第5章 类 和 对 象

教学目标：

☑ 掌握类的构造方法的定义、作用，以及如何实现类的构造方法。

☑ 掌握如何创建类对象、如何使用类对象。

☑ 掌握静态成员和实例成员的使用方法，以及二者之间的区别。

☑ 掌握类成员的访问权限的设置方法以及使用原则。

教学重点：

本章首先要求读者掌握对象的定义和引用的方法，在此基础上深入理解静态成员和实例成员的定义及其应用、方法重载及其作用，类的封装和访问控制权限。

视频讲解

5.1 对象的定义和引用

从程序设计角度来看，可以把类看成一个数据类型，这种数据类型就是对象类型，简称为类。定义一个对象变量和定义基本数据类型变量的格式是一样的，假设以前述定义的 Cust 类为例，Cust 就是对象类型，就像 int 一样，现在来定义 Cust 对象类型变量，如有定义：“Cust　myCust;”，则 myCust 就是 Cust 类的对象变量。但对象类型和基本数据类型有着本质上的区别，声明对象变量之后，还不能使用对象，必须先用操作符 new 创建对象实体，之后才能使用对象。new 关键字的作用有以下几点：

（1）为对象分配内存空间。

（2）调用类的构造方法。

（3）为对象返回一个引用。

5.1.1 构造方法

构造方法是 Java 的一个重要概念。可以设想若每次创建一个类的实例都去初始化它的所有变量是很烦琐的。如果一个对象在被创建时就完成了所有的初始化工作，将是简单和简洁的。因此，Java 在类里提供了一个特殊的成员方法，叫做构造方法（Constructor）。构造方法必须以类名作为方法的名称，且不返回任何值，也就是说，构造方法是以类名为名称的特殊方法。

【例 5-1】 定义“银行账户”类的构造方法，实现对银行账户属性变量的赋值操作。

```
//文件名：Cust.java
Cust(String newName, int newID, String newPWD, int newMoney){
    name = newName;
    ID = newID;
    PWD = newPWD;
    money = newMoney;
}
```

构造方法的作用是确保对象在使用之前经过正确的初始化过程。当用户实例化一个对象时，此对象的构造方法被调用，它将初始化一个对象的内部状态。

知识提示 构造方法没有任何返回类型，即使是 void 类型也没有。

用 new 操作符创建一个实例后，就会得到一个可用的对象，然后用户才能在这个对象上执行其他操作。也就是说，对象在被实例化之前是不能被使用的。

在 Java 类中，最少要有一个构造方法。类的构造方法可以显式定义，也可以隐式定义。显式定义的意思是说在类中已经写好了构造方法的代码；隐式定义是指如果在一个类中没有定义构造方法，那么系统在解释时会分配一个默认的构造方法，这个构造方法只是一个空壳子，没有参数，也没有代码，类的所有属性系统将根据其数据类型赋默认值。所以说类的构造方法是必需的，但其代码可以不编写。总之，如果在类中已经实现了构造方法，系统不会分配构造方法；如果没有实现，系统就会分配。

系统使用默认的构造方法来初始化对象时，会将类中的变量自动初始化为该类型的默认值。例如，整型初始化为 0，浮点型初始化为 0.0，字符型初始化为'\u0000'，逻辑型初始化为 false，类对象初始化为 null 等。

在一个类中可以存在多个构造方法，这些构造方法都采用相同的名字。只是形式参数不同。Java 语言可根据构造对象时的参数个数及参数类型来判断调用哪个构造方法。

知识提示 构造方法是一个特殊的方法，其具有以下特点：

（1）构造方法必须与类同名。

（2）构造方法没有返回类型，也不能定义为 void。

（3）构造方法的主要作用是完成对象的初始化工作，它能够把定义对象时的参数传给对象的成员。

（4）构造方法不能由编程人员调用，而由系统调用。

（5）一个类可以定义多个构造方法，即构造方法可以重载。如果没有定义构造方法，则编译系统会自动插入一个无参数的默认构造方法，这个默认构造方法没有任何代码。

5.1.2 对象的创建

为了在程序中使用对象，首先要声明和创建一个对象，然后给它发送消息，即调用它的方法。

前面定义的类 Cust 仅提供该类的类型定义，定义好类之后，可以在声明中将其用作一种类型使用。例如：

```
Cust  myCust;
```

其中，类名 Cust 是一个类型名称，myCust 为该类型的变量，我们称之为对象变量或对象引用。但执行这条语句时，系统没有调用任何构造方法，这是因为上述的语句并没有生成对象，生成的只是变量 myCust，其默认值为 null。这个变量可以用来存储一个已经生成的 Cust 对象的引用，即 Cust 实例对象的地址。生成对象可以使用 new 操作符，用其初始化对象。例如：

```
myCust = new Cust("Tom",100,"12345",10000);
```

