第 3 章

Java 类与对象

学习目的和要求

Java 是面向对象的程序设计语言。通过对本章的学习，理解面向对象编程的基本概念和思想，掌握类和对象的关系，学会如何使用类的定义、创建对象，并将其应用于程序设计中，从而掌握 Java 语言的面向对象程序设计方法。

主要内容

- 面向对象程序设计
- 类与对象
- 对象的创建与使用
- 构造方法
- 方法的重载
- 参数传递
- static 关键字
- this 关键字
- 包
- impot
- Java 访问权限
- Java 基本数据类型的类封装
- 垃圾回收

3.1　面向对象程序设计

3.1.1　面向对象程序设计的思想

面向对象程序设计（OOP）是 90 年代才开始流行的一种软件编程方法。传统的用结构化方法开发的软件，其稳定性、可修改性和可重用性都比较差。这是因为结构化方法的本质是功能分解，先从代表目标系统整体功能的单个处理着手，自顶向下不断把复杂的处理分解为子处理，这样一层一层地分解下去，直到仅剩下若干个容易实现的子处理为止，再用相应的工具来描述各个底层的处理。因此，结构化方法是围绕实现处理功能的“过程”来构造系统的。然而，用户需求的变化大部分是针对功能的，因此这种变化对基于过程的设计方法来说是灾难性的。用这种方法设计出来的系统结构常常是不稳定的，用户需求的变化往往会造成系统结构的较大变化，从而需要花费很大代价才能实现这种变化。

所谓的面向对象程序设计，就是把面向对象的思想应用到软件工程中，并指导开发和维护软件。面向对象的概念和应用领域已经超越了程序设计和软件开发，扩展到了数据库系统、交互式界面、应用平台、分布式系统、网络管理结构和人工智能等领域中。

最开始的“面向对象”专指在程序设计中采用封装、继承、抽象等设计方法。现在面向对象的思想已经涉及软件开发的各个方面，主要包括如下方面。

- 面向对象分析（object-oriented analysis，OOA）。
- 面向对象设计（object-oriented design，OOD）。
- 面向对象程序设计（object-oriented program，OOP）。

面向对象程序设计具有结构化程序设计的特点，包括将客观事物看作具有属性和行为的对象；不再将问题分解为过程，而是将问题分解为对象，即一个复杂对象由若干个简单对象构成；通过抽象找出同一类对象的共同属性和行为并将其形成类；通过消息实现对象之间的联系，构造复杂系统；通过类的继承与多态实现代码重用。

面向对象程序设计的优点是，使程序能够比较直接地反映问题域的本来面目，使软件开发人员能够利用人们认识事物所采用的一般思维方法来进行软件开发。

3.1.2　面向对象程序设计的特点

面向对象程序设计强调对象的抽象、封装、继承、多态，具有如下特点。

1．封装

封装也称信息隐藏，是指将一个类的使用和实现分开，只保留部分接口、方法与外部

联系，或者只公开一些供软件开发人员使用的方法。软件开发人员只需要关注这个类如何使用，而无须关心其具体的实现过程，这样就能实现 MVC（Model、View 和 Controller）分工合作，也能有效避免程序间的相互依赖，实现代码模块间的松耦合。

2．继承

继承是指子类自动继承父类的属性和方法，软件开发人员可以为其添加新的属性和方法，或者对部分属性和方法进行重写。继承增强了代码的可重用性。

3．多态

子类继承了来自父类的属性和方法，并且其中的部分方法被重写。于是多个子类虽然都具有同一个方法，但是其实例化的对象在调用这些相同的方法后可以获得完全不同的结果，这种技术就是多态。多态增强了软件的灵活性。

3.2 类与对象

类用于让程序设计语言能更清楚地描述日常生活中的事物。类是对某一类事物的描述，是抽象的、概念上的定义。而对象是实际存在的属于该类事物的具体个体，因而也称实例（Instance）。类是对象的模板、图纸，而对象是类的一个实例，是实实在在的个体。一个类可以对应多个对象。如果将对象比作汽车，那么类就是汽车的设计图纸。所以面向对象程序设计思想的重点是类的设计，而不是对象的设计。

