第二章
类与对象的基本概念

2.3.2 内存回收技术

- 同C和C++的区别
  - C语言中通过free来释放内存
  - C++中则通过delete来释放内存
  - 在C和C++中，如果程序员忘记释放内存，则容易造成内存泄漏甚至导致内存耗尽
  - 在Java中不会发生内存泄漏情况，但对于其它资源，则有产生泄漏的可能性

2.3.2 内存回收技术

- 内存回收技术
  - 当一个对象在程序中不再被使用时，就成为一个无用对象
    - 当前的代码段不属于对象的作用域
    - 把对象的引用赋值为空
  - **Java运行时系统**通过**垃圾收集器(gc:garbage collector)**周期性地释放无用对象所使用的内存
  - **Java运行时系统**会在对对象进行自动垃圾回收前，自动调用对象的**finalize()**方法

protected void	finalize() Called by the garbage collector on an object when garbage collection determines that there are no more references to the object.

2.3.2 内存回收技术(续)——垃圾收集器

- 垃圾收集器
  - 自动扫描对象的动态内存区，对不再使用的对象做上标记以进行垃圾回收
  - 作为JVM的一个线程运行
    - 通常在系统空闲时异步地执行
    - 当系统的内存用尽或程序中调用System.gc()要求进行垃圾收集时，与系统同步运行

static void	gc () Runs the garbage collector.

2.3.2 内存回收技术(续)——finalize()方法

- finalize()方法
  - 在类java.lang.Object中声明，因此Java中的每一个类都有该方法
  - 声明格式

    protected void finalize() throws throwable
  - 如果一个类需要释放除内存以外的资源，则需在类中重写finalize()方法

2.3.2 内存回收技术(续)——垃圾收集器

- 在Java程序中，当一个对象变得不可到达的时候，垃圾回收器会回收与该对象关联的存储空间，并不需要程序员做专门的工作。
- finalize()并不能保证会被及时地执行，从一个对象变得不可达到开始，到它的终结函数被执行，这段时间是任意的、不确定的。
- 无论是“垃圾回收”还是“终结”，都不保证一定会发生。如果JVM并未面临内存耗尽的情形，它是不会浪费时间去执行垃圾回收以恢复内存的。
- 避免使用finalize()函数。

2.4 枚举类型

- Java 5的新特色，可以取代Java 5之前的版本中使用的常量
- 需要一个有限集合，而且集合中的数据为特定的值时，可以使用枚举类型
- 格式：

[**public**] enum 枚举类性名称 [**implements** 接口名称列表]

{

    枚举值；

    变量成员声明及初始化；

    方法声明及方法体；

}

7

2.4 枚举类型

- 例2-17

```
enum Score {
        EXCELLENT,
        QUALIFIED,
        FAILED;
};
public class ScoreTester {
        public static void main(String[] args) {
                giveScore(Score.EXCELLENT);
        }
```


8

2.4 枚举类型

```
public static void giveScore(Score s){
        switch(s){
            case EXCELLENT:
                System.out.println("Excellent");
                break;
            case QUALIFIED:
                System.out.println("Qualified");
                break;
            case FAILED:
                System.out.println("Failed");
                break;
        }
    }
}
```

2.4 枚举类型

```
public class ConstantesTest {
    enum Constants { // 将常量放置在枚举类型中
        Constants_A, Constants_B
    }

    // 定义一个方法，这里的参数为枚举类型对象
        public static void doit(Constants c) {
            switch (c) { // 根据枚举类型对象做不同操作
                case Constants_A:
                    System.out.println("doit() Constants_A");
                    break;
                case Constants_B:
                    System.out.println("doit() Constants_B");
                    break;
            }
        }

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        ConstantesTest.doit(Constants.Constants_A); // 使用枚举类型中的常量
        ConstantesTest.doit(Constants.Constants_B); // 使用枚举类型中的常量
    }
}
```

运行结果：
doit() Constants_A
doit() Constants_B

2.4 枚举类型

- 枚举类型可看作是一个类，继承于java.lang.Enum

方法名称	具体含义	使用方法
values()	该方法可以将枚举类型成员以数组的形式返回	枚举类型名称. values()
valueOf()	该方法可以实现将普通字符串转换为枚举实例	枚举类型名称. valueOf()
compareTo()	该方法用于比较两个枚举对象在定义时的顺序	枚举对象. compareTo()
ordinal()	该方法用于得到枚举成员的位置索引	枚举对象. ordinal()

枚举类型的常用方法

2.4 枚举类型

- values()示例

```
public class ShowEnum {
    enum Constants2 { // 将常量放置在枚举类型中
        Constants_A, Constants_B
    }
     // 循环由values()方法返回的数组
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        for (int i = 0; i < Constants2.values().length; i++) {
            // 将枚举成员变量打印
            System.out.println("枚举类型成员变量: " + Constants2.values()[i]);
    }

    }
}
```

运行结果：
枚举类型成员变量：Constants_A
枚举类型成员变量：Constants_B

2.4 枚举类型

- valueOf()与compareTo()示例

```
public class EnumMethodTest {
    enum Constants2 { // 将常量放置在枚举类型中
        Constants_A, Constants_B
    }
    // 定义比较枚举类型方法，参数类型为枚举类型
        public static void compare(Constants2 c) {
            // 根据values()方法返回的数组做循环操作
            for (int i = 0; i < Constants2.values().length; i++) {
                // 将比较结果返回
                System.out.println(c + "与" + Constants2.values()[i] + "的比较结果为："
                        + c.compareTo(Constants2.values()[i]));
            }
        }
        // 在主方法中调用compare()方法
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        compare(Constants2.valueOf("Constants_B"));
    }
}
```