下面两种创建对象的方法是等效的。

```
Cust   myCust ;
```

```
myCust = new Cust("Tom",100,"12345",10000);
```

等效于：

```
Cust  myCust = new Cust("Tom",100,"12345",10000);
```

当系统执行上述语句时，new 操作符将创建一个 Cust 类型的对象，即给这个 Cust 对象分配空间，并调用类 Cust 的构造方法，将参数传给对象的变量 name、ID、PWD 和 money，然后将此对象的内存首地址返回给 myCust，这样就创建了对象 myCust。所以创建一个类的对象时，对象名本身代表的实际上是一个引用地址，通过它可以操作管理对象所拥有的内容。对象的内存模型和对象的赋值过程如图 5-1 所示。

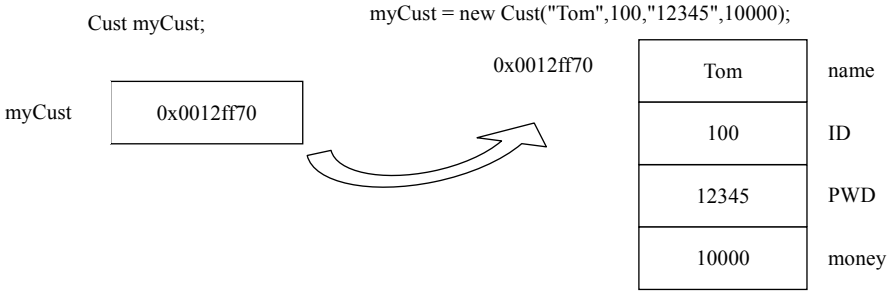


图 5-1　对象的内存模型和对象的赋值示意图

注意：对象变量声明与对象创建的不同，例如：

```
Cust  myCust;
```

此语句执行后，系统并未创建 Cust 对象，而只是分配了一个能存放 Cust 对象地址（存储位置）的变量 myCust。只有使用 new 操作符创建一个对象后，JVM 才会给该对象分配空间，将该地址空间的起始地址写入对象变量。引用类型变量没有值时，我们称之为“空引用”，当变量 myCust 存储了一个对象的地址时，我们说“变量指向一个对象”或“变量引用一个对象”。类、引用变量和对象之间的关系如图 5-2 所示。

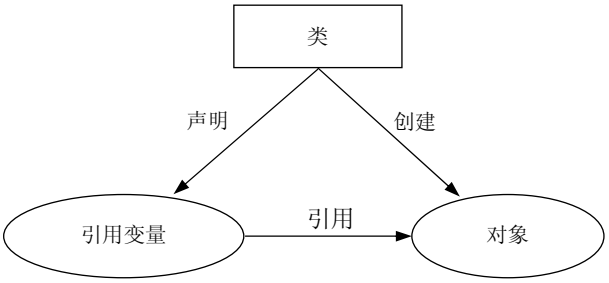


图 5-2　类、引用和对象的关系

类作为同一类对象的模板，使用 new 操作符及其后面跟随的构造方法的调用，可以生成多个不同的对象，这些对象将被分配不同的内存空间。因此，尽管这些对象的变量可能有相同的值，但内存地址不同，是不同的对象。

如果一个对象变量没有被初始化，则该对象变量为 null，表示不指向任何内存地址的引

用。如果一个对象变量赋给另一个对象变量，则两个对象变量的引用地址相同，它们实际指向同一个对象，只是引用对象的变量具有不同的名称。

5.1.3 对象的使用

当创建了一个对象之后，这个对象就拥有所属类的成员变量（实例变量）和方法，就可以引用该对象的成员变量，调用其成员方法。

在类的作用域中，一个类中声明的成员变量和方法可由类中的所有方法访问，并可用它的名称进行引用。

在类的作用域外部，类中声明的成员变量和方法的存取必须通过该类或该类的对象和点操作符（称为类成员存取操作符）来存取，在类的外部调用成员变量和成员方法的语法格式如下：

```
对象名.成员变量名;
对象名.方法名(形参列表);
```

