第 11 章枚举和注解

11. 1 先看一个需求

要求创建季节(Season) 对象，请设计并完成。Enumeration01.java

class Season{//类

private String name;

private String desc;//描述

//构造器

//getXX

//setXX

}

代码:

|  |
| --- |
| //package com.hspedu.enum\_;  ///\*\*  // \* @author 韩顺平  // \* @version 1.0  // \*/  //public class Enumeration01 {  // public static void main(String[] args) {  // //使用  // Season spring = new Season("春天", "温暖");  // Season winter = new Season("冬天", "寒冷");  // Season summer = new Season("夏天", "炎热");  // Season autumn = new Season("秋天", "凉爽");  //// autumn.setName("XXX"); |

|  |
| --- |
| //// autumn.setDesc("非常的热..");  // //因为对于季节而已，他的对象(具体值) ，是固定的四个，不会有更多  // //安老师的这个设计类的思路，不能体现季节是固定的四个对象  // //因此，这样的设计不好===> 枚举类[枚: 一个一个 举： 例举 , 即把具体的对象一个一个例举出来的类  // // 就称为枚举类]  // Season other = new Season("红天", "~~~");  // }  //}  //class Season{//类  // private String name;  // private String desc;//描述  //  // public Season(String name, String desc) {  // this.name = name;  // this.desc = desc;  // }  //  // public String getName() {  // return name;  // }  //  // public void setName(String name) {  // this.name = name;  // }  //  // public String getDesc() { |

|  |
| --- |
| // return desc;  // }  //  // public void setDesc(String desc) {  // this.desc = desc;  // }  //} |

11.2 分析问题

11.2.1 创建 Season 对象有如下特点

1) 季节的值是有限的几个值(spring, summer, autumn, winter)

2) 只读，不需要修改。

11.3 解决方案-枚举

1) 枚举对应英文(enumeration, 简写 enum)

2) 枚举是一组常量的集合。

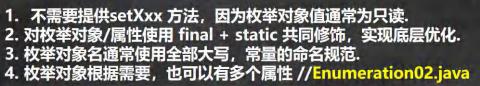
3) 可以这里理解：枚举属于一种特殊的类，里面只包含一组有限的特定的对象。

11.4 枚举的二种实现方式

1) 自定义类实现枚举

2) 使用 enum 关键字实现枚举

11.5 自定义类实现枚举-应用案例



代码:

|  |
| --- |
| package com.hspedu.enum\_; |

|  |
| --- |
| /\*\*  \* @author 韩顺平  \* @version 1.0  \*/  public class Enumeration02 {  public static void main(String[] args) {  System.out.println(Season.AUTUMN);  System.out.println(Season.SPRING);  }  }  //演示字定义枚举实现  class Season {//类  private String name;  private String desc;//描述  //定义了四个对象, 固定.  public static final Season SPRING = new Season("春天", "温暖");  public static final Season WINTER = new Season("冬天", "寒冷");  public static final Season AUTUMN = new Season("秋天", "凉爽");  public static final Season SUMMER = new Season("夏天", "炎热");  //1. 将构造器私有化, 目的防止 直接 new  //2. 去掉 setXxx 方法, 防止属性被修改 |

|  |
| --- |
| //3. 在 Season 内部，直接创建固定的对象  //4. 优化，可以加入 final 修饰符  private Season(String name, String desc) {  this.name = name;  this.desc = desc;  }  public String getName() {  return name;  }  public String getDesc() {  return desc;  }  @Override  public String toString() {  return "Season{" +  "name='" + name + '\'' +  ", desc='" + desc + '\'' +  '}';  }  } |

11.6 自定义类实现枚举-小结

11.6.1 小结：进行自定义类实现枚举，有如下特点：

1) 构造器私有化

2) 本类内部创建一组对象[四个 春夏秋冬]