一般来说，类是由数据成员与函数成员封装而成的，其中数据成员表示类的属性，函数成员（程序代码）表示类的行为。由此可见，类描述了对象的属性和对象的行为。在 Java 语言中，数据成员被称为域变量、属性、成员变量等，而函数成员被称为成员方法，简称方法。所谓的类就是把事物的数据与相关功能封装在一起，形成一种特殊的数据结构，用一种抽象来表达真实事物。

3.2.1 类的定义

由于类是一种将数据和方法封装在一起的数据结构，其中数据表示类的属性，方法表示类的行为，所以定义类实际上就是定义类的属性与方法。用户定义一个类实际上就是定义一个新的数据类型。在使用类之前，必须先定义它，然后才可以利用所定义的类来声明相应的变量，并创建对象。类的基本语法格式如下。

```
[类的修饰符] class 类名称 [extends 父类名][implements 接口声名列表] {
    // 变量的声明：用来体现对象的属性
```

```
    //方法的定义：方法可以对类中声明的变量进行操作，体现对象所具有的行为
}
```

定义类又称声明类，示例如下。

```
public class Person {
    private String name;
    private int age;
    public Person() {
    }
    public Person(String name, int age) {
        this.name = name;    //分别指成员变量 name 和形参 name
        this.age = age;
    }
    public void setAge(int i) {
        if (i < 0 || i > 130)
            return;
        age = i;
    }
    public int getAge() {
        return age;
    }
}
```


该类中包含以下两类变量。

（1）局部变量：在方法、形参或者语句块中定义的变量被称为局部变量。变量声明和初始化都是在方法中进行的，在方法结束后，变量就会自动销毁。

（2）成员变量：成员变量是定义在类中，方法体之外的变量。这种变量会在创建对象的时候实例化。成员变量可以被类中方法、构造方法和特定类的语句块访问。

3.2.2　访问控制符

在 Java 语言中，可以使用访问控制符来保护对类、变量、方法和构造方法的访问。Java 语言支持 4 种不同的访问权限。

（1）public：对所有类可见，其使用对象有类、接口、变量、方法。

（2）protected：对同一包内的类和所有子类可见，其使用对象有变量、方法。

（3）default（默认，什么也不写）：在同一包内可见，不使用任何修饰符，其使用对象有类、接口、变量、方法。

（4）private：在同一类内可见，其使用对象有变量、方法。

3.2.3 类的封装

在面向对象程序设计方法中，封装（Encapsulation）是一种将抽象函数接口的实现细节进行包装、隐藏的方法。封装可以被认为是一个保护屏障，用于防止该类的代码和数据被外部类定义的代码随机访问。要访问该类的代码和数据，必须通过严格的接口控制。

封装最主要的功能在于使程序员能修改自己的实现代码，而不用修改那些调用代码的程序片段。适当的封装可以让程序代码更容易理解与维护，也加强了程序代码的安全性。

封装的优点如下。

（1）良好的封装能够减少耦合。

（2）可以自由修改类的内部结构。

（3）可以对成员变量进行更精确的控制。

（4）隐藏信息，实现细节。

在定义类时，将成员变量设为 private，而将成员方法设为 public，以避免程序的其他部分直接操作类的内部数据，这实际就是数据封装思想的体现。

3.3 对象的创建与使用

对象是整个面向对象程序设计的理论基础，由于在面向对象程序中使用类来创建对象，所以可以将对象理解为一种新型的变量。对象中保存着一些比较有用的数据，程序员可以要求它对自身进行操作。对象之间靠传递消息而相互作用，传递的结果是启动了方法，完成一些行为或者修改接收消息的对象的属性。对象一旦完成了它的工作就会被销毁，所占用的资源将被系统回收以供其他对象使用。

```
public class TestPerson {
    public static void main(String args[]) {
        Person p1 = new Person();
        p1.setAge(3);
        p1.setAge(-6);
        Person p2 = new Person("Jack", 18);
        new Person().setAge(18);    //匿名对象
        System.out.println(p1.getAge());
        System.out.println(p2.getAge());
    }
}
```

