第3章

抽象与封装

面向对象程序设计

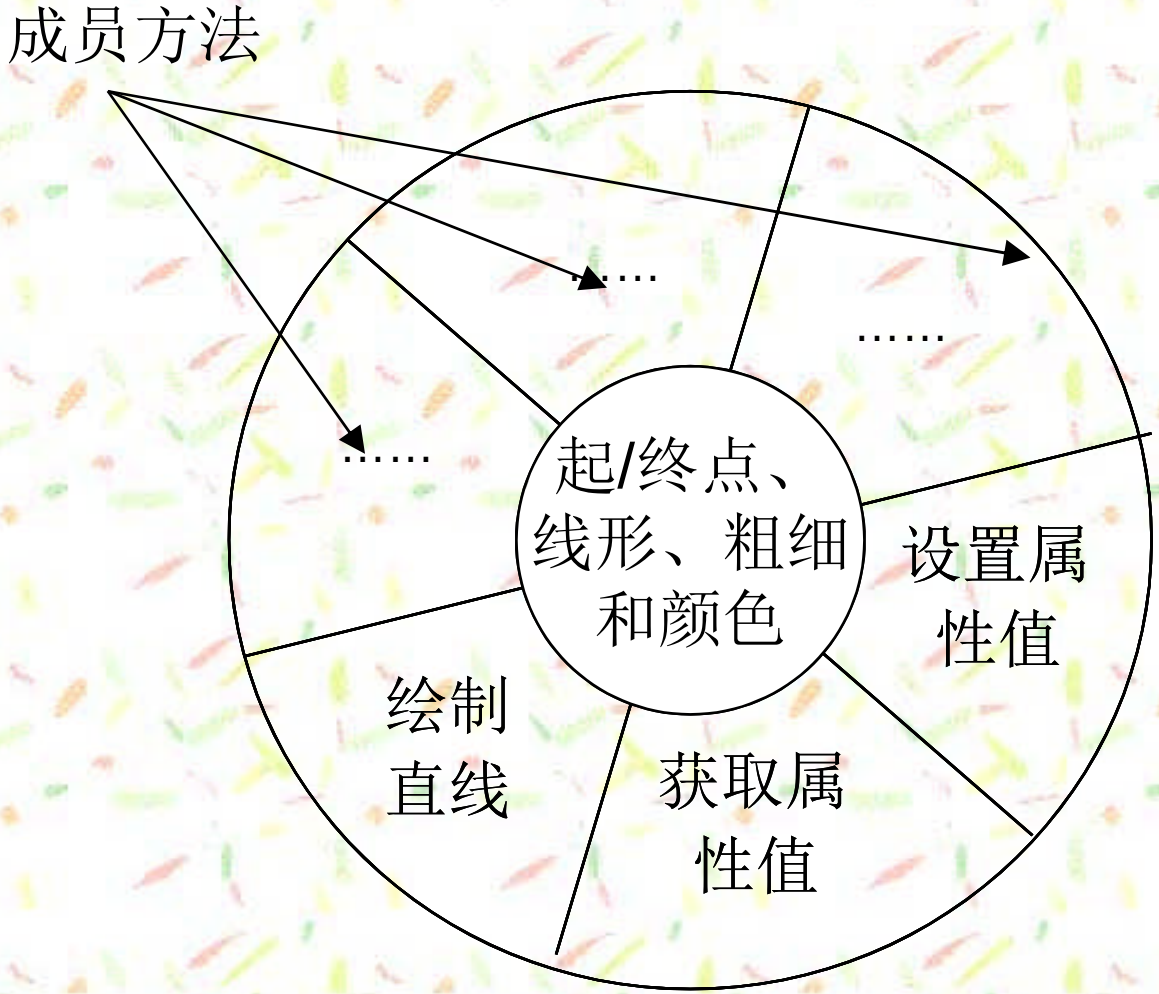
3.1 抽象与封装的实现技术

实现抽象和封装 ——类和对象

一个对象包含了若干个成员变量和成员方法，它是现实世界中特定实体在程序中的具体体现。其中，成员变量反映实体的属性状态，成员方法反映实体具有的行为能力，这些内容的规格描述将由类承担，类是对具有类似特征的对象的抽象说明，对象是类的实例。

