Java语言程序设计（第2版）

第 2 章 类与对象（一）

清华大学　　郑　莉

导学

目录

面向对象方法的特性

抽象、封装、继承、多态

类与对象基础

类的声明
对象的创建
数据成员
方法成员
包
类的访问权限控制
类成员的访问权限控制

对象初始化和回收

构造方法

内存回收

枚举类型

简单枚举类型

枚举类

应用举例

面向对象方法的特性

<2.1>

抽象、封装、继承、多态

抽象

- 忽略问题中与当前目标无关的方面
- 只关注与当前目标有关的方面

例：钟表

- 数据(属性)
  - int Hour; int Minute; int Second;
- 方法(行为)
  - SetTime(); ShowTime();

封装

- 封装是一种信息隐蔽技术

封装

- 利用抽象数据类型将数据和基于数据的操作封装在一起；
- 用户只能看到对象的封装界面信息，对象的内部细节对用户是隐蔽的；
- 封装的目的在于将对象的使用者和设计者分开，使用者不必知道行为实现的细节。

继承

- 基于已有类产生新类的机制

继承

- 是指新的类可以获得已有类（称为超类、基类或父类）的属性和行为，称新类为已有类的子类（也称为派生类）；
- 在继承过程中子类继承了超类的特性，包括方法和实例变量；
- 子类也可修改继承的方法或增加新的方法；
- 有助于解决软件的可重用性问题，使程序结构清晰，降低了编码和维护的工作量。

- 单继承
  - 一个子类只有单一的直接超类。
- 多继承
  - 一个子类可以有一个以上的直接超类。
- Java语言仅支持单继承。

多态

- 超类及其不同子类的对象可以响应同名的消息，具体的实现方法却不同；
- 主要通过子类对父类方法的覆盖来实现。

类声明与对象创建

<2.2>类与对象基础

<2.2.1>

类与对象的关系

- 类是对一类对象的描述；
- 对象是类的具体实例。

类声明

类声明

```
[public] [abstract | final] class 类名称
[extends 父类名称]
[implements 接口名称列表]
{
    数据成员声明及初始化;
    方法声明及方法体;
}
```

- class
  - 表明其后声明的是一个类。
- extends
  - 如果所声明的类是从某一父类派生而来，那么，父类的名字应写在extends之后
- implements
  - 如果所声明的类要实现某些接口，那么，接口的名字应写在implements之后
- public
  - 表明此类为公有类
- abstract
  - 指明此类为抽象类
- final
  - 指明此类为终结类

例：钟表类

```
public class  Clock {
  //变量成员
    int hour ;
    int minute ;
    int second ;
   // 方法成员
   public void setTime(int newH, int newM, int newS){
      hour=newH ;
      minute=newM ;
      second=news ;
   }
   public void showTime() {
       System.out.println(hour+":"+minute+":"+second);
   }
}
```

对象声明与创建

- 创建类的实例（对象），通过对象使用类的功能。

对象引用声明

- 语法
  类名 引用变量名;
- 例
  - Clock是已经声明的类名，声明引用变量aclock，用于存储该类对象的引用：
    Clock aclock;
- 声明一个引用变量时并没有生成对象。

对象的创建

- 语法形式：
  new <类名>()
- 例如：
  aclock=new Clock()
  new的作用是：
  - 在内存中为Clock类型的对象分配内存空间；
  - 返回对象的引用。
- 引用变量可以被赋以空值
  例如：aclock=null;

数据成员

<2.2.2>

数据成员

- 表示对象的状态
- 可以是任意的数据类型(简单类型，类，接口，数组等——括号内文字不显示)

数据成员声明

- 语法形式

  [public | protected | private]
  [static][final][transient] [volatile]
  数据类型 变量名1[=变量初值], 变量名2[=变量初值], … ;
- 说明
  - public、protected、private 为访问控制符。
  - static指明这是一个静态成员变量（类变量）。
  - final指明变量的值不能被修改。
  - transient指明变量是不需要序列化的。
  - volatile指明变量是一个共享变量。

实例变量

- 没有static修饰的变量（数据成员）称为实例变量；
- 存储所有实例都需要的属性，不同实例的属性值可能不同；
- 可通过下面的表达式访问实例属性的值

  <实例名>.<实例变量名>

例：圆类

- 圆类保存在文件Circle.java 中，测试类保存在文件ShapeTester.java中，两文件放在相同的目录下

```
public class Circle {
        int radius;
}

public class ShapeTester {
    public static void main(String args[]) {
        Circle x;
        x = new Circle();
        System.out.println(x);
        System.out.println("radius = " + x.radius);
    }
}
```

运行结果

- 编译后运行结果如下：

  Circle@26b249

  radius =0

- 说明

  默认的toString()返回：

  getClass().getName() +”@” + Integer.toHexString(hashCode())