例如：

```
Cust  myCust = new Cust("Tom",100,"11111",10000);
myCust.setMoney(5000);
```

知识提示 在调用方法时，即使没有参数，圆括号也不能省略。如果有参数，那么每个参数必须有确定的值。方法名称后括号中的参数称为“实际参数”，简称“实参”。

5.1.4 对象的销毁

通过 new 操作符实例化对象时，系统为对象分配所需的存储空间，存放其成员属性的值。但内存空间是有限的，不能存放无限多的对象。为此，Java 提供了资源回收机制，自动销毁无用的对象，收回其所占用的空间。一般情况下，用户不需要专门设计释放对象的方法。

但如果需要主动释放对象，或在释放对象时需要执行特定的操作，例如，编程者的类对象使用的资源是操作系统提供的一些图形、字体等非 Java 语言环境的资源时，则在类中可以定义 finalize()方法。当系统销毁对象时，将自动执行 finalize()方法。对象也可以调用 finalize()方法来销毁自己。

finalize()方法没有参数，也没有返回值。一个类只能有一个 finalize()方法，其方法定义形式如下：

```
protected  void  finalize() {
    方法体;
}
```

5.2 案例分析：银行账户对象的创建

视频讲解

1. 案例描述

建立一个银行账户类，要求能够存放用户的账号、户名、密码和账户余额等个人信息，并包含存款、取款、查询余额和修改账户密码等操作，并用此类创建对象，对象的账号为 100，户名为 Tom，密码为 11111，账户余额为 10000。

2. 案例分析

建立银行账户类 Cust 之后，通过 new 操作符调用类的构造方法，将各项成员属性信息传给对象，并在主方法中调用类的成员方法。

3. 案例实现

本例的代码如下：

```
//文件名：MainDemo.java
class Cust {
    String name;
    int ID;
    String PWD;
    int money;

    Cust(String newName,int newID,String newPWD,int newMoney){
        name = newName;
        ID = newID;
        PWD = newPWD;
        money = newMoney;
    }
    void getMoney(int getMoney){
        money = money - getMoney;
    }
    void setMoney(int saveMoney){
        money = money + saveMoney;
    }
    void search(){
        System.out.println("户名："+name);
        System.out.println("账号："+ID);
        System.out.println("账户余额："+money);
    }
    void changePWD(String newPWD){
        PWD = newPWD;
    }
}

public  class  MainDemo{
    public static void main(String[] args){
        Cust myCust = new Cust("Tom",100,"11111",10000);
        myCust.setMoney(5000);
        myCust.getMoney(3000);
        myCust.changePWD("Tom");
        myCust.search();
    }
}
```

4. 归纳与提高

对于创建的类，主要是通过创建该类的对象来访问其成员属性或成员方法。创建对象主要通过 new 操作符完成，当执行 new 操作时，系统会根据类名后的参数类型及个数来确定所调用的构造方法。

5.3 静态成员与实例成员

Java 的类中可以包含两种成员：实例成员和静态成员。

简单来说，类是一种类型而不是具体的对象，一般在类中定义的成员是每个由此类产生的对象都拥有的，因此可以称之为实例成员或对象成员。只有创建了对象之后，才能通过对

象访问实例成员变量、调用实例成员方法。

如果需要让类的所有对象在类的范围内共享某个成员，而这个成员不属于任何由此类产生的对象，那么它是属于整个类的，这种成员称为静态成员或类成员。例如，Math 类中的 pow()方法就是一个静态方法。静态方法不向对象施加任何操作。我们用 Math.pow(x,y)调用 Math 类的 pow 方法计算幂 x^y，它并不使用任何一个 Math 对象来执行此任务，也没对任何对象施加操作。

5.3.1 静态属性与实例属性

用 static 修饰符声明的成员属性为静态属性。一个静态属性只标识一个存储位置。无论创建了多少个类实例，静态属性永远都在同一个存储位置存放其值，静态属性是被共享的。因此，当某个对象修改了静态属性的值之后，所有对象都将使用修改的静态属性值。

没有 static 修饰的属性为实例属性。每个对象分别包含一组该类的所有实例属性。创建类的对象时，都会为该对象的属性创建新的存储位置，也就是说，类的每个对象的实例属性的存储位置是不相同的。因此修改一个对象的实例属性值，对另一个对象的该实例属性的值没有影响。

