```
/*
 * 枚举：列举，罗列
 *
 * 枚举是代表这样的一系列的类型，这些类型有一个非常明显的特征：他们的对象是有限的几个。
 * 例如：Week（星期），我可以限定Week类型的对象只有7个：Monday,Tuesday,Wednesday,Thursday,Friday,Saturday,Sunday
 *      Season（季节），可以限定对象为4个：Spring，Summber,Autumn，Winter
 *      Status（员工状态），可以限定为4个：空闲（Free），忙（Busy），休假（Vocation），离职（Left）
 *
 * 枚举类是JDK1.5之后引入的。
 *
 *
 */
```

```
/*
 * JDK1.5之前：如何实现枚举的需求
 *
 * 要点：整个系统中，某个类型的对象是有限的几个，不多也不少。
 * （1）限制使用者随意的创建对象-->构造器私有化
 * （2）把有限的几个对象，预先创建好，放着，供使用者使用
 */
```

```
* 要点：整个系统中，某个类型的对象是有限的几个，不多也不少。
* （1）限制使用者随意的创建对象-->构造器私有化
* （2）把有限的几个对象，预先创建好，放着，供使用者使用-->在枚举类中，用静态的类变量把有限的几个对象存储起来，使用者通过“类名.对象名”来获取
*/
```

```
/*
 * 枚举：列举，罗列
 *
 * 枚举是代表这样的一系列的类型，这些类型有一个非常明显的特征：他们的对象是有限的几个。
 * 例如：Week（星期），我可以限定Week类型的对象只有7个：Monday,Tuesday,Wednesday,Thursday,Friday,Saturday,Sunday
 *      Season（季节），可以限定对象为4个：Spring，Summber,Autumn，Winter
 *      Status（员工状态），可以限定为4个：空闲（Free），忙（Busy），休假（Vocation），离职（Left）
 *
 * 枚举类是JDK1.5之后引入的。
 *
 * 枚举类型也是类，原来用class声明类，现在用enum来声明枚举。
 *
 *
 */
```

```
* 枚举类是JDK1.5之后引入的。
*
* 枚举类型也是类，原来用class声明类，现在用enum来声明枚举。
* 声明的语法格式：
* 【权限修饰符】 enum 枚举类型名{
*       常量对象列表;
* }
*
* 说明：
* （1）枚举类型的构造器一定是私有的
* （2）枚举类型的常量对象列表必须在枚举类的首行。
*
*
*
* 首行：
* （1）package
* （2）super(),this()
* （3）枚举常量对象
*
```

```
* 说明：
* （1）枚举类型的构造器一定是私有的
* （2）枚举类型的常量对象列表必须在枚举类的首行。
*       如果常量对象列表后面没有其他成员，那么;可以省略，如果后面还有其他的成员，;不能省略。
* （3）所有枚举类型有一个直接父类java.lang.Enum类型，所以你不能在继承其他类
*
*
```

```
* 说明：
* （1）枚举类型的构造器一定是私有的
* （2）枚举类型的常量对象列表必须在枚举类的首行。
*       如果常量对象列表后面没有其他成员，那么;可以省略，如果后面还有其他的成员，;不能省略。
* （3）所有枚举类型有一个直接父类java.lang.Enum类型，所以你不能在继承其他类
* （4）switch...case
*
*
* 首行：
* （1）package
* （2）super(),this()
* （3）枚举常量对象
*
* switch...case的表达式的类型：
* （1）基本数据类型：byte,short,int,char
* （2）引用数据类型：枚举（JDK1.5之后），String（JDK1.7之后）
*
```

```
* java.lang.Enum类，它是所有 Java 语言枚举类型的公共基本类。
* 即所有枚举类型都继承它，不能在继承别的类型。
* （1）name()：返回此枚举常量的名称
* （2）ordinal()：返回枚举常量的序数（它在枚举声明中的位置，其中初始常量序数为零）。
* （3）int compareTo(x)：默认按照枚举对象的顺序排序
* （4）API中没有，编译器自动生成的
*  枚举类型[] values()
*  枚举类型 valueOf(常量名)
*/
```

```
/*
 * 注解：
 * 长什么样？ @注解名
 *
 * 注解是什么？
 *    注解也是注释，这是代码级别的注释，用代码给代码注释
 *
 * 注解有三个部分组成？
 * （1）声明注解，定义注解
 * 我们开发中，绝大多数都是别人定义好的。
 * （2）使用注解（重要）
 * 我们开发中，主要是这步
 * （3）读取注解的信息
 * 我们把读取注解信息的代码称为“注解信息处理流程”，如果没有（3）前面两步都没有意义
 * 读取注解的信息的代码基本上也是别人写好的，
 * 读取注解信息的代码需要“反射”知识。
 */
```

```
* 常见的注解：
* 一、系统预定义的三个最最基本的注解
* 1、@Override
* 它是由JDK的核心类库定义，读取它是由编译器，例如：javac.exe。
* 作用：注释这个方法是一个“重写”的方法，让编译器对这个方法的签名进行格式检查，是否满足“重写”的要求。
*
* 重写的要求：
* （1）方法名称和形参列表必须相同
* （2）返回值类型：
*       基本数据类型和void：必须相同
*       引用数据类型：<=
* （3）权限修饰符：>=
```

```
* 重写的要求：
* （1）方法名称和形参列表必须相同
* （2）返回值类型：
*       基本数据类型和void：必须相同
*       引用数据类型：<=
* （3）权限修饰符：>=
*
* 哪些方法不能重写：
* （1）static
* （2）final
* （3）private
*
```

```
* 2、@SuppressWarnings(xxx)
* 它是由JDK的核心类库定义，读取它是由编译器，例如：javac.exe。
* 作用：抑制警告
*
*/
```

```
* 3、@Deprecated
* 它是由JDK的核心类库定义，读取它是由编译器，例如：javac.exe。
* 作用：告知编译器和程序员这个方法、属性、类等已过时，不建议再使用，如果使用出了问题，自己负责。
*
```

```
/*
 * Java中的注释：
 * (1)单行注释
 * (2)多行注释
 * (3)文档注释，Java特有
 * 需要配合注解使用
 *
 *
 * 二、用于文档注释的注解
 */
```

```
 * 二、用于文档注释的注解
 * @author
 * @version
 * @since
 * ...
 *
 * @param
 *  (1)必须该方法有形参，才能写，有几个形参，写几个
 *  (2)@param 形参名 形参数据类型 解释
 * @return
 *  (1)必须方法的返回值类型不是void，一个方法@return最多有一个
 *  (2)@return 返回值类型 解释
 * @throws
 *  (1)必须方法throws异常，有几个写几个
 *  (2)@throws 类型 解释
 *
 */
```

```
/*
 * 域名：
 * com,edu,gov,org...
 *
 * 三、单元测试相关的注解
 * 1、声明：第三方回归测试框架声明，不是JDK声明
 * 2、读取：有JUnit框架来读取
 *
 * 3、使用
 * （1）引入第三方回归测试框架jar（一堆类的class文件）
 * 目前的IDE（Eclipse，IDEA）都有集成
 *
 * 项目名上右键->Build Path->Add Library-->JUnit-->选择版本-->finish
 *
 * （2）可以使用的注解（在方法上）
 * @Test
 *  Run as ->Junit Test 单元测试
 */
```

```
* 白盒测试：程序自测，需要明确知道测试的代码的编写，功能...
* 黑盒测试：测试人员，不需要知道功能如何实现，只对着用户的需求文档，性能要求，安全要求进行各种测试
```

```
（2）可以使用的注解（在方法上）
@Test
 运行：
 （1）Run as ->Junit Test 单元测试
 （2）如果没有选择方法，那么当前类的所有的@Test标记的方法都会执行；如果选择了其中一个方法，那么就只运行这一个
```

