Аннотации

Аннотации – мета-теги, которые добавляются к коду и применяются к объявлению пакетов, типов, конструкторов, методов, полей, параметров и локальным переменным. В результате можно задать зависимость, например, метода от другого метода. Аннотации позволяют избежать создания шаблонного кода во многих ситуациях, активируя утилиты для его генерации из аннотаций в исходном коде.

В следующем коде приведено объявление аннотации.

```
/* пример # 20 : многочленная аннотация : RequestForCustomer.java */
package chapt03;

public @interface RequestForCustomer {
      int level();
      String description();
      String date();
}
```

Ключевому слову **interface** предшествует символ **@**. Такая запись сообщает компилятору об объявлении аннотации. В объявлении также есть три метода-члена: **int level()**, **String description()**, **String date()**.

Все аннотации содержат только объявления методов, добавлять тела этим методам не нужно, так как их реализует сам язык. Кроме того, эти методы не могут содержать параметров, секции **throws** и действуют скорее как поля. Допустимые типы возвращаемого значения: базовые типы, **String**, **Enum**, **Class** и массив любого из вышеперечисленных типов.

Все типы аннотаций автоматически расширяют интерфейс **java.lang.annotation.Annotation**. В этом интерфейсе даны методы: **hashCode()**, **equals()** и **toString()**, определенные в типе **Object**. В нем также приведен метод **annotationType()**, который возвращает объект типа **Class**, представляющий вызывающую аннотацию.

После объявления аннотации ее можно использовать для аннотирования объявлений. Объявление любого типа может иметь аннотацию, связанную с ним. Например, можно снабжать примечаниями классы, методы, поля, параметры и константы типа **enum**. Даже к аннотации можно добавить аннотацию. Во всех случаях аннотация предшествует объявлению.

Применяя аннотацию, нужно задавать значения для ее методов-членов. Далее приведен фрагмент, в котором аннотация **RequestForCustomer** сопровождает объявление метода:

```
@RequestForCustomer (
   level = 2,
   description = "Enable time",
   date = "10/10/2007"
)
public void customerThroughTime() {
   //...
}
```

Данная аннотация связана с методом **customerThroughTime()**. За именем аннотации, начинающимся с символа @, следует заключенный в круглые скобки список инициализирующих значений для методов-членов. Для того чтобы передать значение методу-члену, имени этого метода присваивается значение. Таким образом, в приведенном фрагменте строка **"Enable time"** присваивается методу **description()**, члену аннотации типа **RequestForCustomer**. При этом в присваивании после имени **description** нет круглых скобок. Когда методу-члену передается инициализирующее значение, используется только имя метода. Следовательно, в данном контексте методы-члены выглядят как поля.

```
/* пример # 21 : применение аннотации: Request.java */
package chapt03;
import java.lang.reflect.Method;

public class Request {
      @RequestForCustomer(level = 2,
                          description = "Enable time",
                          date = "10/10/2007")
      public void customerThroughTime() {
            try {
            Class c = this.getClass();
            Method m = c.getMethod("customerThroughTime");
            RequestForCustomer ann =
            m.getAnnotation(RequestForCustomer.class);
                  //запрос аннотаций
                  System.out.println(ann.level() + " "
                              + ann.description() + " "
                              + ann.date());
            } catch (NoSuchMethodException e) {
                  System.out.println("метод не найден");
            }
      }
      public static void main(String[] args) {
            Request ob = new Request();
            ob.customerThroughTime();
      }

   }
```

В результате будет выведено:

```
2 Enable time 10/10/2007
```

Если аннотация объявляется в отдельном файле, то ей нужно задать правило сохранения **RUNTIME** в виде кода

```
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
@Retention(RetentionPolicy.RUNTIME) //правило сохранения
```

помещаемого перед объявлением аннотации, которое предоставляет максимальную продолжительность существования аннотации.

С правилом **SOURCE** аннотация существует только в исходном тексте программы и отбрасывается во время компиляции.

Аннотация с правилом сохранения **CLASS** помещается в процессе компиляции в файл **.class**, но не доступна в JVM во время выполнения.

Аннотация, заданная с правилом сохранения **RUNTIME**, помещается в файл **.class** в процессе компиляции и доступна в JVM во время выполнения. Следовательно, правило **RUNTIME** предлагает максимальную продолжительность существования аннотации.

Основные типы аннотаций: аннотация-маркер, одночленная и многочленная.

Аннотация-маркер не содержит методов-членов. Цель – пометить объявление. В этом случае достаточно присутствия аннотации. Поскольку у интерфейса аннотации-маркера нет методов-членов, достаточно определить наличие аннотации.

```
public @interface TigerAnnotation {}
```

Для проверки наличия аннотации-маркера используется метод **isAnnotationPresent()**.

Одночленная аннотация содержит единственный метод-член. Для этого типа аннотации допускается краткая условная форма задания значения для метода-члена. Если есть только один метод-член, то просто указывается его значение при создании аннотации. Имя метода-члена указывать не нужно. Но для того чтобы воспользоваться краткой формой, следует для метода-члена использовать имя **value()**.

Многочленные аннотации содержат несколько методов-членов. Поэтому используется полный синтаксис (**имя_параметра = значение**) для каждого параметра.

В языке Java определено семь типов встроенных аннотаций, четыре типа – **@Retention**, **@Documented**, **@Target** и **@Inherited** – импортируются из пакета **java.lang.annotation**. Оставшиеся три – **@Override**, **@Deprecated** и **@Suppresswarnings** – из пакета **java.lang**.

Аннотации получают все более широкое распространение и активно используются в различных технологиях.

Задания к главе 3

Вариант А

1. Определить класс **Вектор** размерности n. Реализовать методы сложения, вычитания, умножения, инкремента, декремента, индексирования. Определить массив из m объектов. Каждую из пар векторов передать в методы, возвращающие их скалярное произведение и длины. Вычислить и вывести углы между векторами.
2. Определить класс **Вектор** размерности n. Определить несколько конструкторов. Реализовать методы для вычисления модуля вектора, скалярного произведения, сложения, вычитания, умножения на константу. Объявить массив объектов. Написать метод, который для заданной пары векторов будет определять, являются ли они коллинеарными или ортогональными.