例如，在前述的 Cust 类中，增加类的成员属性 bankName，表示账户所属的银行，由于所有账户都是同一个银行，它不属于某个单独账户，因此，应将其定义为静态属性，还可以给每个 Cust 类添加一个流水编号和账户总编号，并将账户总编号定义为静态属性。当创建一个 Cust 对象时，按创建 Cust 对象的顺序自动给该对象的账户编号设置一个值。

```
class Cust {
    …
    static String bankName="中国农业银行";
    int selfNum=0;
    static int allNum=0;
    Cust(String newName,int newID,String newPWD,int newMoney)  {
        …
        allNum++;
        selfNum=allNum;
    }
    …
}
```

5.3.2 静态方法与实例方法

静态方法是不向调用它的对象施加操作的方法，因为静态方法并不操作调用它的对象，所以不能用静态方法来访问实例属性，使用“类名.方法名”来调用静态方法，尽管如此，我们通过对象来调用静态方法是合法的，不会发生错误。由于静态方法计算或操作的结果和调用它的任何对象都没有关系，所以用对象调用静态方法很容易让人迷惑。因此要求使用“类名.方法名”来调用静态方法。在以下两种情况下使用静态方法：

（1）该方法不需要访问对象的状态，其所需的参数都通过类的显示参数提供（如 Math.pow()方法）。

（2）该方法只需要访问类的静态属性。

5.3.3 静态成员与实例成员的特征

类的成员或者是静态成员，或者是实例成员。一般来说，将静态成员看作属于类，而将

实例成员看作属于对象（类的实例）。

1. 静态成员的特征

当成员变量或成员方法包含 static 修饰符时，即被声明为静态成员。静态成员具有下列特征：

（1）当以 E.M 形式引用静态成员时，E 必须是成员 M 的类型。

（2）一个静态属性只标识一个存储位置。无论创建了多少个类的实例，永远都只有静态属性的一个副本。

（3）静态方法不在某个特定对象上操作，在这样的方法中引用 this 是错误的。

2. 实例成员的特征

当成员属性、成员方法、构造方法的声明中不包含 static 修饰符时，声明为实例成员，实例成员也可称为非静态成员。实例成员具有下列特征：

（1）当以 E.M 形式引用实例成员时，E 是成员 M 的类型的对象。

（2）类的每个对象分别包含一组该类的所有实例属性。类的每个对象都为每个实例属性建立一个副本。也就是说，类的每个对象的实例属性的存储位置是不相同的。

（3）实例方法在类的给定对象上操作，此对象可以作为 this 访问。

知识提示 因为静态方法并不操作对象，所以不能用静态方法来访问实例属性，但静态方法可以访问自身类的静态属性。静态成员与实例成员归纳如下：

（1）静态方法可以访问静态成员变量，不可以访问实例成员变量。

（2）实例方法可以访问静态成员变量，也可以访问实例成员变量。

为了进一步明确静态方法与实例方法的区别，现举例说明。

【例 5-2】 静态方法与实例方法的区别。

```
//文件名：StaticDemo.java
public  class  StaticDemo {
    static double pi=3.14;          //静态变量,类变量
    double pix=3.14;                //实例变量,对象变量
    double  getArea(){              //实例方法
        return pi*3*3;              //类变量,实例方法能用类变量
    }
    static double getArea1(){
        return pi*3*3;              //类方法能用类变量
    }
    double  getArea2(){
        return pix*3*3;             //实例方法能用实例变量
    }
    //static double getArea3(){
    //return pix*3*3;               //类方法不能用实例变量
    //}
}
```

【例 5-3】 静态成员的加法运算。

```
//文件名：HasStatic.java
public  class  HasStatic{
    private static int x = 100;
    public static void main(String args[]){
        HasStatic hs1 = new HasStatic();
        HasStatic hs2 = new HasStatic();
        hs1.x++;
```

```
            hs2.x++;
            hs1.x++;
            System.out.println("x="+x);
        }
    }
```


程序的运行结果如下：

```
x=103
```

视频讲解

5.3.4 关键字 this 的使用

每个对象都有一个名为 this 的引用，它指向当前对象本身，主要有如下 4 个方面的应用：

（1）this 调用本类中的属性，也就是类中的成员变量。

通过 this 引用成员变量的格式如下：

```
this.成员变量名;
```

当成员方法中没有与成员变量同名的参数时，this 可以省略。但当成员方法中存在与成员变量同名的参数时，引用成员变量其名前的 this 不能省略，因为成员方法中默认的是引用方法中的参数，而不是成员变量。

