1、概念

把实例有限且固定的类称之为枚举类。枚举类用关键字**enum**定义，枚举类是一种的特殊的类，但一样可以有自己的属性和方法，一样可以实现一个或多个接口，也可以定义自己的构造器。

枚举类与普通类的区别：

* 可以实现一个或多个接口，使用enum定义的枚举类默认继承了java.lang.Enum类，而不是Object类，Enum类实现了java.lang.serializable和java.lang.Comparable类。
* 枚举类的构造器只能用private访问控制符，若省略，默认为private,显式指定，只能用private。
* 枚举类的实例必须在枚举类中显式列出，否则这个枚举类永远不能产生实例。列出的实例，系统会自动添加public static final 修饰。
* 所有枚举类都提供了一个values方法，该方法可方便遍历枚举值。

2、Enum 类源代码片段

**public abstract class** Enum<**E extends** Enum<**E**>> **implements** Comparable<**E**>**, Serializable**{

// name属性，枚举类元素的名称

**private** **final** String name;

**public final** String name() {**return** name;}

// ordinal属性，枚举类元素的在当前枚举类中出现的位置

**private final int** ordinal;

**public final int** ordinal() {**return** ordinal;}

// ordinal属性，枚举类元素的在当前枚举类中出现的位置

**protected** Enum(String name, **int** ordinal) {  
 **this**.name = name;  
 **this**.ordinal = ordinal;  
}

**public** String toString() {**return** name;}

// 通过valueOf可以获取指定枚举类实例

**public static** <**T extends** Enum<**T**>> **T** valueOf(Class<**T**> enumType,  
 String name) { //… }

// 省略以下方法

equals(),hashCode(),clone(),compareTo(),finalize(),

readObject(),readObjectNoData()

}

由于所有枚举类都继承了Enum类，所以所有枚举类都可以使用name()获取枚举名，可以使用ordinal()获取当前枚举对象实例在枚举类中的位置，可以通过valueOf()方法获取枚举对象实例，通过values()方法遍历枚举类中所有枚举对象实例。

3、枚举类的属性、方法、构造器

枚举类通常设计为不可变类，也就说明它的属性值不应该允许改变，这样会更安全。所以枚举类的属性都应该用private final修饰。

**public** **enum** EnumGendar {

// 枚举类的实例

*MALE*("男"),*FAMALE*("女");

**private** **final** String name;

// 带参的构造函数

**private** EnumGendarTest(String name){

**this**.name = name;

}

// getter方法

**public** String getName(){

**return** **this**.name;

}

**public** **static** **void** main(String[] args){

//枚举类的调用

EnumGendar egt = Enum.*valueOf*(EnumGendar.**class**,"MALE");

String name1 = egt.getName();

String name2 = egt.name();

String ordinal = egt.ordinal();

System.*out*.println(name1+","+name2+","+ordinal);

}

}

一旦属性使用final修饰，属性的初始化就只能由带参的构造器来完成。

4、实现接口的枚举类

如果由枚举类实现接口里的方法，则每个枚举值在调用该方法时，都有相同的行为方法，如果需要每个枚举值在调用该方法时呈现不同的行为方式，则可以让每个枚举值来实现该方法。例：

**interface** Season{

**public** **void** info();

}

//枚举类实现接口

**public** **enum** EnumSeasonImpl **implements** Season {

//匿名内部类为每个实例实现方法，定义不同的行为

*SPRING*{

**public** **void** info(){

System.*out*.println("春暖花开，蝴蝶自来");

}

},

*SUMMER*{

**public** **void** info(){

System.*out*.println("炎炎夏日，烈日 灼灼");

}

},

*FALL*{

**public** **void** info(){

System.*out*.println("一叶知秋，金风送爽");

}

},

*WINTER*{

**public** **void** info(){

System.*out*.println("寒冬腊月，傲雪凌霜");

}

};

**public** **static** **void** main(String[] args) {

//枚举类单个实例调用

EnumSeasonImpl esi = Enum.*valueOf*(EnumSeasonImpl.**class**, "SPRING");

esi.info();

//枚举类所有实例遍历

**for**(EnumSeasonImpl es:EnumSeasonImpl.*values*()){

es.info();

}

}

}

5、包含抽象方法的枚举类

枚举类里定义抽象方法时无需显式使用abstract关键字将枚举类定义成抽象类，但因枚举类需显式创建枚举值，而不是作为父类，所以定义每个枚举值时必须为抽象方法提供实现，否则将出现编译错误。例：

**public** **enum** EnumOperation {

*PLUS*{

**public** **double** eval(**double** x,**double** y){

**return** x+y;

}

},*MINUS*{

**public** **double** eval(**double** x,**double** y){

**return** x-y;

}

},*TIMES*{

**public** **double** eval(**double** x,**double** y){

**return** x\*y;

}

},*DIVIDE*{

**public** **double** eval(**double** x,**double** y){

**return** x/y;

}

};

//枚举类中抽象方法

**public** **abstract** **double** eval(**double** x,**double** y);

//测试方法

**public** **static** **void** main(String[] args) {

**for**(EnumOperation eo:EnumOperation.*values*()){

**double** result = eo.eval(9, 3);

System.*out*.println(result);

}

}

}