例：矩形类

- 矩形类保存在Rectangle.java中，测试类保存在ShapeTester.java中，两文件保存在相同目录下

```
public class Rectangle {
        double width = 10.128;
        double height = 5.734;
}

public class ShapeTester {
        public static void main(String args[]) {
                Circle    x;
                Rectangle y;
                x = new Circle();
                y = new Rectangle();
                System.out.println(x + "    " + y);
            }
}
```

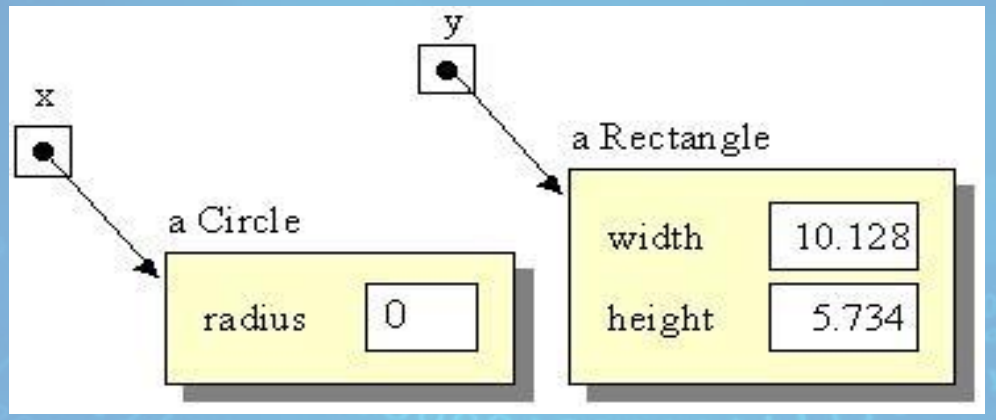
运行结果

- 编译后运行结果如下：

  Circle@82f0db　Rectangle@92d342

- 说明

  Circle及Rectangle类对象的状态如图



类变量

- 为该类的所有对象共享（——不显示）

类变量（静态变量）

- 用static修饰。
- 在整个类中只有一个值。
- 类初始化的同时就被赋值。
- 适用情况
  - 类中所有对象都相同的属性。
  - 经常需要共享的数据。
  - 系统中用到的一些常量值。
- 引用格式

  <类名 | 实例名>.<类变量名>

例：具有类变量的圆类

```
public class Circle {
    static double PI = 3.14159265;
    int radius;
}
```

当我们生成Circle类的实例时，在每一个实例中并没有存储PI的值，PI的值存储在类中。

对类变量进行测试

```
public class ClassVariableTester {
   public static void main(String args[]) {
      Circle x = new Circle();
      System.out.println(x.PI);
      System.out.println(Circle.PI);
      Circle.PI = 3.14;
      System.out.println(x.PI);
      System.out.println(Circle.PI);
   }
}
```

运行结果：

```
3.14159265
3.14159265
3.14
3.14
```

方法成员

<2.2.3>

分为实例方法和类方法

语法形式

[public | protected | private]

[static][final][abstract] [native] [synchronized]

返回类型 方法名([参数列表]) [throws exceptionList]

{

    方法体

}

- public、protected、private 控制访问权限。
- static指明这是一个类方法（静态方法）。
- final指明这是一个终结方法。
- abstract指明这是一个抽象方法。
- native用来集成java代码和其它语言的代码（本课程不涉及）。
- synchronized用来控制多个并发线程对共享数据的访问。

- 返回类型
  - 方法返回值的类型，可以是任意的Java数据类型；
  - 当不需要返回值时，返回类型为void。
- 参数类型
  - 简单数据类型、引用类型(数组、类或接口)；
  - 可以有多个参数，也可以没有参数，方法声明时的参数称为形式参数。
- 方法体
  - 方法的实现；
  - 包括局部变量的声明以及所有合法的Java语句；
  - 局部变量的作用域只在该方法内部。
- throws exceptionList
  - 抛出异常列表。

实例方法

- 表示特定对象的行为；
- 声明时前面不加static修饰符；

实例方法调用

- 给对象发消息，使用对象的某个行为/功能：调用对象的某个方法。

- 实例方法调用格式
  <对象名>.<方法名>（［参数列表］）
  <对象名>为消息的接收者。
- 参数传递
  - 值传递：参数类型为基本数据类型时
  - 引用传递：参数类型为对象类型或数组时

例：具有实例方法的圆类

```
public class Circle {
    static double PI = 3.14159265;
    int radius;
    public double circumference() {
        return 2 * PI * radius;
    }
   public void enlarge(int factor) {
        radius = radius * factor;
    }
    public boolean fitsInside (Rectangle r) {
         return (2 * radius < r.width) && (2 * radius < r.height);
    }
 }
```