面向对象程序设计

对象体的构成



面向对象程序设计

3.2 类

3.2.1 类的定义

Java语言中，类主要有两个来源途径

- Java类库
- 用户自定义的类

所有的类都是Object类的子类。如果在自定义类时，没有写明父类，则默认的父类为Object。从严格意义上讲,Java程序中定义的所有类都是子类。

最简单的类定义格式为

```
class  ClassName
{
      ClassBody        //类体
}
```

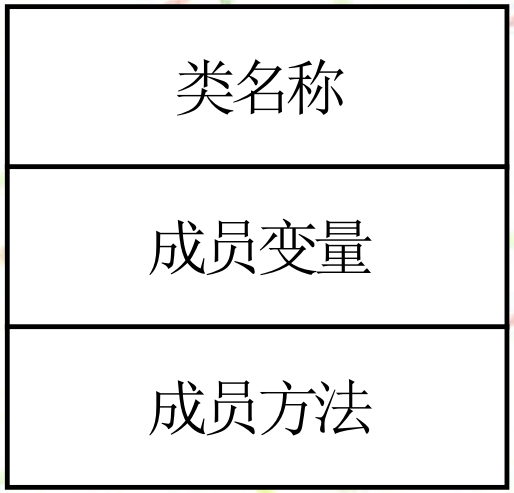
class为关键字，ClassName为定义的类名称

ClassBody为类体，包含成员变量、成员方法、类、接口、构造方法、静态初始化器

面向对象程序设计

类的UML表示

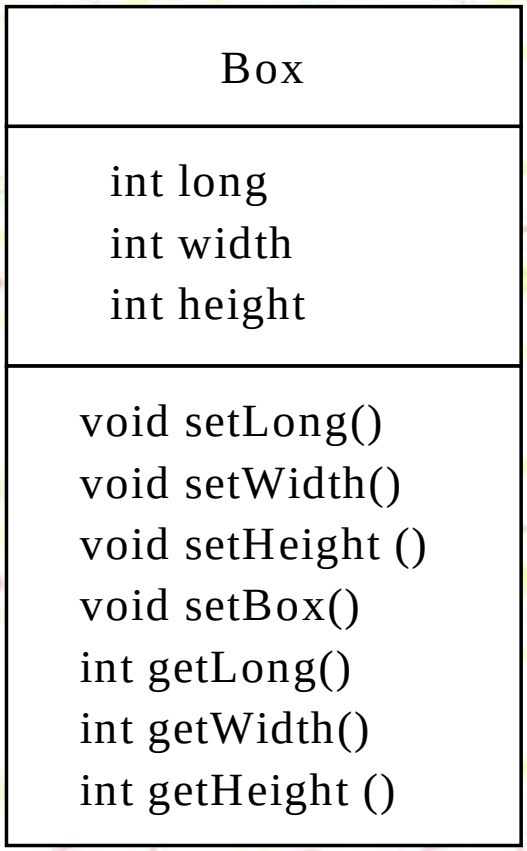
一个类的UML图形表示如图



面向对象程序设计

一个Box类的UML表示

BOX类的UML表示



Box类的定义

```
public class Box
{
  int long,width, height;
  void setLong(int longValue){long=longValue;}
  void setWidth(int widthValue){width=widthValue;}
  void setHeight (int heightValue){ height = heightValue;}
  void setBox(int longValue,int widthValue,int heightValue)
  {
     long=longValue;
     width=widthValue;
     height = heightValue;
  }
  int getLong(){return long;}
  int getWidth(){return width;}
  int getHeight (){return height;}
}
```

面向对象程序设计

成员变量和成员方法两种形式

实例变量和实例方法

每个变量和方法惟一地与一个对象相关联，即在创建某个对象时，同时为每个对象创建所有实例变量的副本，关联所有的实例方法

类变量和类方法（后续章节）

类的嵌套定义

```
class OutClass            //顶层类
{
   int conut;
   class InClass          //内部类
   {
       void printConut (){System.out.println("conut: "+(++ conut));}
    }
    void createInObject()
    {
       InClass  in=new InClass();      //引用内部类
       in.printConut ();
    }
 }
```