运行结果：
Constants_B与Constants_A的比较结果为：1
Constants_B与Constants_B的比较结果为：0

2.4 枚举类型

- ordinal() 示例

```
public class EnumIndexTest {
    enum Constants2 { // 将常量放置在枚举类型中
        Constants_A, Constants_B, Constants_C
    }

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        for (int i = 0; i < Constants2.values().length; i++) {
            // 在循环中获取枚举类型成员的索引位置
            System.out.println(Constants2.values()[i] + "在枚举类型中位置索引值"
                    + Constants2.values()[i].ordinal());
        }
    }
}
```

运行结果：
Constants_A在枚举类型中位置索引值0
Constants_B在枚举类型中位置索引值1
Constants_C在枚举类型中位置索引值2

2.4 枚举类型

- 枚举类型中，可以添加构造方法，但必须为private修饰，语法如下：

```
enum 枚举类型名称{
    Constants_A("我是枚举成员_A"),
    Constants_B("我是枚举成员_B"),
    Constants_C("我是枚举成员_C"),
    Constants_D(3);
    private String description;
    private Constants2(){
      //定义默认构造方法
    }
    private Constants2(String description){
      //定义带参数的构造方法，参数类型为字符串型
        this.description=description;
    }
    private Constants2(int i){
      //定义带参数的构造方法，参数类型为整型
        this.i=this.i+i;
    }
```

2.4 枚举类型

- 使用枚举类型的优势：
  - 类型安全
  - 紧凑有效的数据定义
  - 可以和程序其他部分完美交互
  - 运行效果高

没有指针的Java语言

- 引用（reference）实质就是指针（pointer）
  - 但是它是受控的、安全的
  - 比如：
  - 会检查空指引
  - 没有指针运算*（p+5）
  - 不能访问没有引用到的内存
  - 自动回收垃圾

没有指针的Java语言

- C语言指针在Java中的体现：
  - （1）传地址->对象
    - 如引用类型：引用本身就相当于指针，可以用来修改对象的属性、调用对象的方法
  - （2）指针运算->数组
    - 如：*（p+5）可以用args[5]
  - （3）函数指针->接口、Lambda表达式
  - （4）指向结点的指针->对象的引用
  - （5）使用JNI（Java Native Interface）

没有指针的Java语言

```
public class EqualsTester {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        String str1=new String("hello");
        String str2=new String("hello");

        System.out.println(str1==str2);
        System.out.println(str1.equals(str2));

    }

}
```

运行结果：
false
true

思考："=="相等还是不等？需根据具体情况分析。

没有指针的Java语言

- 基本类型
  - 数值类型：转换后进行比较
  - 浮点数，最好不直接用“==”
  - Double.NAN==Double.NAN结果为false
    - 参见JDK的API文档
  - boolean型无法与int型比较

没有指针的Java语言

```
Integer i = new Integer(10);
Integer j = new Integer(10);
System.out.println(i==j);
```

false，因为对象是两个

```
Integer m = 10;
Integer n = 10;
System.out.println(m==n);
```

true，因为对象有缓存

```
Integer p = 200;
Integer q = 200;
System.out.println(p==q);
```

false，因为对象是两个

装箱对象是否相等：注意缓存

If the value p being boxed is true, false,a byte, or a char in the range \u0000 to \u007f, or an int or short number between -128 and 127 (inclusive), then let r1 and r2 be the results of any two boxing conversions of p. It is always the case that r1 ==r2.

没有指针的Java语言

- 枚举类型
  - 内部进行了唯一实例化，所以可以直接判断
- 引用对象
  - 直接看两个引用是否一样
  - 如果要判断内容是否一样，则要重新equals方法
  - 如果重写equals方法，最好重写hashCode()方法

没有指针的Java语言

- String对象的特殊性
  - 判断相等，不要用“==”，用“equals”
  - 但是字符串常量（String literal）及字符串常量会进行内部化（interned），相同的字符串常量是==的。

```
String hello="Hello",lo="lo";
System.out.println(hello=="Hello");
System.out.println(hello==("Hel"+"lo"));
System.out.println(hello==("Hel"+lo));
System.out.println(hello==new String("Hello"));
System.out.println(hello==("Hel"+lo).intern());
```

运行结果：
true
true
false
false
true

本章小结

- 本章内容
  - 面向对象程序设计的基本概念和思想
  - Java语言类与对象的基本概念和语法，包括类的声明、类成员的访问，以及对象的构造、初始化和回收
  - 枚举类型
- 本章要求
  - 理解类和对象的概念
  - 熟练使用类及其成员的访问控制方法
  - 熟练掌握各种构造方法
  - 了解java的垃圾回收机制

本章小结

- ✓ 1.面向过程和面向对象的区别和联系
- ✓ 2.类和对象的关系
- ✓ 3.局部变量、成员变量、静态变量分别怎么声明?（驼峰命名原则）
- ✓ 4.构造方法的作用和特征
- ✓ 5.this关键字的作用和用法
- ✓ 6.static关键字的作用
- ✓ 7.栈的特点是?存放什么内容?堆得特点是?存放什么内容?
- ✓ 8.private、默认、protected、public四个修饰符的作用
- ✓ 9.一般属性是否要设置为private?如果属性设置为private，如何让外部访问该属性?
- ✓ 10.面向对象中的封装，追求的是“高内聚，低耦合”，如何理解?