圆类的测试类

```
public class InsideTester {
 public static void main(String args[]) {
        Circle c1 = new Circle();
        c1.radius = 8;
        Circle c2 = new Circle();
        c2.radius = 15;
        Rectangle r = new Rectangle();
        r.width = 20;
        r.height = 30;
        System.out.println("Circle 1 fits inside Rectangle:" + c1.fitsInside(r));
        System.out.println("Circle 2 fits inside Rectangle:" + c2.fitsInside(r));
 }
}
```

运行结果

Circle 1 fits inside Rectangle: true
Circle 2 fits inside Rectangle: false

类方法

- 表示类中对象的共有行为

类方法

- 也称为静态方法，声明时前面需加static修饰。
- 不能被声明为抽象的。
- 可以类名直接调用，也可用类实例调用。

例：温度转换

- 将摄氏温度(centigrade)转换成华氏温度(fahrenheit)
  - 转换公式为　fahrenheit = centigrade * 9 / 5 + 32
  - 除了摄氏温度值及公式中需要的常量值，此功能不依赖于具体的类实例的属性值，因此可声明为类方法
  - 转换方法centigradeToFahrenheit放在类Converter中

```
public class Converter {
  public static int centigradeToFahrenheit(int cent)
  { return (cent * 9 / 5 + 32);
  }
 }
```

  - 方法调用

```
Converter.centigradeToFahrenheit(40)
```

可变长参数

可变长参数

- 可变长参数使用省略号表示，其实质是数组。
- 例如，“String ... s”表示“String[] s”。
- 对于具有可变长参数的方法，传递给可变长参数的实际参数可以是零个到多个对象。

例：可变长参数

```
static double maxArea(Circle c, Rectangle... varRec) {
        Rectangle[] rec = varRec;
        for (Rectangle r : rec) {
          //...
        }
}

 public static void main(String[] args) {
        Circle c = new Circle();
        Rectangle r1 = new Rectangle();
        Rectangle r2 = new Rectangle();
       System.out.println("max area of c, r1 and r2 is  " + maxArea(c, r1, r2));
       System.out.println("max area of c and r1 is  " + maxArea(c, r1));
       System.out.println("max area of c and r2 is  " + maxArea(c, r2));
       System.out.println("max area of only c is  " + maxArea(c));
    }
```

包

<2.2.4>

- 包是一组类的集合
- 一个包可以包含若干个类文件，还可包含若干个包

包的作用

- 将相关的源代码文件组织在一起。
- 类名的空间管理，利用包来划分名字空间可以避免类名冲突。
- 提供包一级的封装及存取权限。

包的命名

- 每个包的名称必须是“独一无二”的。
- Java中包名使用小写字母表示。
- 命名方式建议
  - 将机构的Internet域名反序，作为包名的前导；
  - 若包名中有任何不可用于标识符的字符，用下划线替代；
  - 若包名中的任何部分与关键字冲突，后缀下划线；
  - 若包名中的任何部分以数字或其他不能用作标识符起始的字符开头，前缀下划线。

编译单元

- 一个Java源代码文件称为一个编译单元

编译单元

- 一个Java源代码文件称为一个编译单元，由三部分组成：
  - 所属包的声明（省略，则属于默认包）；
  - Import（引入）包的声明，用于导入外部的类；
  - 类和接口的声明。
- 一个编译单元中只能有一个public类，该类名与文件名相同，编译单元中的其他类往往是public类的辅助类，经过编译，每个类都会产一个class文件。

包的声明

- **命名的包（Named Packages）**
  - 例如： package Mypackage;
- **默认包（未命名的包）**
  - 不含有包声明的编译单元是默认包的一部分。

包与目录

- 包名就是文件夹名，即目录名；
- 目录名并不一定是包名；

引入包

- 为了使用其它包中所提供的类，需要使用import语句引入所需要的类。
- Java编译器为所有程序自动引入包java.lang。
- import语句的格式：
  import package1[.package2…]. (classname |*);
  - package1[.package2…]表明包的层次，对应于文件目录；
  - classname指明所要引入的类名；
  - 如果要引入一个包中的所有类，可以使用星号（*）来代替类名。

- 如果在程序中需要多次使用静态成员，则每次使用都加上类名太繁琐。用静态引入可以解决这一问题。

静态引入

- 单一引入是指引入某一个指定的静态成员，例如：import static java.lang.Math.PI;
- 全体引入是指引入类中所有的静态成员，例如：import static java.lang.Math.*;
- 例如：

```
import static java.lang.Math.PI;
public class Circle {
    int radius;
    public double circumference() {
        return 2 * PI * radius;
    }
}
```