CHAPTER 5

第5章

类和对象

本章学习目标：

- 掌握类的定义
- 了解包的定义及引入
- 熟练掌握变量的作用域
- 熟练掌握方法中参数的传递
- 熟练掌握方法重载
- 了解访问控制修饰符
- 了解类的封装性

5.1 类 和 包

在面向对象程序设计语言中，类是构成程序的基本单元，引进包机制是为了处理类重名的问题。

5.1.1 类的定义

类是用来定义对象的模板。类按照某种方式对对象进行分类，描述了对象应该包含的内容，可以通过类来创建对象（也称为类的实例）。

类的声明包括两部分：类名和类体。类体包括成员变量和成员方法。

成员变量用来描述类的属性。例如，定义一个学生类，成员变量就是用来描述学生的身高、体重、性别等自然属性。

成员方法用来描述类的功能。例如，定义一个机器人，成员方法就是用来刻画机器人能够通过何种方式辨别方向、与人交流等功能。

类的声明格式如下：

```
class 类名{
    成员变量
    成员方法
}
```

例如：

```
class Student{
    float height;
    float weight;
    void study{
    …
    }
}
```

一般 Java 程序需要多个类完成一项任务，但其中至少包含一个主类（包含 main 方法的类）。

5.1.2 包的声明和使用

包是为了处理导入类重名问题而引入的一种 Java 编程机制，可以将一组相关的类或接口（第 7 章讲解）封装在包（Package）里，从而更好地管理已经开发的 Java 代码。最大优点是增加代码复用率。

Java 语言提供了很多已经写好的包。

例如：

```
java.lang
java.io
java.util
java.sql
```

Java 语言也允许程序员自定义包，定义格式如下：

```
package 包名;
```

例如：

```
package sun.com.cn;
```

通过关键字 package 声明包语句。Package 语句作为 Java 源程序的第一条语句（注释除外），指明了该源程序定义的类所在的包。

如果源程序中省略了 package 语句，源程序中定义命名的类被隐含地认为是无名包的一部分，即源程序中定义命名的类在同一个包中，但该包没有名字。包名可以是一个合法的标识符，也可以是若干个标识符加“.”分割而成。

使用 import 语句可以引入包中的类。

在编写源程序时，除了自己编写类外，经常需要使用 Java 语言提供的许多类，这些类可能在不同的包中。

编写 Java 语言程序时，使用已经存在的类，可以避免一切从头做起，这是面向对象编程的一个重要方面。

在一个 Java 源程序中可以有多个 import 语句，它们必须写在 Package 语句和源程序中类的定义之间。

例如：

```
import java.lang.*;
import sun.com.cn.*
```

或者

```
import java.lang.*, sun.com.cn.*;
```

*代表的是可以引入包中任意的类，如果只引入包中的一个类，可以这样写：

```
import java.util.Arrays;
```

一般java.lang包中的类不需要在程序中引入，Java虚拟机会自动载入其中包含的类。

5.2 变　　量

类中定义的变量包括成员变量和局部变量两大类。成员变量定义在类体中，局部变量定义在成员方法中。

5.2.1 变量的声明

成员变量和局部变量可以定义为Java语言中的任何数据类型，包括简单类型和引用类型。

例如：

```
class A{
    int a=12;                                    //成员变量
    void f(){
    int x=34;                                    //局部变量
    }
}
```

5.2.2 成员变量的分类

成员变量可以分为实例成员变量和类成员变量(全局变量或静态成员变量)。如果一个成员变量的定义前有static关键字，那么它就是类成员变量(全局变量)，其他形式的成员变量都为实例成员变量。

例如：

```
class A{
    int a=12;                                    //实例成员变量
    static int b=34;                             //类成员变量
    void f(){
    int x=56;                                    //局部变量
    }
}
```

通过如下程序的运行结果可以清晰地看出实例成员变量和类成员变量的区别，这也是

为什么称类成员变量为全局变量的原因。不同对象的同名实例变量被分配不同的内存空间，如果类中的成员变量有类变量，那么所有对象的这个类变量都分配给相同的一块内存，改变其中一个对象的类变量值会影响其他对象中类变量的值。

例 5-1：类成员变量和实例成员变量的区别。

```
class A{
    static int x;
    int y;
        A(){
        x++;
        y++;
    }
}
class B{
    public static void main(String args[]){
    A a1 = new A();
        System.out.println("a1.x = " + a1.x);
        System.out.println("a1.y = " + a1.y);
        A a2 = new A();
    System.out.println("a2.x = " + a2.x);
    System.out.println("a2.y = " + a2.y);
        ++A.x;
        System.out.println("A.x = " + A.x);
    }
}
```