面向对象程序设计

类的嵌套定义

```
public class TestInClass        //用于测试内部类应用的类
{
  public static void main(String[] args)
  {
    OutClass outObj=new OutClass();   //创建外部类对象
    outObj.createInObject();   //调用创建内部类的成员方法
    OutClass.InClass inObj=outObj.new InClass(); //在外
                                部创建内部类对象
    inObj.printConut ();
  }
}
```

运行结果：

```
conut:1
conut:2
```

JAVA类的存储

有两种存储方式：

一种是将两个类定义存放在一个文件中

另一种是将两个类分别存放在两个不同的文件中

不管类定义的源代码是多个类存放在一个文件中，还是一个类存放在一个文件中，编译后都将一个类生成一个字节码文件，且文件名的前缀为类名，后缀为.class

面向对象程序设计

含有属于另外一个类的成员变量的典型例子

```
class Date
{
  int year,month,day;   //描述年、月、日的三个成员变量
  void setYear(int y){year=y;}
  void setMonth(int m){month=m;}
  void setDay(int d){day=d;}
  void setDate(int y,int m,int d)
  {
     year=y;
     month=m;
     day=d;
   }
   int getYear(){return year;}
   int getMonth(){return month;}
   int getDay(){return day;}
}
```

面向对象程序设计

Book类定义

```
class Book     //书籍类
{
   String name;
   String author;
   Date  publishDate;
   float  price;
   ……      //其他一些成员变量
   ……      //成员方法
}
```

Date是描述日期的类，Book是描述书籍的类。在Book类中，设一个表示出版日期的成员变量，它属于Date类。

Book类与Date类关系

可以将这两个类看成具有“整体-部分”的关系。即Book类是由Date类对象和一些其他类型的成员变量组合而成的，它们共同地反映了书籍信息。下面是这种关系的UML图形符号：

面向对象程序设计

3.2.2 成员变量的定义与初始化

实例变量的定义

```
Modifiers DataType  MemberName;
```

Modifiers： 修饰符,决定成员变量的存储方式和访问权限

DataType: 成员变量的类型

MemberName：是成员变量的名称

面向对象程序设计

初始化实例变量主要有5个途径

- 每个数据数据类型有默认的初始值
- 可以在定义的同时赋予相应的初值
- 在一个成员方法中,为每个实例变量赋值
- 在类的构造方法中实现初始化实例变量
- 利用初始化块对成员变量进行初始化

初始化块、构造方法初始化成员变量

```
class Point
{
   int x,y;
   {     //初始化块
      x=10;   y=20;
      System.out.println("Point initialization block");
   }
   Point(){};                //无参数的构造方法
   Point(int dx,int dy)        //带两个参数的构造方法
   {
      x=dx;y=dy;
      System.out.println("Point construct method");
   }
}
```

3.2.3 成员方法的定义

成员方法主要承担外部操作对象属性的接口任务。在一个类中，至少应该包含对类中的每个成员变量赋值，获取每个成员变量的当前值等功能的一系列成员方法。

成员方法也包含静态（static）和非静态两种形式，分别被称为类方法和实例方法。

实例方法

定义格式为：

```
Modifiers ResultType  MethodName(parameterList) [throws
        exceptions]
{
    MethodBody
}
```

Modifiers ： 修饰符

ResultType: 方法返回类型

MethodName: 方法名称

parameterList:参数列表

throws exceptions: 列出方法能够抛出的异常种类

面向对象程序设计

时间类Time的定义

```
public class Time            //时间类
{
   int hour, minute, second;      //时、分、秒
   void setTime(int h,int m,int s)        //设置时间
   {
        hour = (h<0)? 0: h%24;
       minute=(m<0):0;m%60;
       second=(s<0):0;s%60;
   }
   int getHour(){return hour;}          //返回时
   int getMinute(){return minute;}       //返回分
   int getSecond(){return second;}       //返回秒
}
```