例如，在前述的方法 setMoney 中，根据其参数名字的不同，可以有两种写法。

```
//方法一：
void setMoney(int saveMoney)    //成员方法中变量名与成员变量名不同
{
    money = money + saveMoney;
}
//方法二：
void setMoney(int money)        //成员方法中变量名与成员变量名相同
{
    this.money = this.money + money;
}
```

（2）this 调用本类中的其他方法。

通过 this 调用成员方法的格式如下：

```
this.成员方法名（参数表）;
```

其中，成员方法名前的 this 可以省略。

（3）this 调用本类中的其他构造方法。

在构造方法中，可以通过 this 调用本类中具有不同参数表的构造方法，其调用的格式如下：

```
this（参数表）;
```

【例 5-4】 为 Cust 类编写两个构造方法：一个构造方法为无参构造方法，利用 this 关键字调用有参构造方法；另一个为有参构造方法，参数分别为户名、账号、密码及余额。

```
Cust(){
    this("Tom",100,"12345",10000);
}
Cust(String newName,int newID,String newPWD,int newMoney){
    name = newName;
    ID = newID;
    PWD = newPWD;
```

```
        money = newMoney;
    }
```

在构造方法中使 this 关键字表示调用类中的构造方法。Java 编译器会根据所传递的参数数量的不同，来判断该调用哪个构造方法。

不过如果要使用这种方式来调用构造方法，有一个语法上的限制。一般来说，利用 this 关键字来调用构造方法，只能在无参数构造方法的第一句使用 this 调用有参数的构造方法；否则编译的时候就会提示错误信息。

（4）返回对象的值。

在代码中，可以使用 return this 返回某个类的引用。此时这个 this 关键字就代表类的名称。如在上面这个 Cust 类中，return this 的含义就是 return Cust。可见，this 关键字除了可以引用变量或者成员方法之外，还可以作为类的返回值。

5.4 方法的重载

5.4.1 成员方法的重载

每一成员方法都有其签名，方法的签名由方法的名称及它的形参的数量、每个形参的类型组成。具体来说，方法签名不包含返回类型。

在类中如果声明有多个同名的方法但它们的签名不同，则称为方法的重载。当方法对不同数据类型进行操作时，方法的重载非常有用，因为方法的重载提供了对可用数据类型的选择，从而使方法的使用更为容易。

5.4.2 构造方法的重载

类定义中含有两个以上参数个数或类型不同的构造方法时，称为构造方法重载。构造方法实际上是对对象进行实例化时调用的方法。如希望创建一个可以以多种方式构造对象的类，就需要重载构造方法。

和一般的方法重载一样，重载的构造方法具有不同个数或不同类型的参数，编译器就可以根据这一点判断出用 new 关键字产生对象时，该调用哪个构造方法。

【例 5-5】 重载 Cust 类的构造方法：一个为无参构造方法，实现成员变量的初始化；另一个为有参构造方法，实现形参到成员变量的赋值。

```
Cust(){
    name = "";
    ID = 0;
    PWD = "";
    money = 0;
}
Cust(String newName,int newID,String newPWD,int newMoney){
    name = newName;
    ID = newID;
    PWD = newPWD;
    money = newMoney;
}
```

在创建对象时，可以根据需要用不同的方式创建对象，对对象完成不同的初始化操作，例如执行如下的两条语句：

```
Cust st1 = new Cust("Tom",100,"11111",10000);
Cust st2 = new Cust();
```

此时，系统将创建对象 st1、st2，但对象 st1 是通过参数对其成员属性进行了初始化，而对象 st2 则通过无参构造方法的调用，将其成员属性初始化为各类型的默认值。

5.5 案例分析：银行账户类构造方法的重载

1. 案例描述

建立一个银行账户类，要求能够存放用户的账号、户名、密码、账户余额、账号流水号等个人信息，以及所在银行名称、总账户数等公共信息，功能方面要包含存款、取款、查询余额、修改账户密码等操作，要求采用不同的构造方法来构造实例对象。

2. 案例分析

根据案例描述中的信息，分别创建类的成员属性和成员方法，对构造方法进行重载。对此案例可以设计一个无参构造方法，以对创建的银行账户对象的各个成员属性初始化；另一个为有参构造方法，以实现将参数赋值给对象成员属性。