3) 对外暴露对象（通过为对象添加 public final static 修饰符）

4) 可以提供 get 方法，但是不要提供 set

11.7 enum 关键字实现枚举-快速入门

11.7.1 说明

使用 enum 来实现前面的枚举案例，看老师演示，主要体会和自定义类实现枚举不同的地方。Enumeration03.java

代码

|  |
| --- |
| package com.hspedu.enum\_;  /\*\*  \* @author 韩顺平  \* @version 1.0  \*/  public class Enumeration03 {  public static void main(String[] args) {  System.out.println(Season2.AUTUMN);  System.out.println(Season2.SUMMER);  }  }  //演示使用 enum 关键字来实现枚举类  enum Season2 {//类 |

|  |
| --- |
| //定义了四个对象, 固定.  // public static final Season SPRING = new Season("春天", "温暖");  // public static final Season WINTER = new Season("冬天", "寒冷");  // public static final Season AUTUMN = new Season("秋天", "凉爽");  // public static final Season SUMMER = new Season("夏天", "炎热");  //如果使用了 enum 来实现枚举类  //1. 使用关键字 enum 替代 class  //2. public static final Season SPRING = new Season("春天", "温暖") 直接使用  // SPRING("春天", "温暖") 解读 常量名(实参列表)  //3. 如果有多个常量(对象) ， 使用 ,号间隔即可  //4. 如果使用 enum 来实现枚举，要求将定义常量对象，写在前面  //5. 如果我们使用的是无参构造器，创建常量对象，则可以省略 ()  SPRING("春天", "温暖"), WINTER("冬天", "寒冷"), AUTUMN("秋天", "凉爽"),  SUMMER("夏天", "炎热")/\*, What()\*/;  private String name;  private String desc;//描述  private Season2() {//无参构造器  }  private Season2(String name, String desc) {  this.name = name;  this.desc = desc;  } |

|  |
| --- |
| public String getName() {  return name;  }  public String getDesc() {  return desc;  }  @Override  public String toString() {  return "Season{" +  "name='" + name + '\'' +  ", desc='" + desc + '\'' +  '}';  }  } |

11.7.2 enum 关键字实现枚举注意事项

1) 当我们使用 enum 关键字开发一个枚举类时，默认会继承 Enum类, 而且是一个 final 类[如何证明],老师使用javap 工

具来演示

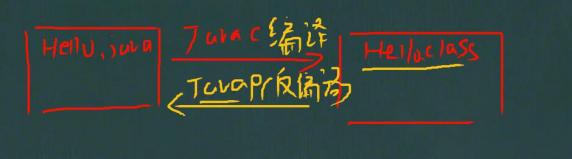
2) 传统的 public static final Season2 SPRING = new Season2("春天", "温暖"); 简化成 SPRING("春天", "温暖") ， 这里必

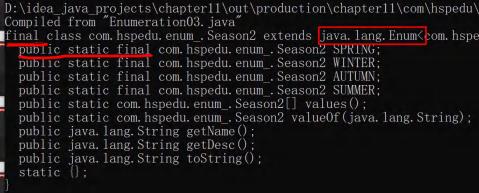
须知道，它调用的是哪个构造器.

3) 如果使用无参构造器 创建 枚举对象，则实参列表和小括号都可以省略

4) 当有多个枚举对象时，使用, 间隔，最后有一个分号结尾

5) 枚举对象必须放在枚举类的行首.





11.8 enum 关键字实现枚举-课堂练习

下面代码是否正确, 并说明表示的含义?

enum Gender{ //1min

BOY , GIRL; //这里其实就是调用 Gender 类的无参构造器

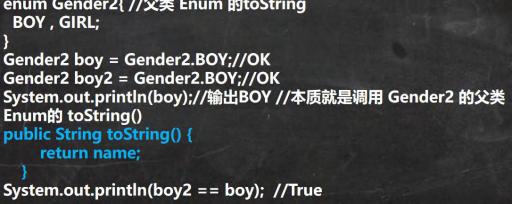
}

1) 上面语法是 ok

2) 有一个枚举类 Gender ， 没有属性。

3) 有两个枚举对象 BOY, GIRL, 使用的无参构造器创建.