运行结果：

```
a1.x = 1
a1.y = 1
a2.x = 2
a2.y = 1
A.x = 3
```

5.2.3 成员变量与局部变量的作用域

成员变量可以定义在类体中的任意位置，它的作用域为整个类，也就是说，可以在类中的任何位置访问到它。

例如：

```
class A{
    static int x;
        A(){
        x++;
        y++;
    }
    int y;
}
```

局部变量的作用域从定义它的位置开始，直到定义它的语句块结束。

例如：

```
class A{
    int a = 12;
    void f(){int sum = 0;
        for(int i = 1;i <= 100;i++)
        sum = sum + i;
        System.out.println("i = " + i);                    //出现错误,访问不到 i
        System.out.println("sum = " + sum);                //可以访问到,在定义的语句块之内
        System.out.println("a = " + a);                    //可以访问到,作用域为整个类
    }
}
```

成员变量和局部变量可以重名，成员方法中访问重名的局部变量时，成员变量被隐藏。如果想在成员方法中访问重名的成员变量，需要在前面加关键字 this，稍后会讲解 this 的用法。

5.3 成 员 方 法

类体中定义的成员方法分为实例成员方法、类成员方法（静态成员方法）和构造方法，本节主要说明前两种成员方法，后面将对构造方法进行讲解。

5.3.1 成员方法的声明

类的功能在 Java 语言中被称为方法，在其他语言中被称为函数或过程，Java 语言中的方法不允许单独存在，它必须在一个类内部定义，这一点与 C++ 面向对象程序设计语言不同。

方法定义的格式：

```
[修饰符 1 修饰符 2 …]返回值类型 方法名(方法的形参列表){
    [方法体]
}
```

修饰符：修饰符可以有多个，顺序可任意颠倒，不影响成员方法的调用。

返回值类型：返回值的数据类型可以是简单类型，也可以是引用类型，当无返回值时返回类型为 void ，一个成员方法最多只能返回一个值。

方法名：可以是任何一个合法的标识符。

方法的形参列表：成员方法可以没有形参，若定义多个形参时，用逗号“,”分隔开，定义的成员方法被调用时，实参（外界调用方法时实际传进来的数据）对应形参，实参的类型、次序、数目，必须与形参的定义一致。

方法体：方法体是由描述方法功能的若干语句块组成。

例 5-2：定义成员方法，并通过对象调用。

```
class B{
   int getId( int id){
   return id;
```

```
    }
    public static void main(String args[]){
        B b = new B();
        System.out.print(b.getId(12));
        }
}
```

运行结果：

```
12
```

5.3.2 成员方法的分类

类体中的成员方法分为实例成员方法、类成员方法(静态成员方法)。

1. 类成员方法(静态成员方法)

用 static 修饰的方法称为静态方法，静态方法被调用时无须创建对象，直接用“类名.静态成员方法(参数列表)”调用即可。

对于类中的类方法，在该类被加载到内存时，就分配了相应的入口地址。从而类方法不仅可以被类创建的任何对象调用执行，也可以直接通过类名调用。类方法的入口地址直到程序退出才被取消。

类方法中出现的成员变量必须是被所有对象共享的类变量。

例 5-3：实例成员变量无法通过类直接访问。

```
class A{
    int z = 0;
    public static int add(int i,int j){
        int k;
        k = i + j;
        return k;
    }
}
public class B{
    public static void main(String args[]){
        int result = A.add(10,20);
        System.out.println("result = " + result);
        System.out.println("z = " + A.z);                    //无法访问到,必须创建对象
    }
}
```

运行结果：

```
result = 30
```

2. 实例成员方法

如果一个成员方法前有 static 关键字，那么它就是类成员方法(静态成员方法)，其他形式的成员方法都为实例成员方法。

当类的字节码文件被加载到内存时，类的实例方法不会被分配入口地址，当该类创建对象后，类中的实例方法才分配入口地址，从而实例方法可以被类创建的任何对象调用执行。

当创建第一个对象时，类中的实例方法就分配了入口地址，当再创建对象时，不再分配入口地址（地址被所有的对象共享），当所有的对象都不存在时，方法的入口地址才被取消。

5.3.3 参数传递

方法中定义的参数类型分为简单数据类型和引用数据类型。

当方法被调用时，如果方法有参数，参数必须要初始化，即参数变量必须有对应的实参值传入。在 Java 中，方法的所有参数都是“传值”的，也就是说，方法中参数变量的值是调用者指定的值的拷贝。