3.3.1 对象与匿名对象的创建

由于对象是类的实例，所以对象属于某个已知的类，因此要创建属于某个类的对象，可以通过以下两个步骤来完成。

（1）声明指向由类创建的对象的变量。

（2）利用 new 运算符创建新的对象，并指派给前面创建的变量。

在一个对象被创建之后，在调用该对象的方法时，也可以不定义对象的引用变量，而直接调用这个对象的方法，这样的对象被称为匿名对象，如下所示。

```
new Person().setAge(18);      //匿名对象
```

使用匿名对象通常有以下两种情况。

（1）如果对一个对象只需要进行一次方法调用，那么可以使用匿名对象。

（2）将匿名对象作为实参传递给一个方法调用。

3.3.2 对象的使用

在创建新的对象之后，就可以对对象的成员进行访问了。通过对象来引用对象成员的语法格式如下。

```
    对象名.对象成员
```

对象名和对象成员之间用“.”相连，通过这种引用方式可以访问对象的成员。如果对象成员是成员变量，则通过这种引用方式可以获取或修改类中成员变量的值。

3.4 构造方法

3.4.1 构造方法的作用与定义

构造方法（Constructor）是一种特殊的方法，它是在对象被创建时初始化对象成员的方法。构造方法的名称必须与它所在的类名完全相同。构造方法没有返回值，但在定义构造方法时，构造方法的方法名前不能用 void 修饰符来修饰，这是因为一个类的构造方法的返回值就是该类本身。在定义构造方法之后，创建对象时会自动调用它，因此构造方法不需要在程序中直接调用，而是在对象创建时自动调用并执行，这一点不同于一般的方法（一般的方法在用到时才调用）。

构造方法的特征主要体现在以下几个方面。

（1）构造方法的方法名与类名相同。

（2）构造方法没有返回值，但不能用 void 修饰符来修饰。

（3）构造方法的主要作用是完成对类的对象的初始化。

（4）构造方法一般不能由程序员显式地直接调用，而是用 new 运算符来调用。

（5）在创建一个类的对象的同时，系统会自动调用该类的构造方法对新对象进行初始化。

3.4.2 默认的构造方法

如果没有在类中定义构造方法，那么依然可以创建新的对象。Java 编译器会自动为该类生成一个默认的构造方法（Default Constructor），在程序中创建对象时会自动调用该方法。

3.5 方法的重载

方法的重载是实现多态的一种方法。在面向对象程序设计语言中，有一些方法的含义相同，使用相同的名称，但带有不同的参数，这就叫作方法的重载（Overloading）。也就是说，重载是指在同一个类中具有相同名称的多个方法，在这些同名的方法中，如果其参数个数不同，或者参数个数相同但类型不同，则这些同名的方法具有不同的功能。

```
public class Overload {
    public void test() {
        System.out.println("无参方法");
    }
    public void test(String msg) {
        System.out.println("重载的方法 " + msg);
    }
    public static void main(String[] args) {
        Overload ol = new Overload();
        ol.test();
        ol.test("hello");
    }
}
```

3.6 参数传递

方法中最重要的部分是方法的参数，参数属于局部变量。当对象调用方法时，参数会

被分配内存空间，调用者需要向参数中传递值，即通过实参和形参进行值的传递。

在 Java 语言中，方法中参数变量的值是调用者指定的值的复制，如果改变参数的值，则不会影响向参数中传递值的变量。特别的，对于引用型数据，包括数组、对象和接口，当其参数是引用类型时，传递的值是变量中存放的“引用”，而不是变量所引用的实体。这样，对于两个相同类型的引用型变量，如果具有同样的引用，则会拥有同样的实体。因此，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；如果改变参数中存放的“引用”，则不会影响向其传递值的变量中存放的“引用”。

【例 3.1】方法参数传递示例

```
package c03;
public class Example3_01 {
    public static void change(int i, char[] c, PassTest obj) {
        i = 3;
        c[0] = 'A';
        obj.x = 3;
    }
    public static void main(String[] args) {
        PassTest obj = new PassTest();
        obj.x = 5;
        int i = 5;
        char[] c = {'a','b'};
        change(i,c,obj);
        System.out.println(i);
        System.out.println(c[0]);
        System.out.println(obj.x);
    }
}
class PassTest{
    int x;
}
```