面向对象程序设计

成员方法中处理的数据主要来源途径

- 传递给成员方法的参数
- 类中的成员变量，包括实例变量和类变量
- 在方法体内定义的局部变量
- 在方法中调用其他成员方法所得到的返回值

3.2.4 成员方法的重载

重载——

在一个类中，同一个名称的成员方法可以被定义多次的现象 。

一个成员方法重载的例子

在Time类，我们又增加了一个设置时间的成员方法，但这个成员方法的参数属于String类，时间将以“12:04:35”的格式传递给该成员方法。

```
class Time
{
   int hour, minute, second;
   void setTime(int h,int m,int s)     //参数为三个int变量
   {
      hour = (h<0)? 0: h%24;
      jinute=(m<0)?0:m%60;
      second=(s<0)?0:s%60;
   }
```

```
    void setTime(String time)       //参数为一个String类对象
    {
        hour= Integer.parseInt(time.substring(0,1));
        hour=hour<0?0:hour%24;
        minute= Integer.parseInt(time.substring(3,4));
        minute=minute<0?0:minute%60;
        second= Integer.parseInt(time.substring(6,7));
        second=second<0?0:second%60;
     }
    int getHour(){return hour;}
    int getMinute(){return minute;}
    int getSecond(){return second;}
} //end of class Time
```

面向对象程序设计

```
public class TestTime      //测试类
{
    public static void main(String args[])
    {
       Time t=new Time();   //创建Time对象
       t.setTime("13:04:20");//调用参数为String 的setTime()方法

      System.out.println(t.getHour()+":"+t.getMinute()+":"+
                                                    t.getSecond());
       t.setTime(20,30,38);  //调用参数为三个int的setTime()方法
      System.out.println(t.getHour()+":"+t.getMinute()+":"+
                                                    t.getSecond())
   }
}
```

将上面这两个类定义存储在一个文件名为TestTime.java中。经过编译
生成Time.class和TestTime.class

运行的结果为:

13:4:20

20:30:38

3.2.5 构造方法

构造方法——

在构造类对象时使用的一种特殊的成员方法，其主要作用是初始化成员变量。

构造方法的定义格式为：

[public] ClassName (parameterList)

public：控制访问权限的修饰符

ClassName：类名称

parameterList：参数表

简单的例子

```
class Point
{
   int x,y,;
   Point(int dx,int dy){x=dx;y=dy;}
   ……  //其他的成员方法
}
```

构造方法Point(int dx, int dy)仅对两个成员变量x和y赋予了初值。在利用new运算符创建Point类对象时，系统会自动地调用这个构造方法，实现对实例变量初始化，而不需要用户显式地调用它。

构造方法的重载

```
public class Time
{
   int hour, minute, second;
   Time(int h,int,m,int s){……}   //含有三个int类型参数的构造方法
   Time(long time) {……}          //含有一个long类型参数的构造方法
   Time(String time) {……}        //含有一个String类参数的构造方法
   void setTime(int h,int m,int s) {……}
   int getHour(){……}
   int getMinute(){……}
   int getSecond(){……}
}
```

在这个类中，有三个构造方法，它们的参数表均不相同。这样，就可以用三种不同形式的参数创建并初始化对象。

默认构造方法

如果在定义类的时候，没有定义任何构造方法，系统将提供一个参数表为空的默认构造方法。在这个默认的构造方法的方法体中，只有一个调用父类无参数构造方法的语句super()。

面向对象程序设计

例如：

```
public class Point
{
    int x,y;
    ……;  //其他的成员方法
}
```

等价于

```
public class Point
{
    int x,y;
    public Point(){ super(); }
    ……;  //其他的成员方法
}
```

3.3 对象

对象是对现实世界中实体进行抽象的结果，现实世界中的任何实体都可以映射成对象，而解决问题的过程就是对对象分析、处理的过程。从面向对象程序设计的观点看，程序是由有限个对象构成的，对象是程序操作的基本单位。所谓程序运行，就是对象之间不断地发送消息及响应消息的过程。