下面代码输出什么? EnumExercise01.java



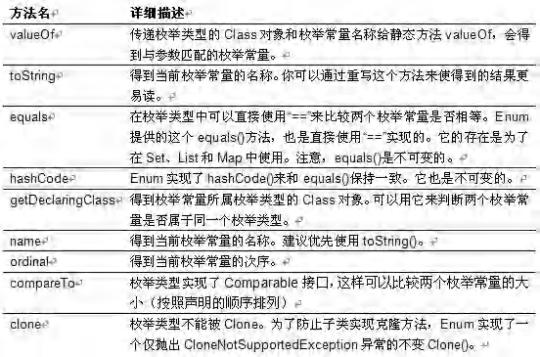
11.9 enum 常 2 用方法说明

说明：使用关键字 enum 时，会隐式继承 Enum类, 这样我们就可以使用 Enum类相关的方法。[看下源码定义.]

public abstract class Enum<E extends Enum<E>>

implements Comparable<E>, Serializable {

}



11. 10enum 常用方法应用实例

我们一起来举例说明 enum 常用的方法的使用，对 Season2 测试. EnumMethod.java

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) name：返回当前对象名（常量名），子类中不能重写

3) ordinal：返回当前对象的位置号，默认从 0 开始

4) values：返回当前枚举类中所有的常量

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！

6) compareTo：比较两个枚举常量，比较的就是编号！

7) 代码

|  |
| --- |
| package com.hspedu.enum\_;  /\*\*  \* @author 韩顺平  \* @version 1.0  \* 演示 Enum类的各种方法的使用  \*/  public class EnumMethod {  public static void main(String[] args) {  //使用 Season2 枚举类，来演示各种方法  Season2 autumn = Season2.AUTUMN;  //输出枚举对象的名字  System.out.println(autumn.name());  //ordinal() 输出的是该枚举对象的次序/编号，从 0 开始编号 |

|  |
| --- |
| //AUTUMN 枚举对象是第三个，因此输出 2  System.out.println(autumn.ordinal());  //从反编译可以看出 values 方法，返回 Season2[]  //含有定义的所有枚举对象  Season2[] values = Season2.values();  System.out.println("===遍历取出枚举对象(增强 for)====");  for (Season2 season: values) {//增强 for 循环  System.out.println(season);  }  //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  //执行流程  //1. 根据你输入的 "AUTUMN" 到 Season2 的枚举对象去查找  //2. 如果找到了，就返回，如果没有找到，就报错  Season2 autumn1 = Season2.valueOf("AUTUMN");  System.out.println("autumn1=" + autumn1);  System.out.println(autumn == autumn1);  //compareTo：比较两个枚举常量，比较的就是编号  //老韩解读  //1. 就是把 Season2.AUTUMN 枚举对象的编号 和 Season2.SUMMER 枚举对象的编号比较  //2. 看看结果  /\*  public final int compareTo(E o) { |

|  |
| --- |
| return self.ordinal - other.ordinal;  }  Season2.AUTUMN 的编号[2] - Season2.SUMMER 的编号[3]  \*/  System.out.println(Season2.AUTUMN.compareTo(Season2.SUMMER));  //补充了一个增强 for  // int[] nums = {1, 2, 9};  // //普通的 for 循环  // System.out.println("=====普通的 for=====");  // for (int i = 0; i < nums.length; i++) {  // System.out.println(nums[i]);  // }  // System.out.println("=====增强的 for=====");  // //执行流程是 依次从 nums 数组中取出数据，赋给 i, 如果取出完毕，则退出 for  // for(int i : nums) {  // System.out.println("i=" + i);  // }  }  } |

课堂完成 EnumExercise02.java

声明 Week 枚举类，其中包含星期一至星期日的定义； MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY,

SATURDAY, SUNDAY;