3.7　static 关键字

static 被称为静态修饰符。在声明成员变量时，用 static 关键字修饰的变量称作类变量，也称 static 变量或静态变量，而没有用 static 关键字修饰的变量称作实例变量。与静态变量相似，用 static 修饰符修饰的方法属于类的静态方法，又称类方法。静态方法实质上是

属于整个类的方法，而不加 static 修饰符的方法是属于某个具体对象的方法，被称为实例方法。

3.7.1 实例变量与类变量

首先，不同对象的实例变量互不相同。在一个类中使用 new 运算符可以创建多个不同的对象，这些对象将被分配不同的实例变量，分配给不同的对象的实例变量占有不同的内存空间，改变其中一个对象的实例变量不会影响其他对象的实例变量。

其次，所有对象共享类变量。如果类中有类变量，则当使用 new 运算符创建多个不同的对象时，分配给这些对象的类变量占有相同的内存，改变其中一个对象的类变量会影响其他对象的类变量，这就是对象共享类变量。

最后，类变量不仅可以通过某个对象访问，还可以直接通过类名访问。而对象的实例变量只能通过该对象访问，而不能使用类名访问。原因是当执行 Java 程序时，类的字节码文件会被加载到内存中，如果该类没有创建对象，则类中的实例变量不会被分配内存空间。但是，类中的类变量在该类被加载到内存时，就被分配了相应的内存空间。如果该类创建对象，那么不同对象的实例变量互不相同，即分配不同的内存空间；而类变量不再重新分配内存空间，所有对象共享类变量，即所有对象的类变量都占用相同内存空间，直到程序退出运行才释放所占内存空间。类变量的优点是可以减少开辟新的内存空间，直接引用固有变量，缺点是会造成对象之间的耦合。

3.7.2 实例方法与类方法

对象可以调用实例方法。当类的字节码文件被加载到内存中时，类中的实例方法不会被分配入口地址，只有在该类创建对象后，类中的实例方法才会被分配入口地址，从而被类创建的任何对象调用和执行。需要注意的是，在创建第一个对象时，类中的实例方法就被分配了入口地址，当再次创建对象时，无须分配入口地址。也就是说，方法的入口地址被所有的对象共享，当所有的对象都不存在时，方法的入口地址才会被取消。

在实例方法中不仅可以操作实例变量，还可以操作类变量。当对象调用实例方法时，该方法中出现的实例变量和类变量都是分配给该对象的变量，只不过其类变量要和其他所有的对象共享而已。

对于类方法，既可以通过类名调用，也可以通过对象名调用。在类被加载到内存中时，就分配到了相应的入口地址，因此类方法不仅可以被类创建的任何对象调用执行，还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会被取消。需要特别注意的是，实例方法不能通过类名调用，是因为在类创建对象之前，实例变量还没有被分配内存空间，所以类方法不能操作实例变量。

如果一个方法不需要操作类中的任何实例变量就可以满足程序的需要，那么可以考虑将这样的方法设计为一个 static 方法，并且推荐通过类名直接调用。

【例 3.2】静态成员和实例成员示例

```
package c03;
public class Example3_02 {
    public static void main(String[] args) {
        Chinese c1 = new Chinese("Jack");
        Chinese c2 = new Chinese("JOJO");
        Chinese.singOurCountry();
        c1.singOurCountry();
        c2.sing();
        Chinese.country = "American";
        c2.sing();
    }
}

class Chinese {
    static String country = "中国";
    String name;
    int age;
    public Chinese() {
    }
    public Chinese(String name) {
        this.name = name;
    }
    void sing() {
        System.out.println(country + name);
    }
    static void singOurCountry() {
        System.out.println(country );
    }
}
```


3.7.3　静态初始化块

静态初始化块是由 static 关键字修饰的，由一对花括号构成的语句组。它的作用与类的构造方法有些相似，都是用来做初始化工作的，它们的不同点如下。

（1）构造方法会对每个新创建的对象进行初始化，而静态初始化块会对类自身进行初始化。

（2）构造方法是在用 new 运算符创建新对象时由系统自动执行的，而静态初始化块一般不能由程序调用，它在类被加载入内存时由系统调用执行。

（3）用 new 运算符创建多少个新对象，构造方法就会被调用多少次，而静态初始化块只会在类被载入内存时执行一次，与创建多少个对象无关。

（4）不同于构造方法，静态初始化块不是方法，因此没有方法名、返回值和参数。

```
static {
   int x = 8 ;
   System.out.println x );
}
```