3.3.1 对象的创建

声明对象的语法格式为：

```
ClassName  objectName[,objectName];
```

例如：

```
Date dateObject;
Time timeObject1, timeObject2;
```

new运算符

new运算符主要完成的操作：

一为对象分配存储空间

二根据提供的参数格式调用与之匹配的构造方法

创建对象的格式为：

new ClassName (parameterList)

ClassName：对象所属的类名称

parameterList：创建对象时提供的参数

例如：**dateObject=new Date(2003,5,23,10,34,56)**

面向对象程序设计

3.3.2 对象成员的使用

对象创建后，就可以在对象之间相互发送及响应消息，以便驱动程序的运作。在Java语言中，所谓对象发送消息就是发出调用自身或其他类对象的成员方法的命令，所谓对象响应消息就是具体地执行上述的调用命令。

在对对象操作时，可能会引用成员变量或调用成员方法。下面是引用成员变量和调用成员方法的语法格式。

objectName.memberVariableName

objectName.memberMethodName(parameterList)

调用对象成员方法实际上就是向该对象发送请求操作的消息

面向对象程序设计

例 3.3.1　对象作为数组元素

```
class ScoreClass      //成绩类
{
   int No;         //学号
   int score;       //成绩
   ScoreClass()    //无参数的构造方法
   {
      No=1000;
      score=0;
   }
    ScoreClass(int n,int s)     //有两个参数的构造方法
    {
        No=n;
        score=s;
    }
    void setInfo(int n,int s)      //设置成绩
    {
        No=n;
        score=s;
    }
    int getNo(){return No;}          //获取学号
    int getScore(){return Score;}    //获取成绩
}
```

```
public class ScoreTest          //测试类
{
    public static void main(String[] args)
    {
        ScoreClass[] student=new ScoreClass[10];   //声明并创建一维数组
        for (int i=0;i<10;i++){
          student[i]=new ScoreClass(1000+i,(int)(Math.random()*100)); //创建对象
        }
        for (int i=0;i<10;i++){
          System.out.println(student[i].getNo()+"\t"+student[i].getScore());
        }
    }
}
```

可见，如果数组元素为对象，需要经过下面几个操作步骤：

首先要利用new运算符创建数组。此时，每个数组元素为null引用。

再利用循环结构，为每个数组元素创建对象。

通过引用成员变量或调用成员方法对数组中每个对象进行操作。

例 3.3.2 类中的成员变量为对象

```
class Point              //点坐标
{
   int x,y;              //x、y坐标
   Point(){ x=0;y=0;}
   Point(int dx,int dy){x=dx;y=dy;}
   void setXY(int dx,int dy){x=dx;y=dy;}     //设置x、y
   int getX(){return x;}              //返回x
   int getY(){return y;}              //返回y
   public String toString()   //将坐标信息转换成String形式
  { return "("+x+","+y+")" ;}
}
```

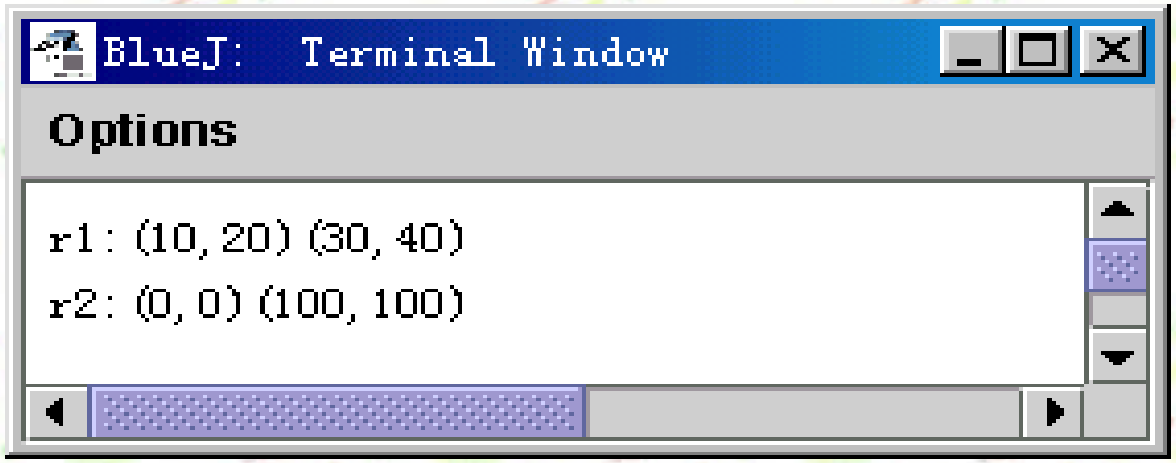
```
class Rect          //矩形类
{
   Point leftTop,rightBottom; //矩形左上角坐标,右下角坐标
   Rect()
   {
     leftTop=new Point();
     rightBottom=new Point();
   }
  Rect(int x1,int y1,int x2,int y2)
   {
     leftTop=new Point(x1,y1);
     rightBottom=new Point(x2,y2);
   }
```