3. 案例实现

本例的代码如下：

```
//文件名：CustDemo1.java
class Cust {
    String name;
    int ID;
    String PWD;
    int money;
    static String bankName="中国农业银行";
    int selfNum=0;
    static int allNum=0;

    Cust(){
        name = "";
        ID = 0;
        PWD = "";
        money = 0;
        allNum++;
        selfNum=allNum;
    }
    Cust(String newName,int newID,String newPWD,int newMoney){
        name = newName;
        ID = newID;
        PWD = newPWD;
        money = newMoney;
        allNum++;
        selfNum=allNum;
    }
    void getMoney(int getMoney){
        money = money - getMoney;
    }
    void setMoney(int saveMoney){
        money = money + saveMoney;
    }
    void search(){
        System.out.println("所属银行:"+Cust.bankName);
        System.out.println("您是本银行的"+allNum+"个顾客中的第"+
```

```
                selfNum+"个顾客");
        System.out.println("户名: "+name);
        System.out.println("账号: "+ID);
        System.out.println("账户余额: "+money);
    }
    void changePWD(String newPWD){
        PWD = newPWD;
    }
    void setInfo(String newName,int newID,String newPWD,int newMoney){
        name = newName;
        ID = newID;
        PWD = newPWD;
        money = newMoney;
    }
}
public  class  CustDemo1 {
    public static void main(String[] args){
        Cust st1 = new Cust("Tom",100,"11111",10000);
        Cust st2 = new Cust();

        st1.setMoney(5000);
        st1.getMoney(3000);
        st1.changePWD("Tom");
        st1.search();

        st2.setInfo("Jerry", 200, "22222", 10000);
        st2.setMoney(10000);
        st2.getMoney(5000);
        st2.changePWD("Jerry");
        st2.search();
    }
}
```

4. 归纳与提高

在 Java 中，每个类至少需要一个构造方法（可以有多个），它用于构造类的对象。在 Java 中构造方法必须与类名相同。构造方法可以不带有参数，也可以带有参数。不带有参数的构造方法被称为无参构造方法。如果我们不给类提供构造方法，那么编译器会自动提供一个无参构造方法。换句话说，一个类至少有一个构造方法，而且默认程序员可以不写的构造方法，但是最好的习惯是加上默认构造方法。

当构造方法的形参和成员变量同名时，成员变量一定要加上 this 强调当前对象，如果没有，并没有对成员变量赋值，只是形参的赋值运算而已，输出的成员变量也只是系统赋予的默认值为 0，所以构造方法中的初始化都要加上 this 强调是当前对象。

5.6 类的封装和访问控制

5.6.1 类的封装

封装性是面向对象的核心特征之一，它提供了一种信息隐藏技术。类的封装性的含义是将数据和对数据的操作组合起来构成类，类是一个不可分割的独立单位。类中既要提供与外部联系的接口，同时又要尽可能隐藏类的实现细节。封装性为软件提供了一种模块化的设计机制，设计者提供标准化的类模块，使用者根据实际需求来选择所需要的模块，通过组装模块实现大型软件系统。

类的设计者和使用者考虑问题的角度不同，设计者需要考虑如何定义类中的成员变量和方法，如何设置其访问权限等问题；而类的使用者只需要知道有哪些类可以选择，每个类有哪些功能，每个类中有哪些可以访问的成员变量和成员方法等，而不需要了解其实现细节。

5.6.2 访问控制

按照类的封装性原则，类的设计者既要提供类与外部的联系方式，又要尽可能地隐藏类的实现细节。这就要求设计者应根据实际需要，为类和类中的成员变量和成员方法分别设置合理的访问权限。

Java 为类中的成员变量和成员方法设置了 4 种访问权限，为类本身设置了 2 种访问权限。

1. 类成员的访问权限

Java 提供的 4 种访问权限分别为：public（公有）、protected（保护）、默认和 private（私有）。具体含义如下：

1）public

被 public 修饰的成员变量和成员方法可以在所有的类中被访问。所谓在某类中访问某成员变量，是指在该类的方法中给该成员变量赋值、输出其值、在表达式中应用其值等；所谓在某类中访问成员方法，是指在该类的方法中调用该成员方法。因此在所有类的方法中都可以使用被 public 修饰的成员变量，调用被 public 修饰的成员方法。

2）protected

被 protected 修饰的成员变量和成员方法可以在声明它们的类中被访问，或在该类的子类中被访问，也可以被与该类位于同一个包中的类访问，但不能被其他包中的非子类访问（子类和包的知识详见第 6 章）。