1. 简单数据类型的传值

对于基本数据类型的参数，向该参数传递的数据类型长度不可以大于该参数数据类型的长度。例如，不可以向 int 型参数传递一个 float 值，但可以向 double 型参数传递一个 float 型值。

另外，对于简单数据类型的传值，如果向方法的 int 型参数 x 传递一个 int 值，那么参数 x 得到的值是传递值的拷贝。方法中改变参数的值，不会影响向参数“传值”的变量的值。

2. 引用类型的传值

Java 的引用类型数据包括数组、字符串以及预定义和自定义的类。当参数是引用类型时，“传值”传递的是变量的引用而不是变量所引用的实体。如果改变参数变量所引用的实体，就会导致原变量的实体发生同样的变化。但是，改变参数的引用不会影响向其传值的变量的引用。

例 5-4：简单数据类型和引用类型作参数。

```
class A{
    int x = 1;
    void aa(int i){
        i = 5;
        System.out.println("i = " + i);
    }
    void bb(A a){
        a.x = 3;
        System.out.println("a.x" + a.x);
    }
}
public class B{
    public static void main(String args[]){
        int y = 0;
        A a1 = new A();
        a1.aa(y);
        System.out.println("y = " + y);
        a1.bb(a1);
```

```
        System.out.println("a1.x = " + a1.x);
    }
}
```

运行结果：

```
i = 5
y = 0
a. x3
a1.x = 3
```

5.4 对　　象

类是对象的模板。对象是一类事物中的一个特例。对象通过类来创建，一般通过 new 关键字来创建，也称为类的实例，任何一个对象都有其对应的类模板。

5.4.1 对象的创建

创建一个对象包括对象的声明和为对象分配内存两个步骤。

1. 对象的声明

一般格式为：

类的名字　对象名字;

例如：

```
Student s1;
```

2. 为声明的对象分配内存

使用 new 运算符和类的构造方法为声明的对象分配内存，如果类中没有构造方法，系统会调用默认的构造方法(默认的构造方法是无参数的，构造方法的名字必须和类名相同)。

例如：

```
s1 = new Student();
```

对象不仅可以操作自己的变量改变状态，而且还拥有了使用创建它的那个类中的方法的能力，对象通过使用这些方法可以产生一定的行为。

通过使用运算符“.”，对象可以实现对自己的变量访问和方法的调用。

声明对象和创建也可以写成如下形式：

```
Student s1 = new Student();
```

5.4.2 构造方法

构造方法通常用来初始化实例对象，如果类中定义成员变量已经赋值，那么初值为类中

定义的值，否则基本数值类型整型默认值为0，布尔类型为false；引用数据类型为null。

构造方法的特点是与类同名，没有返回值，可以含有多个构造方法（构造方法重载），但必须具有不同的参数列表。

一个类必须要有构造方法，如果没有显式地定义类的构造方法，则系统会为该类定义一个默认的构造方法。该构造方法不含任何参数。但是，如果在类中定义了构造方法，系统就不会再创建这个默认的不含参数的构造方法。

例5-5：构造方法的定义及重载。

```
class A{
    int x = 1;
    int y;
        A(){}
        A(int z){
            y = z;
    }
    void aa(){
        System.out.println("我是实例成员方法!");
    }
    public static void main(String args[]){
        A a1 = new A();
        A a2 = new A(8);
        System.out.println("a1.x = " + a1.x);
        a2.aa();
        System.out.println("a2.y = " + a2.y);
     }
}
```

运行结果：

```
a1.x = 1
我是实例成员方法!
a2.y = 8
```

5.4.3 this关键字

this是Java语言中一个非常重要的关键字，用来表示某个对象。它可以出现在实例成员方法和构造方法中，但不可以出现在类成员方法中。this关键字出现在类的实例成员方法中时，代表正在调用该方法的当前对象，要获取当前对象使用this。有时候方法的参数和类中的成员变量重名时，也要使用this。在构造方法中，除了可以使用this来给成员变量赋值之外，也可以使用this(参数列表)，来调用类中的其他构造方法，而且this(参数列表)必须放在第一行。