使用 values 返回所有的枚举数组, 并遍历 , 输出左图效果

5min 完成

|  |
| --- |
| package com.hspedu.enum\_;  /\*\*  \* @author 韩顺平  \* @version 1.0  \*/  public class EnumExercise02 {  public static void main(String[] args) {  //获取到所有的枚举对象， 即数组  Week[] weeks = Week.values();  //遍历，使用增强 for  System.out.println("===所有星期的信息如下===");  for (Week week : weeks) {  System.out.println(week);  }  }  }  /\*  声明 Week 枚举类，其中包含星期一至星期日的定义；  MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY;  使用 values 返回所有的枚举数组, 并遍历 , 输出左图效果  \*/ |

|  |
| --- |
| enum Week {  //定义 Week 的枚举对象  MONDAY("星期一"), TUESDAY("星期二"), WEDNESDAY("星期三"), THURSDAY("星期四"),  FRIDAY("星期五"), SATURDAY("星期六"), SUNDAY("星期日");  private String name;  private Week(String name) {//构造器  this.name = name;  }  @Override  public String toString() {  return name;  }  } |

11. 11enum 实现接口

EnumDetail.java

1) 使用 enum 关键字后，就不能再继承其它类了，因为 enum 会隐式继承 Enum ，而 Java 是单继承机制。

2) 枚举类和普通类一样，可以实现接口，如下形式。

enum 类名 implements 接口 1 ，接口 2{}

3) 代码

|  |
| --- |
| package com.hspedu.enum\_;  /\*\* |

|  |
| --- |
| \* @author 韩顺平  \* @version 1.0  \*/  public class EnumDetail {  public static void main(String[] args) {  Music.CLASSICMUSIC.playing();  }  }  class A {  }  //1.使用 enum 关键字后，就不能再继承其它类了，因为 enum 会隐式继承 Enum ，而 Java 是单继承机制  //enum Season3 extends A {  //  //}  //2.enum 实现的枚举类，仍然是一个类，所以还是可以实现接口的.  interface IPlaying {  public void playing();  }  enum Music implements IPlaying {  CLASSICMUSIC;  @Override  public void playing() {  System.out.println("播放好听的音乐...");  } |

|  |
| --- |
| } |

11. 12注解的理解

1) 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。

2) 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。

3) 在 JavaSE 中，注解的使用目的比较简单，例如标记过时的功能，忽略警告等。在 JavaEE 中注解占据了更重要的角

色，例如用来配置应用程序的任何切面，代替java EE 旧版中所遗留的繁冗代码和 XML 配置等。

11. 13基本的 Annotation 介绍

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。用于修饰它支持的程序元

素

三个基本的 Annotation:

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

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

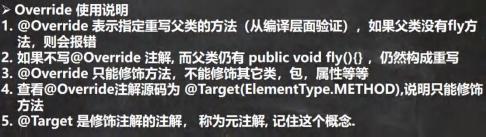
11. 14基本的 Annotation 应用案例

11.14. 1 @Override 注解的案例 Override\_.java



|  |
| --- |
| package com.hspedu.annotation\_;  /\*\*  \* @author 韩顺平  \* @version 1.0  \*/  public class Override\_ {  public static void main(String[] args) {  }  }  class Father{//父类  public void fly(){  System.out.println("Father fly...");  }  public void say(){}  } |

|  |
| --- |
| class Son extends Father {//子类  //老韩解读  //1. @Override 注解放在 fly 方法上，表示子类的 fly 方法时重写了父类的 fly  //2. 这里如果没有写 @Override 还是重写了父类 fly  //3. 如果你写了@Override注解，编译器就会去检查该方法是否真的重写了父类的  // 方法，如果的确重写了，则编译通过，如果没有构成重写，则编译错误  //4. 看看 @Override 的定义  // 解读： 如果发现 @interface 表示一个 注解类  /\*  @Target(ElementType.METHOD)  @Retention(RetentionPolicy.SOURCE)  public @interface Override {  }  \*/  @Override //说明  public void fly() {  System.out.println("Son fly....");  }  @Override  public void say() {}  } |



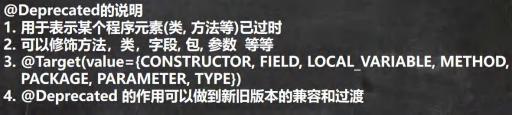
11.14.2 @Deprecated 注解的案例 Deprecated\_.java

