一、枚举介绍

1、简介

1.1、枚举对应英文(enumeration，简写enum)
1.2、枚举是一组常量的集合
1.3、枚举属于一种特殊的类，里面只包含一组有限的特定的对象

二、自定义类实现枚举

1、实现细节

1.1、构造器私有化
1.2、本类内部创建一组对象
1.3、对外暴露对象（通过为对象添加 public final static 修饰符）
1.4、可以提供get方法，但不能提供set方法

三、enum关键字实现枚举

1、实现细节

1.1、使用关键字 enum 替代 class
1.2、传统的 public static final Season2 SPRING = new Season2("春天", "温暖"); 简化成 SPRING("春天", "温暖"); 这里需要注意的是，它调用的是哪个构造器
1.3、如果有多个常量（对象），使用 ,号间隔
1.4、如果使用 enum 来实现枚举，要求将定义常量对象，写在前面
1.5、如果使用无参构造器创建常量对象，则可以忽略()不写

2、常用方法

2.1、toString：Enum类已经重写过了，返回的是当前对象名，子类可以重写该方法，用于返回对象的属性信息
2.2、name：返回当前对象名（常量名），子类中不能重写
2.3、ordinal：返回当前对象的位置号，默认从0开始
2.4、values：返回当前枚举中所有的常量
2.5、valueOf：将字符串转换成枚举对象，要求字符串必须为已有的常量名，否则报异常
2.6、compareTo：比较两个枚举常量，比较的就是位置号

3、enum实现接口

3.1、使用enum关键字后，就不能再继承其他类了，因为enum会隐式继承Enum，而Java是但继承机制
3.2、枚举类和普通类一样，可以实现接口

四、JDK内置的基本注解类型

1、简介

注解（Annotation）也被称为元数据（Metadata），用于修饰解释 包、类、方法、属性、构造器、局部变量等数据信息。

和注释一样，注解不影响程序逻辑，但注解可以被编译或运行，相当于嵌入在代码中的补充信息。

2、基本注解

使用Annotation时要在其前面增加 @ 符号，并把该 Annotation当成一个修饰符使用

1) @Override：限定某个方法，是重写父类方法，该注解只能用于方法中

2) @Deprecated：用于表示某个程序元素（类、方法等）已过时

3) @SuppressWarnings：抑制编译器警告

3、@Override-语法

```
class Father {
    public void fly() {
        System.out.println("Father fly...");
    }
}
class Son extends Father {
    @Override
    public void fly() {
        System.out.println("Son fly...");
    }
}
```

4、@Override-细节

4.1、@Override 表示指定重写父类的方法（从编译层面验证），如果父类没有fly方法，则会报错

4.2、如果不写 @Override 注解，而父类仍有 public void fly(){}，仍然构成重写

4.3、@Override 只能修饰方法，不能修饰其他类、包、属性等等

4.4、查看 @Override 注解源码为 @Target(ElementType.METHOD)，说明只能修饰方法

4.5、@Target 是修饰注解的注解，称为元注解

5、@Deprecated-细节

5.1、@Deprecated 用于表示某个程序元素（类、方法等）已经过时

5.2、可以修饰方法、类、字段、包、参数 等等

5.3、@Target(value = {CONSTRUCTOR, FIELD, LOCAL_VARIABLE, METHOD, PACKAGE, PARAMETER, TYPE})

5.4、@Deprecated 的作用可以做到新旧版本的兼容和过渡

6、@SuppressWarnings-细节

```
6.1、unchecked 是忽略没有检查的警告
6.2、rawtypes 是忽略没有指定泛型的警告（传参时没有指定泛型的警告错误）
6.3、unused 是忽略没有使用某个变量的警告错误
6.4、@SuppressWarnings 可以修饰的程序元素为（查看源码中@Target）
6.5、生成 @SuppressWarnings 时，不用背，直接点击右侧的黄色提示，就可以选择
```

五、元注解：对注解进行注解

1、简介

```
JDK的元Annotation 用于修饰其他Annotation。
```

2、分类

```
2.1、@Retention：指定注解的作用范围：SOURCE、CLASS、RUNTIME
2.2、@Target：指定注解可以在哪些地方使用
2.3、@Documented：指定该注解是否会在javadoc体现
2.4、@Inherited：子类会继承父类注解
```

3、@Retention

```
# 简介
    只能用于修饰一个Annotation定义，用于指定该Annotation可以保留多长时间，@Retention包含一个RetentionPolicy类型的成员变量，使用@Retetion时必须为该value成员变量指定值。
```

```
# 值
    1) RetentionPolicy.SOURCE：编译器使用后，直接丢弃这种策略的注解
    2) RetentionPolicy.CLASS：编译器将把注解记录在Class文件中，当运行Java程序时，JVM不会保留注解。这也是默认值
    3) RetentionPolicy.RUNTIME：编译器将把注解记录在class文件中，当运行Java程序时，JVM会保留注解，程序可以通过反射获取该注解
```

4、@Target

```
# 简介
    用于修饰Annotation定义，用于指定被修饰的Annotation能用于修饰哪些程序元素。@Target也包含一个名为value的成员变量。
```

5、@Documented

```
# 简介
    用于指定被该元Annotation修饰的Annotation类将被Javadoc工具提取成文档，即在生成文档时可以看到该注解。
```

6、@Inherited

```
# 简介
    被它修饰的Annotation将具有继承性，如果某个类使用了被@Inherited修饰的Annotation，则其子类将自动具有该注解
```