3）默认

默认是指不使用任何权限修饰符。被默认修饰的成员变量和成员方法可以被声明它们的类访问，也可以被与该类在同一包中的类访问，但不能被位于其他包中的类访问。默认权限以包为界划分访问权限的范围，使成员可以被与其所属的类位于同一包中的类访问，而不能被该包之外的类访问。

4）private

被 private 修饰的成员变量和成员方法只能在声明它们的类中被访问，而不能被其他类，甚至其子类所访问。被 private 修饰的成员，其被访问的权限范围最小，对所有其他类都隐藏信息。

对 4 种访问权限修饰符的总结如表 5-1 所示。

表 5-1 访问权限

比较项目	public	protected	默认	private
本类	√	√	√	√
本类所在包	√	√	√	—
其他包中的子类	√	√	—	—
其他包中的非子类	√	—	—	—

不能用访问权限修饰符修饰成员方法中声明的变量或形式参数，因为方法中声明的变量或形式参数的作用域仅限于该方法，在方法之外是不可见的，在其他类中更无法访问。

2. 类的访问权限

声明一个类，可以使用的权限修饰符只有 public 和默认两种，不能使用 protected 或 private。

【例 5-6】 创建不同的包，并在包内创建不同的类，实现不同包间类的访问。

在 MyEclipse 某一工程中创建如图 5-3 所示的目录结构，这里主要创建 lession3 包及 lession3.otherpackage 包，用来存放非同一目录的类。

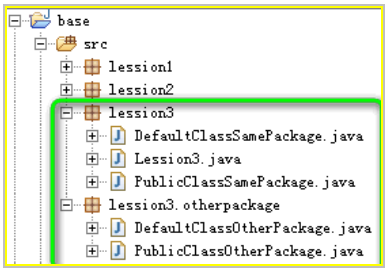


图 5-3　工程目录结构示意图

在 lession3 包中的主文件 Lession3.java 代码如下：

```
//文件名：Lession3.java
package lession3;
//注意，我们使用了其他package的类,所以需要import进来
//注意：路径必须为完整的路径
import lession3.otherpackage.PublicClassOtherPackage;

//public的类，可以被任何类在任何地方访问
//默认权限的类(即没写任何访问描述符的类)只能在当前package访问，不能被其他package的类
//访问
public  class  Lession3 {
    public static void main(String[] args) {
        //调用同一package下的public访问权限类
        System.out.println (new PublicClassSamePackage().toString());
        //调用同一package下的默认访问权限类
        System.out.println (new DefaultClassSamePackage().toString());
        //调用不同package下的public访问权限类
        System.out.println (new PublicClassOtherPackage().toString());
        //调用不同package下的默认访问权限类
        //System.out.println(new DefaultClassOtherPackage().toString());
    }
}
```

在 lession3 包中 PublicClassSamePackage.java 代码如下：

```
//文件名：PublicClassSamePackage.java
package lession3;
/**
* 相同package的公开访问权限类
*/
public  class  PublicClassSamePackage {
    public String toString() {
        return "相同package的公开类";
    }
}
```

在 lession3 包中 DefaultClassSamePackage.java 代码如下：

```
//文件名：DefaultClassSamePackage.java
package lession3;
/**
* 相同package的默认访问权限类
*/
public  class  DefaultClassSamePackage {
    public String toString() {
        return "相同package的默认类";
    }
}
```

在 lession3.otherpackage 包中 PublicClassOtherPackage.java 代码如下：

```
//文件名：PublicClassOtherPackage.java
package lession3.otherpackage;
/**
* 不同package的公开访问权限类
*/
public  class  PublicClassOtherPackage {
    public String toString() {
        return "不同package的公开类";
    }
}
```

在 lession3.otherpackage 包中 DefaultClassOtherPackage.java 代码如下：

```
//文件名：DefaultClassOtherPackage.java
package lession3.otherpackage;
/**
* 不同package的默认访问权限类
*/
public  class  DefaultClassOtherPackage {
    public String toString() {
        return " 不同package的默认类";
    }
}
```