@Deprecated: 用于表示某个程序元素(类, 方法等)已过时

代码

|  |
| --- |
| package com.hspedu.annotation\_;  /\*\*  \* @author 韩顺平  \* @version 1.0  \*/  public class Deprecated\_ {  public static void main(String[] args) {  A a = new A();  a.hi();  System.out.println(a.n1);  }  }  //老韩解读  //1. @Deprecated 修饰某个元素, 表示该元素已经过时  //2. 即不在推荐使用，但是仍然可以使用  //3. 查看 @Deprecated 注解类的源码 |

|  |
| --- |
| //4. 可以修饰方法，类，字段, 包, 参数 等等  //5. @Deprecated 可以做版本升级过渡使用  /\*  @Documented  @Retention(RetentionPolicy.RUNTIME)  @Target(value={CONSTRUCTOR, FIELD, LOCAL\_VARIABLE, METHOD, PACKAGE, PARAMETER, TYPE})  public @interface Deprecated {  }  \*/  @Deprecated  class A {  @Deprecated  public int n1 = 10;  @Deprecated  public void hi(){  }  } |



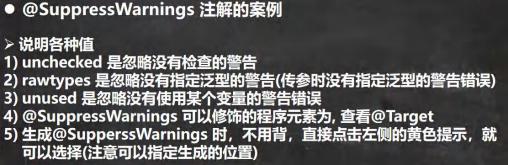
11.14.3 @SuppressWarnings 注解的案例 SuppressWarnings\_.java

@SuppressWarnings: 抑制编译器警告

|  |
| --- |
| package com.hspedu.annotation\_;  import java.util.ArrayList;  import java.util.List;  /\*\*  \* @author 韩顺平  \* @version 1.0  \*/  @SuppressWarnings({"rawtypes", "unchecked", "unused"})  public class SuppressWarnings\_ {  //老韩解读  //1. 当我们不希望看到这些警告的时候，可以使用 SuppressWarnings 注解来抑制警告信息  //2. 在{""} 中，可以写入你希望抑制(不显示)警告信息  //3. 可以指定的警告类型有  // all ，抑制所有警告  // boxing ，抑制与封装/拆装作业相关的警告  // //cast ，抑制与强制转型作业相关的警告  // //dep-ann ，抑制与淘汰注释相关的警告  // //deprecation ，抑制与淘汰的相关警告  // //fallthrough ，抑制与 switch 陈述式中遗漏 break 相关的警告  // //finally ，抑制与未传回 finally 区块相关的警告  // //hiding ，抑制与隐藏变数的区域变数相关的警告  // //incomplete-switch ，抑制与 switch 陈述式(enum case)中遗漏项目相关的警告  // //javadoc ，抑制与javadoc 相关的警告 |

|  |
| --- |
| // //nls ，抑制与非 nls 字串文字相关的警告  // //null ，抑制与空值分析相关的警告  // //rawtypes ，抑制与使用 raw 类型相关的警告  // //resource ，抑制与使用 Closeable 类型的资源相关的警告  // //restriction ，抑制与使用不建议或禁止参照相关的警告  // //serial ，抑制与可序列化的类别遗漏 serialVersionUID 栏位相关的警告  // //static-access ，抑制与静态存取不正确相关的警告  // //static-method ，抑制与可能宣告为 static 的方法相关的警告  // //super ，抑制与置换方法相关但不含 super 呼叫的警告  // //synthetic-access ，抑制与内部类别的存取未最佳化相关的警告  // //sync-override ，抑制因为置换同步方法而遗漏同步化的警告  // //unchecked ，抑制与未检查的作业相关的警告  // //unqualified-field-access ，抑制与栏位存取不合格相关的警告  // //unused ，抑制与未用的程式码及停用的程式码相关的警告  //4. 关于 SuppressWarnings 作用范围是和你放置的位置相关  // 比如 @SuppressWarnings 放置在 main 方法，那么抑制警告的范围就是 main  // 通常我们可以放置具体的语句, 方法, 类.  //5. 看看 @SuppressWarnings 源码  //(1) 放置的位置就是 TYPE, FIELD, METHOD, PARAMETER, CONSTRUCTOR, LOCAL\_VARIABLE  //(2) 该注解类有数组 String[] values() 设置一个数组比如 {"rawtypes", "unchecked", "unused"}  /\*  @Target({TYPE, FIELD, METHOD, PARAMETER, CONSTRUCTOR, LOCAL\_VARIABLE})  @Retention(RetentionPolicy.SOURCE)  public @interface SuppressWarnings {  String[] value(); |