例5-6：this关键字的两种用法。

```
class A{
    int x = 1;
    boolean y;
    A(int x){
```

```
        this.x = x;
    }
    A(boolean z){
        this(10);
        y = z;
    }
    public static void main(String args[]){
        A a1 = new A(5);
        A a2 = new A(true);
        System.out.println("a1.x = " + a1.x);
        System.out.println("a1.y = " + a1.y);
        System.out.println("a2.x = " + a2.x);
        System.out.println("a2.y = " + a2.y);
     }
}
```

运行结果：

```
a1.x = 5
a1.y = false
a2.x = 10
a2.y = true
```

例如，图书进存销管理系统中的 CommonPanel.java 源文件中有 this 关键字的两种用法，一是处理成员变量和局部变量的重名，二是指代调用方法的对象。

```
package org.crazyit.book.ui;
import java.util.Vector,
import javax.swing.Box;
import javax.swing.BoxLayout;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JTable;
import javax.swing.table.DefaultTableModel;
/**
 * 各个 JPanel 的基类
 *
 * @author yangenxiong yangenxiong2009@gmail.com
 * @version  1.0
 * <br/>网站：<a href = "http://www.crazyit.org">疯狂 Java 联盟</a>
 * <br>Copyright (C), 2009 - 2010, yangenxiong
 * <br>This program is protected by copyright laws.
 */
public abstract class CommonPanel extends JPanel {
    //存放数据的 table
    private JTable table;
    //列表数据
    protected Vector<Vector> datas;
    public void setJTable(JTable table) {
        this.table = table;
```

```
}
public JTable getJTable() {
    return this.table;
}
public Vector<Vector> getDatas() {
    return datas;
}
public void setDatas(Vector<Vector> datas) {
    this.datas = datas;
}
/*
 * 将数据设置进 JTable 中
 */
public void initData() {
    if (this.table == null) return;
    DefaultTableModel tableModel = (DefaultTableModel)this.table.getModel();
//将数据设入表格 Model 中
    tableModel.setDataVector(getDatas(), getColumns());
    //设置表格样式
    setTableFace();
}
/*
 * 刷新列表的方法
 */
public void refreshTable() {
    initData();
    getJTable().repaint();
}
/*
 * 获取表列集合，由子类去实现
 */
public abstract Vector<String> getColumns();
/*
 * 设置列表的样式，由子类去实现
 */
public abstract void setTableFace();
/*
 * 设置数据列表的方法,由子类实现
 */
public abstract void setViewDatas();
/*
 * 清空界面下边的列表
 */
public abstract void clear();
/*
 * 分隔用的 box
 */
public Box getSplitBox() {
    Box box = new Box(BoxLayout.X_AXIS);
    box.add(new JLabel("              "));
    return box;
```

```
    }
    //给子类使用的方法，用于获取一个列表的 id 列值
    public String getSelectId(JTable table) {
        int row = table.getSelectedRow();
        int column = table.getColumn("id").getModelIndex();
        String id = (String)table.getValueAt(row, column);
        return id;
    }
    //显示警告
    protected int showWarn(String message) {
        return JOptionPane.showConfirmDialog(this, message, "警告",
                JOptionPane.OK_CANCEL_OPTION);
    }
}
```

5.4.4 垃圾回收机制

Java 语言中很多对象的数据(成员变量)都存放在堆中，对象从中分配空间。Java 虚拟机的堆中储存着正在运行的应用程序所建立的所有对象，这些对象通过 new 等关键字建立，但是它们不需要程序代码来显式地释放。一般来说，堆中的无用对象是由垃圾回收器来负责的，尽管 Java 虚拟机规范并不要求特殊的垃圾回收技术，甚至根本就不需要垃圾回收，但是由于内存的有限性，虚拟机在实现的时候都有一个由垃圾回收器管理的堆。垃圾回收是一种动态存储管理技术，它自动地释放不再被程序引用的对象，按照特定的垃圾收集算法来实现资源自动回收的功能。

当一个对象不再被引用时，垃圾收集器会在适当的时候将其销毁，适当的时候，即意味着无法预知何时垃圾收集器会进行工作，当对象被回收的时候，finalize()方法会被虚拟机自动调用，如果重写了 finalize()方法，这个方法也会被自动执行(Object 类中的一个 protected 方法)。finalize()方法一般用户不自行调用。