面向对象程序设计

```
    Rect(Point lefttop,Point rightbottom)
    {
        leftTop=lefttop;
        rightBottom=rightbottom;
    }
    Point getLeftTop(){return leftTop;}      //返回左上角坐标
    Point getRightBottom(){return rightBottom;}    //返回右下角
       坐标
    public String toString()    //将矩形信息转换成String形式
    {
        return  leftTop.toString()+rightBottom.toString();
    }
} //end of class Rect
```

```
public class RectTest      //测试类
{
   public static void main(String[] args)
   {
      Rect r1,r2,r3;
      Point p1,p2;
      r1=new Rect(10,20,30,40);
      p1=new Point();
      p2=new Point(100,100);
      r2=new Rect(p1,p2);
      System.out.println("r1:"+r1.toString());
      System.out.println("r2:"+r2.toString());
   }
}
```

运行这个程序后，应该得到图所示的结果



通过这个例子，可以看出，对于含有成员对象的类，首先要创建每个成员对象，然后才能对其进行操作。建议最好将它们放置在构造方法中实现。

3.3.3 对象的清除

创建对象的主要任务是为对象分配存储空间，而清除对象的主要任务是回收存储空间。为了提高系统资源的利用率，Java语言提供了“自动回收垃圾”的机制。即在Java程序的运行过程中，系统会周期地监控对象是否还被使用，如果发现某个对象不再被使用，就自动地回收为其分配的存储空间。

3.4 访问属性控制

定义的每一个类和接口，以及其中的每一个成员变量和成员方法都存在是否可以被他人访问的访问属性。不同的访问属性，标识着不同的可访问性。所谓可访问性是一种在编译时确定的静态特性。在Java语言中，也正是利用访问属性机制实现数据隐藏，限制用户对不同包和类的访问权限。

4种访问属性

- 默认访问属性
- public（公有）访问属性
- private（私有）访问属性
- protected（保护）访问属性

面向对象程序设计

为类、接口、成员变量和成员方法指定访问属性的格式为：

[public|private|protected] ClassName;
[public|private|protected] memberVariableName;
[public|private|protected] emberMethodName(parameterList);

由于除默认访问属性外，指定其他三种访问属性需要写在定义之前，所以，又将它们称为修饰符。

3.4.1 默认访问属性

如果在定义类、接口、成员变量和成员方法时没有指定访问修饰符，它们的访问属性就为默认访问属性。具有默认访问属性的类、成员变量和成员方法，只能被本类和同一个包中的其他类、接口及成员方法引用。

3.4.2 public访问属性

拥有public访问属性的类、接口、成员变量、成员方法可以被本类和其他任何类及成员方法引用。它们既可以位于同一个包中，也可以位于不同包中。

例 3.4.1不同包之间的类访问

下面定义的三个类Point、Line和Test分别放在两个不同的包中。其定义如下：

```
package PointPackage;
//将Point和Line类放入PointPackage包
public class Point
{
  public int x,y;
  public void move(int dx,int dy){x+=dx;y+=dy;}
  //其他的成员方法
  ……;
}
```

```
class Line
{
    Point start,end;
    Line()
    {
        start=new Point();
        end=new Point();
    }
    public setLine(Point p1,Point p2){.....}
    //其他的成员方法
    ……;
}
```