3.8 this 关键字

this 关键字用于在 Java 语言中表示某个对象，可以出现在实例方法和构造方法中，但不可以出现在类方法中，使用方式基本有 3 种。

【例 3.3】this 关键字的 3 种用法示例

```
package c03;
public class Example3_03 {
   public static void main(String[] args) {
      Person p = new Person("Jack",18);
      p.ageAdd().ageAdd();
      System.out.println(p.age);
   }
}
class Person {
   String name;
   int age;
   public Person() {
   }
   public Person(String name) {
      this.name = name;      //1. 指代类变量
   }
   public Person(String name, int age) {
      this(name);            //2. 指代 Person(String name)构造方法
```

```
        this.age = age;
    }
    public void setName(String name) {
        this.name = name;
    }
    public Person ageAdd() {
        this.age++;
        return this;        //3. 指代对象自己
    }
}
```

3.9　包

包（Package）是 Java 语言提供的一种区别类名空间的机制，是类的组织方式。每个包对应一个文件夹，包中可以嵌套其他包，称为包等级结构。包是 Java 语言管理类的一个机制，不同的 Java 源文件中可能出现名字相同的类，如果想区分这些类，就需要使用包名。

在使用 package 关键字声明包时，包语句应该作为 Java 源文件中的第一条语句，指明该源文件定义的类所在的包即该源文件中声明的类指定包，一般的语法格式如下。

```
package 包名;
```

包名既可以是一个合法的标识符，也可以是若干个标识符加“.”，示例如下。

```
package cn.edu.zust.itee;
```

包等级结构中的根文件夹是由环境变量 ClassPath 来确定的。在 Java 源文件中若没有使用 package 关键字声明类所在的包，则 Java 默认包的路径是当前文件夹。默认包没有包名，即无名包，无名包中不能有子包。如果一个类有包名，那么就不能在任意位置存放，否则虚拟机将无法加载这样的类。

JDK 的常用包有 java.lang、java.io、java.util、java.net、java.awt、javax.swing 等。

3.10　import

一个类可能需要将另一个类声明的对象作为自己的成员或方法中的局部变量，如果这两个类在同一个包中，那么自然没有问题。但是，如果这两个类不在同一个包中，这时就必须使用 import 语句。

使用 import 语句可以引入包中的类，在编写源文件时，除了自己编写类，还经常需要

使用 JDK 提供的许多类，这些类可能在不同的包中。为了使用 Java 语言提供的类，可以使用 import 语句引入包中的类。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（假如有 package 语句）和源文件中类的定义之间。如果要引入一个包中的全部类，则可以用通配符号星号“*”来代替，但请注意，使用“*”只能表示本层次的所有类，而不包括子层次下的类，示例如下。

```
import java.util.*;       //表示引入 java.util 包中所有的类
import java.util.Date;    //表示引入 java.util 包中的 Date 类
```

另外，Java 编译器会为所有程序自动隐式导入 java.lang 包，因此程序员无须用 import 语句导入便可使用其中的类。

3.11 Java 访问权限

所谓访问权限是指对象是否可以通过“.”运算符操作自己的变量或调用类中的方法。访问控制符有 private、protected 和 public，这些都是 Java 的关键字，用来修饰类、成员变量和方法。

封装也称信息隐藏，是面向对象程序设计最重要的特性之一，使用访问控制符可以实现封装性。基本原则是对类中成员变量（数据）封闭，不允许外部程序直接访问，而是通过该类提供的方法来实现对隐藏信息的操作和访问，提供开放的 getter()方法和 setter()方法，即通过 private 数据，利用 public 接口实现对对象属性的访问。

1．private

用 private 修饰的类成员，只能被该类自身的方法访问和修改。

2．default（默认）

如果一个类或类中成员没有访问控制符，则说明它具有默认的访问控制特性。这种默认的访问控制特性规定，该类只能被同一个包中的类访问和引用，而不能被其他包中的类使用，这种访问特性又称包访问性。