5.5 方法的重载

方法重载也被称为功能多态性，是指可以向功能(方法)传递不同的消息(参数)，以便让对象根据相应的消息来产生相应的行为。

方法重载是指一个类中可以有多个方法具有相同的名字，但这些方法的参数必须不同，即或者是参数的个数不同，或者是参数的类型不同。返回值不能作为方法重载的判断依据。

重载的方法被调用时，实参与形参类型完全一致时，直接调用匹配的重载方法；如实参与形参类型不一致时，则少字节类型向多字节类型匹配。例如，实参 int 类型将匹配形参 int 类型，如果没有对应形参类型，则匹配 long、float 或者 double 类型，但 int 类型不会匹配 short 等比它少的字节类型(因为多字节类型变量就不允许未经过强制类型转换而直接赋值给少字节类型的变量)。

例 5-7：成员方法重载。

```
class A{
```

```
    void aa (int x){
        System.out.println("x = " + x);
    }
    double aa(double y){
        return y;
    }
    public static void main(String args[]){
        A a1 = new A();
        a1.aa(10);
        System.out.println("y = " + a1.aa(10.0F));
    }
}
```

运行结果：

```
x = 10
y = 10.0
```

5.6 访问权限修饰符

所谓的访问权限是指成员变量和成员方法能够被访问到的位置。在 Java 语言中共有 4 种访问权限，分别是 public(公有的)、protected(受保护的)、无修饰符、private(私有的)。

5.6.1 public

public 可以用来修饰类、类成员和构造方法。修饰类表示类为公有的，在一个 Java 源程序中只允许有一个类被 public 修饰，这样的类可以用来在任何包中创建对象。用 public 修饰的成分访问权限最高。

类成员被 public 修饰表示该成员为公有的，可以在任何包中访问到该成分。

例如：

```
public class A{
    public int x;
    public void aa(){
        System.out.println("x = " + x);
    }
}
```

5.6.2 protected

protected 修饰的成分被称为受保护的成员。protected 可以修饰成员方法和成员变量，但是不能修饰类。

protected 修饰的成员可以分为以下两种情况来讨论。

1. 同一个包

protected 修饰的成分在同一个包中被调用时，直接通过类创建的对象即可调用。

例如：

```
package a;
class A{
    protected int x;
    protected void aa(){
        System.out.println("x = " + x);
    }
}
```

2. 不同包

protected 修饰的成分不在同一个包中被调用时，要求一定要具有继承关系，即 protected 修饰的成分位于父类中，父类位于 a 包中，子类位于 b 包中，子类继承父类中的 protected 修饰成分，这样的情况，可以在 b 包中访问到 a 包中 protected 修饰成分，即在 b 包中创建子类对象可以访问到从 a 包中继承下来的 protected 修饰成分。

例 5-8：定义包与引入包中的类。

```
------------------ 定义在 A.java 源程序中 ------------------
package a;
class A{
    protected int x;
    protected void aa(){
        System.out.println("x = " + x);
    }
}
------------------ 定义在 B.java 源程序中 ------------------
package b;
import a.*;
class B extends A{
    protected int y;
    protected void bb(){
        System.out.println("y = " + y);
    }
}
public class C{
    public static void main(String args[]){
        B b1 = new B();
        System.out.println(b1.x);
        System.out.println(b1.y);
        b1.aa();
        b1.bb();
    }
}
```

运行结果：

```
0
0
0
0
```

5.6.3 无修饰符

无修饰符权限即默许权限，类、类成员和构造方法都可以使用无修饰符权限，即在前面不加任何关键字。无修饰符权限的成分在同一个包中可以访问到。

例 5-9：无修饰符权限的成分在同一个包中可以访问到。

```
class A{
      A(){}
      void aa(){
            System.out.println("A.aa()");
      }
public class B{
      public static void main(String[ ] args){
            A a1 = new A();
            a1.aa();
      }
}
```

运行结果：

```
A. aa()
```

注：没有 package 语句的源程序默认在一个无名包中。

5.6.4 private

private 可以用来修饰类成员和构造方法。这些数据和方法只能在定义它的类中被访问到，在其他类中不能调用。

例 5-10：private 访问权限的成分只能在同一个类中访问。

```
class A{
    private A(){}
    A(int i){}
    static int aa(){
        return 1;
    }
}
public class B{
    public static void main(String[ ] args){
    // A 类中构造函数 A()是 private,
    // 所以不能用它进行初始化
    //A a1 = new A();
    A a2 = new A(1);                              //A(int)是无修饰符访问权限,可以在此调用
    System.out.println(a2.aa());
    }
}
```

