```
            printArgCount(5, 7);
            // printArgCount();//неопределенность!
      }
}
```

В результате будет выведено:

```
Object args: 4
Integer[] args: 3
Object args: 3
Object args: 5
int args: 2
```

При передаче в метод **printArgCount()** единичного массива **i** компилятор отдает предпочтение методу с параметром **Object... args**, так как имя массива является объектной ссылкой и потому указанный параметр будет ближайшим. Метод с параметром **Integer[]...args** не вызывается, так как ближайшей объектной ссылкой для него будет **Object[]...args**. Метод с параметром **Integer[]...args** будет вызван для единичного массива только в случае отсутствия метода с параметром **Object...args**.

При вызове метода без параметров возникает неопределенность из-за невозможности однозначного выбора.

Не существует также ограничений и на переопределение подобных методов.

Единственным ограничением является то, что параметр вида **Тип...args** должен быть последним в объявлении метода, например:

```
void methodName(Тип1 obj, Тип2... args) {}
```

Перечисления

Типобезопасные перечисления (typesafe enums) в Java представляют собой классы и являются подклассами абстрактного класса **java.lang.Enum**. При этом объекты перечисления инициализируются прямым объявлением без помощи оператора **new**. При инициализации хотя бы одного перечисления происходит инициализация всех без исключения оставшихся элементов данного перечисления.

В качестве простейшего применения перечисления можно рассмотреть следующий код:

```
/* пример # 16 : применение перечисления: SimpleUseEnum.java */
package chapt02;

enum Faculty {
      MMF, FPMI, GEO
}
public class SimpleUseEnum {
      public static void main(String args[]) {
            Faculty current;
            current = Faculty.GEO;
            switch (current) {
              case GEO:
                  System.out.print(current);
                  break;
              case MMF:
```

```
                System.out.print(current);
                break;
          // case LAW : System.out.print(current);//ошибка компиляции!
            default:
           System.out.print("вне case: " + current);
          }
    }
}
```

В операторах **case** используются константы без уточнения типа перечисления, так как его тип определен в **switch**.

Перечисление как подкласс класса **Enum** может содержать поля, конструкторы и методы, реализовывать интерфейсы. Каждый тип **enum** может использовать методы:

static enumType[] values() – возвращает массив, содержащий все элементы перечисления в порядке их объявления;

static T valueOf(Class<T> enumType, String arg) – возвращает элемент перечисления, соответствующий передаваемому типу и значению передаваемой строки;

static enumType valueOf(String arg) – возвращает элемент перечисления, соответствующий значению передаваемой строки;

int ordinal() – возвращает позицию элемента перечисления.

```
/* пример # 17 : объявление перечисления с методом : Shape.java */
package chapt02;

enum Shape {
      RECTANGLE, TRIANGLE, CIRCLE;
      public double square(double x, double y) {
            switch (this) {
            case RECTANGLE:
                  return x * y;
            case TRIANGLE:
                  return x * y / 2;
            case CIRCLE:
                  return Math.pow(x, 2) * Math.PI;
            }
            throw new EnumConstantNotPresentException(
                  this.getDeclaringClass(),this.name());
      }
}
```

```
/* пример # 18 : применение перечисления: Runner.java */
package chapt02;

public class Runner {
      public static void main(String args[]) {
            double x = 2, y = 3;
            Shape[] arr = Shape.values();
```

```
            for (Shape sh : arr)
                  System.out.printf("%10s = %5.2f%n",
                              sh, sh.square(x, y));
      }
}
```

В результате будет выведено:

```
RECTANGLE =  6,00
 TRIANGLE =  3,00
   CIRCLE = 12,57
```

Каждый из элементов перечисления в данном случае представляет собой в том числе и арифметическую операцию, ассоциированную с методом **square()**. Без **throw** данный код не будет компилироваться, так как компилятор не исключает появления неизвестного элемента. Данная инструкция позволяет указать на возможную ошибку при появлении необъявленной фигуры. Поэтому и при добавлении нового элемента необходимо добавлять соответствующий ему **case**.

```
/* пример # 19 : конструкторы и члены перечисления: DeanDemo.java */
package chapt02;

enum Dean {
   MMF("Бендер"), FPMI("Балаганов"), GEO("Козлевич");
      String name;

      Dean(String arg) {
            name = arg;
      }
      String getName() {
            return name;
      }
}
package chapt02;

public class DeanDemo {
  public static void main(String[] args) {
    Dean dn = Dean.valueOf("FPMI");
    System.out.print(dn.ordinal());
    System.out.println(" : " + dn + " : " + dn.getName());
      }
}
```

В результате будет выведено:

```
1 : FPMI : Балаганов
```

Однако на перечисления накладывается целый ряд ограничений.

Им запрещено:

- быть суперклассами;
- быть подклассами;
- быть абстрактными;
- создавать экземпляры, используя ключевое слово **new**.