运行：
（1）Run as ->Junit Test 单元测试
（2）如果没有选择方法，那么当前类的所有的@Test标记的方法都会执行；如果选择了其中一个方法，那么就只运行这一个
要求：
用@Test标记的方法有要求：JUnit4版本
（1）这个方法本身必须是public，void，()
（2）这个方法所在的类也得是public

```
* 了解：
* @Before：在每一个@Test标记的方法之前运行。
* @After：在每一个@Test标记的方法之后运行。
* @BeforeClass：在当前类初始化时执行，只执行一次，方法必是static
* @AfterClass：所有的方法之后，只执行一次，方法必须是static
*
```

```
/*
 * 自定义注解：
 * （1）声明
 * （2）使用
 * （3）读取
 *
 * 一、声明
 * 格式：
 * 【修饰符】@interface 注解名{
 * }
 */
```

```
/*
 * 五、配置参数
 * 1、如何声明配置参数
 * 格式：
 * 【权限修饰符】@interface 注解名{
 *       数据类型 配置参数名1();
 *       数据类型 配置参数名2();
 * }
 *
 * 2、配置参数的赋值
 * (1)如果注解声明的配置参数，那么在使用这个注解时，要求给这个配置参数赋值
 * 标准的赋值格式：@注解名(配置参数名1 = 参数值1,配置参数名2 = 参数值2...)
 * (2)如果配置参数的个数只有一个，并且名称是value，那么可以省略“配置参数名=”
 *
 */
```

2、配置参数的赋值
(1)如果注解声明的配置参数，那么在使用这个注解时，要求给这个配置参数赋值
标准的赋值格式：@注解名(配置参数名1 = 参数值1,配置参数名2 = 参数值2...)
(2)如果配置参数的个数只有一个，并且名称是value，那么可以省略“配置参数名=”
(3)配置参数如果有默认值，那么可以不赋值

总结：default
(1)switch
(2)接口默认方法
(3)自定义注解的配置参数默认值

3、配置参数的类型有要求：
类型只能是八种基本数据类型、String类型、Class类型、enum类型、Annotation类型、以上所有类型的数组

```
/*
 * import导包：
 * （1）import 包.类名;
 * （2）import 包.*;
 * （3）import static 包.类名.静态成员;
 * （4）import static 包.类名.*;
 */
```