将Point和Line类定义保存为Point.java文件。

```
package PointsUser;
//将Test类放入PointsUser包
import PointPackage.*;
//加载PointPackage包中的Point和Line类
public class Test
{
    public static void main(String[] args)
    {
        Point p=new Point();
        System.out.println(p.x+" "+p.y);
    }
}
```

将这个类定义保存为Test.java文件。

3.4.3 private访问属性

数据隐藏是面向对象的程序设计倡导的设计思想。将数据与其操作封装在一起，并将数据的组织隐藏起来，利用成员方法作为对外的操作接口，这样不但可以提高程序的安全性、可靠性，还有益于日后的维护、扩展和重用。将类定义中的数据成员设置为private访问属性是实现数据隐藏机制的最佳方式。

private访问属性可以应用于类中的成员，包括成员变量、成员方法和内部类或内部接口。具有private访问属性的成员只能被本类直接引用。

3.4.4 protected访问属性

具有protected访问属性的类成员可以被本类、本包和其他包中的子类访问。它的可访问性介于默认和public之间。如果你希望只对本包之外的子类开放，就应该将其指定为protected访问属性。

归纳上述4种不同的访问属性，可以将各种访问属性的可访问权限总结在表中：

	同一个类	同一个包	不同包中的子类	不同包中的非子类
private	*			
默认	*	*		
protected	*	*	*	
public	*	*	*	*

面向对象程序设计

3.5 静态成员

3.5.1 类变量的定义及初始化

在类中定义成员变量时，如果在访问属性修饰符之后，加上static修饰符，它们就属于静态成员。例如：

```
public static int staticMember;
```

静态成员变量只在加载类时创建一个副本，不管未来创建多少个这个类的对象，都将共享这一个副本 。

类变量的引用

由于类被加载后，就立即创建类变量，所以即使在该类没有创建一个对象的时候，类变量也已经存在。若在此时引用类变量，就只能用类名作为前缀。如果创建了该类的对象，则既可以通过类名引用类变量，也可以通过对象名引用类变量。

类变量不能在构造方法中初始化。因为构造方法只有在创建对象时才被调用，而类变量在没有创建对象之前就已经存在，并可以被引用；另外每创建一个对象，构造方法就要被调用一次，而类变量应该只被初始化一次。

类变量的初始化器

又称为静态变量初始化器来实现初始化。类变量初始化器应该位于类定义中，其语法格式为：

```
static{
  ......    // 类变量初始化
}
```

面向对象程序设计

构造方法与类变量初始化器

（1）构造方法用来初始化对象的实例变量，而类变量初始化器用来初始化类变量；

（2）构造方法在创建对象，即执行new运算时由系统自动地调用，而类变量初始化器是在加载类时被自动地执行；

（3）构造方法是一种特殊的成员方法，而类变量初始化器不是方法。

3.5.2 类方法

类方法也属于类。在类方法中，只能对该方法中的局部变量或类变量进行操作，而不能引用实例变量或调用实例方法。

如果希望将一个成员方法定义成类方法，就在访问属性修饰符之后，加上static修饰符即可。例如：

```
public static int getMember(){return staticMember;}
```

3.6 final、this和null修饰符

final

final修饰某个类：意味着这个类不能再作为父类派生其他的子类

final修饰某个方法：意味着这个方法不能被覆盖

final修饰成员变量：意味着这个成员变量在第一次被赋值后,不得被二次赋值

final修饰局部变量：在第一次被赋值后，也只能被引用，不能被二次赋值。

面向对象程序设计

this

this是对象自身的引用型系统变量。当在实例方法中引用本类的实例成员时，编译器会默认地在实例成员前加上this作为前缀。

例如构造方法Employee(String n,double s)这样编写

```
public Employee(String n,double s)
{
    name=n;
    salary=s;
    id=nextId;
}
```

面向对象程序设计

this

但实际上，编译器会将它改为：

```
public Employee(String n,double s)
{
    this.name=n;
    this.salary=s;
    this.id=nextId;
}
```