运行结果：

```
1
```

具有 private 权限的属性(成员变量)和操作(成员方法),只有在类内部才可以访问。而具有 public 权限的属性和操作,在任何地方(也就是其他任何对象)都可以访问。在类中声明为 protected 权限的属性和操作,它的可访问程度介于 private 和 public 之间,可以在类内部或类的所有子类(包括子类的子类)或同一包中的其他类访问。4 种访问权限详情可见表 5-1。

表 5-1 Java 4 种访问权限表

修饰符	同一个类	同一个包	不同包子类	任何包
public	√	√	√	√
protected	√	√	√	
无修饰符	√	√		
private	√			

5.7 封 装 性

Java 语言中类的封装性是它的三大特性之一。简单地说,就是将属性私有化,提供相应的 set 和 get 方法来操纵类的属性。如果有一个带参的构造函数,那么一定要再写一个不带参的构造函数。

封装性是隐藏对象的属性和实现细节,仅对外公开接口,控制在程序中属性的读取和修改的访问级别。将抽象得到的数据和行为(或功能)相结合,形成一个有机的整体,也就是将数据与操作数据的源代码进行有机的结合,形成“类”,其中数据和方法都是类的成员。

封装的目的是增强安全性和简化编程,编程者不必了解具体的实现细节,而只是要通过外部接口这一特定的访问权限来使用类的成员。

小 结

类是用来定义对象的模板。类按照某种方式对对象进行分类,类描述了对象应该包含的内容,可以通过类来创建对象(也称为类的实例)。

包是为了处理导入类重名问题而引入的一种 Java 编程机制,可以将一组相关的类或接口封装在包(Package)里,从而更好地管理已经开发的 Java 代码。

通过关键字 package 声明包语句。package 语句作为 Java 源程序的第一条语句(注释除外),指明了该源程序定义的类所在的包。

使用 import 语句可以引入包中的类。

类中定义的变量包括成员变量和局部变量两大类。成员变量定义在类体中,局部变量定义在成员方法中。成员变量可以定义在类体中的任意位置,它的作用域为整个类,也就是说,可以在类中的任何位置访问到它。局部变量的作用域从定义它的位置开始,直到定义它的语句块结束。

成员方法定义的格式:

[修饰符 1 修饰符 2 …]返回值类型 方法名(方法的形参列表){

```
    [方法体]
}
```

用 static 修饰的方法称为静态方法，静态方法被调用时无须创建对象，直接用“类名.静态成员方法(参数列表)”调用即可。如果一个成员方法前没有 static 关键字，那么它就是实例成员方法，此方法必须创建对象才能访问。

方法中定义的参数类型分为简单数据类型和引用数据类型。

对于基本数据类型的参数，向该参数传递的数据类型长度不可以大于该参数数据类型的长度。当参数是引用类型时，“传值”传递的是变量的引用而不是变量所引用的实体。

类是对象的模板。对象是一类事物中的一个特例。对象通过类来创建，一般通过 new 关键字来创建，也称为类的实例，任何一个对象都有其对应的类模板。

this 是 Java 语言中一个非常重要的关键字，用来表示某个对象。它可以出现在实例成员方法和构造方法中，但不可以出现在类成员方法中。this 关键字出现在类的实例成员方法中时，代表正在调用该方法的当前对象，要获取当前对象使用 this。有时候方法的参数和类中的成员变量重名时，也要使用 this。在构造方法中，除了可以使用 this 来给成员变量赋值之外，也可以使用 this(参数列表)，来调用类中的其他构造方法，而且 this(参数列表)必须放在第一行。

方法重载也被称为功能多态性，是指可以向功能(方法)传递不同的消息(参数)，以便让对象根据相应的消息来产生相应的行为。

方法重载是指一个类中可以有多个方法具有相同的名字，但这些方法的参数必须不同，即或者是参数的个数不同，或者是参数的类型不同。返回值不能作为方法重载的判断依据。

所谓的访问权限是指成员变量和成员方法能够被访问到的位置。在 Java 语言中共有 4 种访问权限，分别是 public(公有的)、protected(受保护的)、无修饰符、private(私有的)。

封装性是隐藏对象的属性和实现细节，仅对外公开接口，控制在程序中属性的读取和修改的访问级别。将抽象得到的数据和行为(或功能)相结合，形成一个有机的整体，也就是将数据与操作数据的源代码进行有机的结合，形成“类”，其中数据和方法都是类的成员。

封装的目的是增强安全性和简化编程，编程者不必了解具体的实现细节，而只是要通过外部接口这一特定的访问权限来使用类的成员。