|  |
| --- |
| }  \*/  public static void main(String[] args) {  List list = new ArrayList();  list.add("jack");  list.add("tom");  list.add("mary");  int i;  System.out.println(list.get(1));  }  public void f1() {  // @SuppressWarnings({"rawtypes"})  List list = new ArrayList();  list.add("jack");  list.add("tom");  list.add("mary");  // @SuppressWarnings({"unused"})  int i;  System.out.println(list.get(1));  }  } |



11. 15JDK 的元 Annotation(元注解， 了解)

11.15. 1 元注解的基本介绍

JDK 的元 Annotation 用于修饰其他 Annotation

元注解： 本身作用不大，讲这个原因希望同学们，看源码时，可以知道他是干什么.

11.15.2 元注解的种类 (使用不多，了解, 不用深入研究)

1) Retention //指定注解的作用范围，三种 SOURCE,CLASS,RUNTIME

2) Target // 指定注解可以在哪些地方使用

3) Documented //指定该注解是否会在javadoc 体现

4) Inherited //子类会继承父类注解

11.15.3 @Retention 注解

说明

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, @Rentention 包含一个 RetentionPolicy

类型的成员变量, 使用 @Rentention 时必须为该 value 成员变量指定值:

@Retention 的三种值

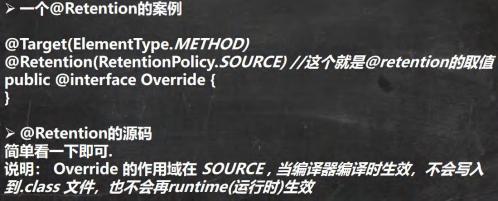
1) RetentionPolicy.SOURCE: 编译器使用后，直接丢弃这种策略的注释

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

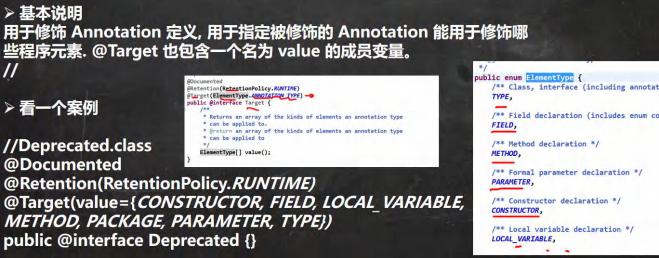
值

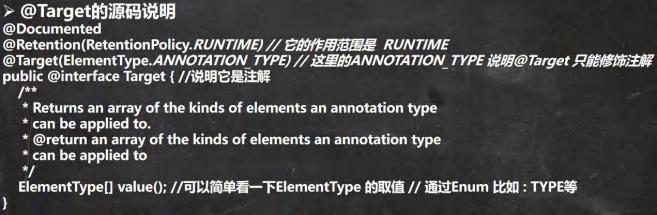
3) RetentionPolicy.RUNTIME:编译器将把注解记录在 class 文件中. 当运行 Java 程序时, JVM 会保留注解. 程序可以

通过反射获取该注解



11.15.4 @Target



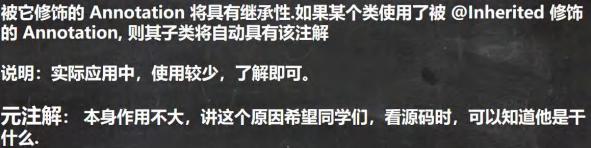


11.15.5 @Documented





11.15.6 @Inherited 注解



11. 16第 10 章作业