通常情况下，对实例成员的引用，编译器都会自动地帮助你加上this前缀。

若类的构造方法中的参数与类的实例变量命名为相同的名称，在成员方法中引用的变量名为局部变量，若想引用实例变量就需要显式地加上this。

面向对象程序设计

null

null是一个直接量，表示引用型变量为空的状态，通常用来作为引用型变量的初始值。

例如：**Employee e=null**

当希望一个对象型变量放弃它引用的对象时，也可以使用null实现。例如：

Employee e=new Employee(“Wnag”,2000.0);

若让e放弃它引用的对象，可以写成：e=null。系统会在适当的时候，自动地回收刚才e引用的对象所占据的存储空间。

3.7 对象拷贝

拷贝可以通过赋值语句来实现。

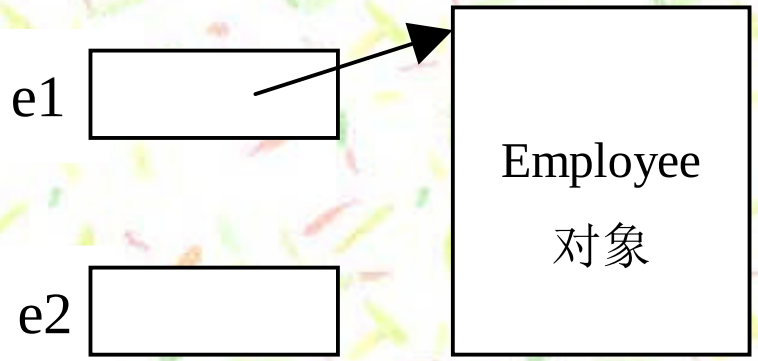
对于基本数据类型的变量，赋值操作的含义是将赋值号右侧表达式的计算结果赋给赋值号左侧的变量。

对于引用型变量，赋值操作实际上是引用的相互赋值。

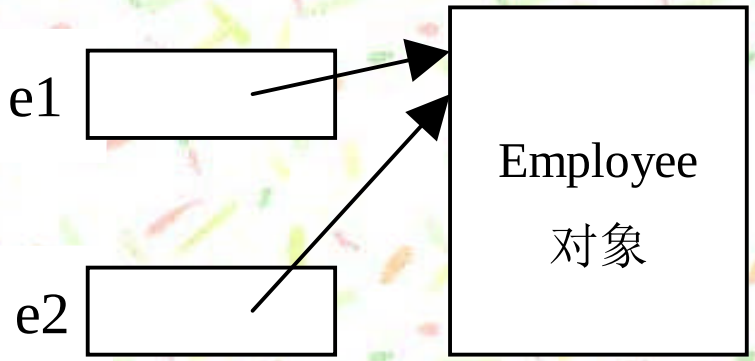
引用变量的赋值

```
Employee e1,e2;
e1=new Employee(“Zhang”,3000);
e2=e1;
```

其执行过程如图：



赋值前的状态



赋值后的状态

对象本身内容的拷贝

Java语言中，提供了一种创建对象拷贝的机制，称为克隆（cloning），它主要有两种方式：浅拷贝和深拷贝。

浅拷贝是指按照二进制位串进行对象拷贝，新创建的对象严格地复制原对象的值。

深拷贝是指对象的完全复制。如果对象的某个成员变量是其他对象的引用，也对所指的子对象依次进行复制。

Cloneable接口

Cloneable接口用来支持浅拷贝和深拷贝方式的对象克隆。

如果希望某个类对象具有克隆功能，就应该让这个类实现Cloneable接口，并对标准Object类的clone方法进行重载。随后就可以利用类似于下面的赋值语句进行对象克隆。

e2=(Employee)e1.clone();

面向对象程序设计

3.8 应用举例

在面向对象程序设计中，首要的问题是根据实际需求，设计出尽可能合理的类以及类与类之间的关系。这些类可以是直接来源于标准类库中的标准类；也可以将标准类作为基类，进一步构造更加能够表示特定问题的子类。类与类之间主要有“一般-特殊”、“整体-部分”及关联关系。