3．protected

用保护访问控制符 protected 修饰的类成员可以被 3 种类访问：该类自身、与它在同一个包中的其他类，以及在其他包中的该类的子类。使用 protected 的主要作用是，允许其他包中的子类来访问其父类的特定属性和方法，否则可以使用默认访问控制符 default。

4．public

当一个类或类中成员被声明为 public 时，就可以被其他任何类所访问了，该类中被设

定为 public 的方法也称这个类对外的接口。

【例 3.4】访问控制符使用示例

```
package c03;
public class Example3_04 {
    public static void main(String[] args) {
        Student s = new Student();
        s.mail = "Jack@163.com";
        s.name = "Jack";
        //stu.no = "1001";  //非法访问
        s.setNo(1001);
        s.getNo();
        System.out.println(s.name + s.getNo());
    }
}
class Student {
          private int no;
          String name;
          protected int age;
          public String mail;
          public int getNo() {
              return no;
          }
          public void setNo(int no) {
              this.no = no;
          }
    }
```


3.12　Java 基本数据类型的类封装

Java 基本数据类型包括 boolean、byte、short、char、int、long、float 和 double。Java 语言同时提供了与基本数据类型相关的类，实现了对基本数据类型的封装。这些类在 java.lang 包中，分别对应 Boolean、Byte、Short、Character、Integer、Long、Float、Double 类。

【例 3.5】Character 类的使用示例

```
package c03;
public class Example3_05 {
```

```
  public static void main(String args[ ]) {
     char[] c={'a','b','c','D','E','F'};
     for(int i=0;i<c.length;i++) {
        if(Character.isLowerCase(c[i])) {
           c[i]=Character.toUpperCase(c[i]);
       }
       else if(Character.isUpperCase(c[i])) {
           c[i]=Character.toLowerCase(c[i]);
       }
     }
        for (char value : c) System.out.print(" " + value);
  }
}
```

将基本数据类型转换为包装类的过程被称为装箱，如把 int 型的对象包装成 Integer 类。将包装类转换为基本数据类型的过程被称为拆箱，如把 Integer 类的对象重新简化为 int 型。

手动实例化一个包装类被称为手动拆箱/装箱。Java 1.5 版本之前必须手动拆箱和装箱，之后可以自动进行，即在进行基本数据类型和对应的包装类转换时，系统将自动进行装箱及拆箱操作，示例如下。

```
Integer x = 100, y = 200;                    //自动装箱
Integer x=new Integer(100);                  //手动装箱
Integer y=new Integer(200);
Integer z = x + y;                           //自动拆箱再装箱
Integer(x.intValue()+y.intValue());          //手动拆箱再装箱
```

【例 3.6】Integer 类的使用示例

```
package c03;
public class Example3_07 {
  public static void main(String[] args) {
     int num = 100;
     String str = Integer.toString(num);          // 将数字转换成字符串
     String str1 = Integer.toBinaryString(num);   // 将数字转换成二进制
     String str2 = Integer.toHexString(num);      // 将数字转换成八进制
     String str3 = Integer.toOctalString(num);    // 将数字转换成十六进制
     System.out.println(str + "的二进制数是：" + str1);
     // 获取 int 型可取的最大值
     System.out.println("Integer 的最大值：" + Integer.MAX_VALUE);
```

```
        // 获取 int 型的二进制位
        System.out.println("Integer 的最大值: " +  Integer.SIZE );
    }
}
```

3.13　垃圾回收

JRE 提供了一个系统的垃圾回收器，负责自动回收那些没有被引用的对象所占用的内存空间，这种清除无用对象进行内存回收的过程就叫作垃圾回收。

垃圾回收有两个好处。

（1）它把程序员从复杂的内存追踪、监测、释放等工作中解放了出来。

（2）它防止了系统内存被非法释放，从而使系统更加稳定。

同时，垃圾回收还有 3 个特点。

（1）只有当一个对象不被任何引用类型的变量使用时，它占用的内存空间才可能被垃圾回收器回收。

（2）不能通过程序强迫垃圾回收器立即执行。垃圾回收器负责释放没有引用与之关联的对象所占用的内存空间，但是回收的时间对程序员是透明的。在任何时候，程序员都不能通过程序强制垃圾回收器立即执行，但可以通过调用 System.gc()或者 Runtime.gc()方法提示垃圾回收器进行内存空间回收操作，但不能确保垃圾回收器立即执行。

