кода программы. Например:

```
switch(nn % 7) {
  case 0: …;
  case 6: …;
 . . .
}
```

Что могут означать «магические числа» 0 и 6? День недели? Номера ячеек? …? Прояснить ситуацию достаточно легко с помощью определения числовых констант:

```
public abstract class DayType {
  public static final int SUNDAY = 0;
  public static final int SATURDAY = 6;
}  …
switch(nn % 7) {
  case DayType.SUNDAY: …;
  case DayType.SATURDAY : …;
 . . .
}
```

Здесь код стал уже достаточно явный. Тем не менее, явно передавать значения для обозначения дня недели строго из интервала 0..6 мы всё же НЕ сможем – придётся использовать целый тип, гораздо более «широкий» чем надо.

# ПРОБЛЕМЫ КЛАССИЧЕСКОГО ИСПОЛЬЗОВАНИЯ КОНСТАНТ

- Небезопасность (type unsafe): в методе, где требуется передать значение перечисления, можно передать любое число, а не только 0..6
- Низкая информативность: например, при отладке вывод элемента предоставит нам число, связь которого с названием придется искать
- Подверженность ошибкам: например, при добавлении нового элемента или при изменении последовательности существующих элементов

# ПЕРЕЧИСЛИМЫЙ ТИП ENUM

Для решения этой проблемы в версии Java SE 5 был введен новый перечислимый тип - тип, значения которого ограничены конечным набором констант.

Тип `enum` для фиксированного набора размеров деталей можно описать следующим образом:

```
package org.academyit;

enum DetailSize {
   TINY, SMALL, NORMAL, BIG, HUGE
 };
```

Создав новое перечисление (`DetailSize`) предварительно определенного типа, можно использовать его при описании классов аналогично любой другой переменной экземпляра. Естественно, перечислению можно присвоить только одно из перечисленных значений (в нашем случае `TINY`, `SMALL`, `NORMAL`, `BIG` или `HUGE`).

# ПЕРЕЧИСЛИМЫЙ ТИП ENUM (ПРОДОЛЖЕНИЕ)

```
package org.academyit;

public class Detail {

  private String name;
  private double weight;
  private DetailSize size;

 public void assignSize( DetailSize p_size ) {
    size = p_size;
  }
 public DetailSize getSize() {
    return size;
  }
  // . . . прочие методы
}
```

В методе `assignSize()` не производится проверки введения неправильного значения, поскольку ввести значение, отличающееся от заданного в типе `enum DetailSize` невозможно.

# ВЫВОД ВСЕХ ДОПУСТИМЫХ ЗНАЧЕНИЙ ДЛЯ ОБЪЕКТА ТИПА ENUM

Следующий пример позволяет осуществить вывод всех допустимых значений для типа `DetailSize`:

```
public class Test1 {
 public static void main(String[] args) {
   for( DetailSize s : DetailSize.values() ) {
      System.out.println("Allowed value: '" + s + "'");
    }
  }
}
```

Результат:

```
Allowed value: 'TINY'
Allowed value: 'SMALL'
Allowed value: 'NORMAL'
Allowed value: 'BIG'
Allowed value: 'HUGE'
```

Метод `values()` возвращает массив индивидуальных экземпляров перечисления `DetailSize`, каждый с одним из перечислимых значений, то есть данный метод вернул `DetailSize[]`.

# СТАТИЧЕСКИЙ ИМПОРТ

Чаще всего перечисление используется в операторах `switch` для выбора некоторых действий в зависимости от значения переменной.

Обратите внимание на использование статического импорта (оператор `import static org.academyit.DetailSize.*;`).

Хотя тип `DetailSize` и класс `Test2` находятся в одном пакете и импорт не требуется, мы его сделали с ключевым словом `static`. Такой импорт позволяет писать в клаузе `case` вместо `DetailSize.TINY` просто `TINY`.
Этим механизмом злоупотреблять не стоит, так как он может снизить читабельность текста, но в некоторых случаях он упрощает текст.

# ПРИМЕР ИСПОЛЬЗОВАНИЯ ПЕРЕЧИСЛИМОГО ТИПА

```
import static org.academyit.DetailSize.*;

public class Test2 {

 public static void main(String[] args) {

   Detail d = new Detail( "shaft", 12.9 );
   d.assignSize( DetailSize.SMALL );
   switch( d.getSize() ) {
    case TINY:
      System.out.println("Detail size is "
                         + d.getSize().toString());
      break;
    case SMALL:
      System.out.println("Detail size is "
                         + d.getSize().toString());
      break;
    . . .
```

# ПРИМЕР ИСПОЛЬЗОВАНИЯ ПЕРЕЧИСЛИМОГО ТИПА (ПРОДОЛЖЕНИЕ)

```
 case NORMAL:
      System.out.println("Detail size is "
                         + d.getSize().toString());
      break;
    case BIG:
      System.out.println("Detail size is "
                         + d.getSize().toString());
      break;
    case HUGE:
      System.out.println("Detail size is "
                         + d.getSize().toString());
      break;
    default:
      System.out.println("Detail size is unknown !");
    }
   System.out.println("----------------------------");
   System.out.println( d );
  }
}
```

Результат:

```
Detail size is SMALL
----------------------------
 Detail = shaft weight = 12.9 size = SMALL
```

# ДОПОЛНИТЕЛЬНЫЕ ВОЗМОЖНОСТИ ПЕРЕЧИСЛИМЫХ ТИПОВ

Перечислимые типы могут иметь атрибуты и методы:

```
package org.academyit;

enum DetailSize {
  TINY ("Крошечный"),
  SMALL ("Маленький"),
  NORMAL ("Нормальный"),
  BIG ("Большой");
  HUGE ("Огромный");

  private final String name;
  private DetailSize( String name ) {
    this.name = name;
  }
  public String getName() {
    return name;
  }
 }
```

Получить доступ к этим атрибутам можно следующим способом:

```
Detail d = new Detail( "shaft", 12.9 );
d.assignSize( DetailSize.SMALL );
 . . .
String name =  d.getSize().getName();
System.out.println( name );
```

Результат: **Маленький**

Одной из особенностей использования перечислимого типа является то, что при выводе на печать мы получаем вместо цифр текстовое значение.

# ИТОГИ

**В теме рассмотрено:**

- Перечислимый тип `enum`
- Статический импорт