（3）当垃圾回收器要释放无用对象占用的内存空间时，会先调用该对象的 finalize()方法。在 Java 语言中，对象的回收是由系统进行的，但有一些任务需要在回收时进行，如清理一些非内存资源、关闭打开的文件等。这可以通过覆盖对象中的 finalize()方法来实现，因为系统在回收时会自动调用对象的 finalize()方法。

3.14　小结

1．类是对象的模板，对象是类的实例。

2．封装、继承、多态是面向对象程序设计的三大要素。

3．封装是指把变量和方法包装在一个类内，以限定成员的访问，从而达到保护数据的目的。

4．this 关键字指代对象本身或代表调用该成员的对象。

5．在一个对象被创建之后，在调用该对象中的方法时，不定义对象的引用变量，而直接调用这个对象的方法，这样的对象被称为匿名对象。

6．用 private 修饰的类成员被称为类的私有成员（Private Member）。私有成员无法从该类的外部访问，只能被该类自身访问和修改，而不能被任何其他类（包括该类的子类）获取或引用；如果在类的成员声明的前面加上修饰符 public，则该成员为公共成员，表示该成员可以被所有其他的类所访问。

7．重载是指在同一个类中定义多个具有相同名称的方法。对于这些同名的方法，其参数的个数不同或者参数的个数相同但类型不同，便可以具有不同的功能。

8．构造方法可以被视为一种特殊的方法，它的主要功能是帮助创建的对象赋初值。

9．构造方法有公共（Public）与私有（Private）之分，公共构造方法可以在程序的任何地方调用，所以新创建的对象均可自动调用它，而私有构造方法则无法在该构造方法所在的类以外的地方被调用。

10．如果一个类没有定义构造方法，则 Java 编译器会自动为其生成默认的构造方法。默认的构造方法中没有任何参数，方法体内也没有任何语句的构造方法。

11．实例变量与实例方法、静态变量与静态方法是不同的成员变量与成员方法。

12．基本类型的变量是指由 int、double 等关键字所声明而得到的变量，而由类声明而得到的变量称为类变量，属于引用类型变量的一种。

13．Java 语言具有垃圾自动回收的功能。

本章练习

一、思考题

1．类的构造方法的作用是什么？它有哪些特性？

2．什么是方法的重载？它有哪些特性？

3．静态方法与实例方法有哪些不同？

4．在一个静态方法内调用一个非静态成员为什么是非法的？

二、编程题

编写一个学生类（Student），包含以下内容。

属性：学号 no，姓名 name，性别 sex，年龄 age。

方法：构造方法，setter()方法和 getter()方法。

测试类 Test，包含以下内容。

主方法 main()，在其中创建并初始化两个学生对象 s1 和 s2，这两个对象的属性可以自行确定，分别显示这两个学生的学号、姓名、性别、年龄，修改 s1 的年龄并显示修改后的结果。

三、写出下列程序的运行结果

1.

```
public class Test1 {
   static String x = "1";
   static int y;
   public static void main(String args[]) {
      int z = 2;
      System.out.println(x + y + z);
      char ch1 = 'A', ch2 = 'w';
      if (ch1 + 2 < ch2)
         ++ch1;
      System.out.println(ch1);
   }
}
```

2.

```
public class Test2 {
    static int x = 1;
    int y;
    StaticTest() {
        y++;
    }
    static {
        x++;
    }
    public static void main(String[] args) {
       StaticTest st = new StaticTest();
       System.out.println("x=" + x);
       System.out.println("st.y=" + st.y);
       st = new StaticTest();
       System.out.println("st.y=" + st.y);
    }

}
```

3.

```
public class Test3{
```

```
  public static void swap(String s,char[] c,int i){
      s = "good";
      c[0] = 'd';
      i = 3;
  }
  public static void main(String args[]){
      String s = "abc";
      char[] c ={'a','b'};
      int   i = 8;
      swap(s, c, i);
      System.out.print(s);
      System.out.print(c[0]);
      System.out.print(i);
  }
}
```