程序的运行结果如下：

```
相同package的公开类
相同package的默认类
不同package的公开类
```

调用不同包下的默认访问权限类，如果把 Lession3.java 中的被注释掉的代码恢复，则会产生编译错误，如图 5-4 所示。

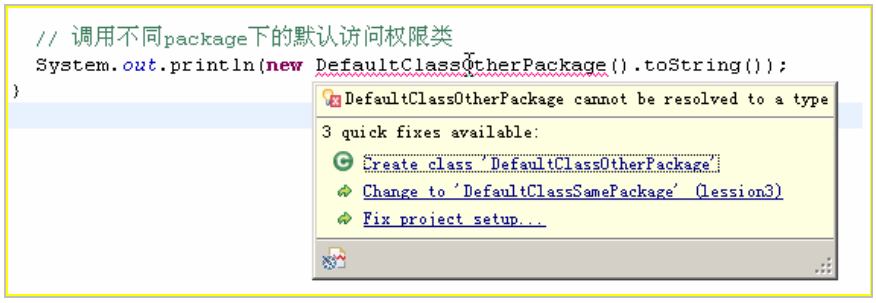


图 5-4　编译错误示意图

可见，一个类如果想直接访问另一个单独的类，有两种情况：

（1）和自己在一个包中，无论其是否为 public。

（2）和自己不在一个包中，且必须为 public。

知识提示 在一个 Java 源程序文件中，可以包含多个类，但只能有一个类使用 public 修饰符，该类的名字必须与源程序文件的名字相同。另外，当程序中创建多个类时，必须运行包含 main()方法的类，否则将会出错。

5.7 案例分析：简单的银行账户管理程序

1. 案例描述

建立一个银行账户类，要求能够存放用户的账号、户名、密码、账户余额、账号的流水号等个人信息，以及所在银行名称、总账户数等公共信息，功能方面要包含存款、取款、查询余额、修改账户密码等操作，要求采用键盘输入数据的方式，并用不同的构造方法来构造实例对象。

2. 案例分析

根据案例描述中的信息，可以设计 Cust 类的无参构造方法和有参构造方法，在构造方法中主要实现各个成员变量的赋值运算、账户流水号及总账户数的计算。Cust 类中还要实现存款方法 saveMoney()、取款方法 getMondy()、查询方法 search()、修改密码方法 changePWD()、获取账号方法 getID()。在主类 Demo 中，主方法 main()要实现各操作的选择菜单，实现基本的账户管理操作。

此外，还可以设计一个 KB 类，主要实现从键盘输入数据。在 KB 类中有 scan()方法，利用 Java 的输入流实现数据输入。在 scan()方法中用到的 System.in 是 java.lang 包中 System 类预定义的属性，是 InputStream 抽象类的一个实例变量。InputStream 类是表示字节输入流的所有类的超类，InputStreamReader 是字节流通向字符流的桥梁，它使用指定的字符集读取字节并将其解码为字符。BufferedReader 类用于从缓冲区中读取内容，所有的输入字节数据都将放在缓冲区中以提高读取效率。关于输入输出流的具体内容将在第 8 章中给出详细介绍。

3. 案例实现

本例的代码如下：

```
//文件名：CustDemo2.java
import java.io.BufferedReader;
import java.io.InputStreamReader;

class KB{
    public static String scan(){
        String str = "";
        try {
            BufferedReader buf = new BufferedReader(new InputStreamReader
            (System.in));
            str = buf.readLine();
        }
        catch (Exception e){  }
            return str;
    }
}
```

```
class Cust{
    private String name;
    private int ID;
    private String PWD;
    private int money;
    static String bankName="中国农业银行";
    private int selfNum=0;
    static int allNum=0;

    Cust(String newName,int newID,String newPWD,int newMoney){
        name = newName;
        ID = newID;
        PWD = newPWD;
        money = newMoney;
        allNum++;
        selfNum=allNum;
    }
    void getMoney(){
        System.out.print("请输入要取出的金额：");
        int n = Integer.parseInt(KB.scan());
        money = money-n;
    }
    void saveMoney(){
        System.out.print("请输入要储蓄的金额：");
        int n = Integer.parseInt(KB.scan());
        money = money+n;
    }
    void search(){
        System.out.println("所属银行:"+Cust.bankName);
        System.out.println("您是本银行"+allNum+"个顾客中的第"+selfNum+"个顾客");
        System.out.println("户名："+name);
        System.out.println("账号："+ID);
        System.out.println("账户余额："+money);
    }
    void changePWD(){
        System.out.print("请输入用户密码：");
        String p=KB.scan();
        PWD=p;
    }
    int getID(){
        return ID;
    }
}

public  class  CustDemo2{
    public static void main(String[] args){
        Cust st = new Cust("Tom",100,"12345",10000);
        System.out.print("请输入您的ID：");
        int ID = Integer.parseInt(KB.scan());
        if(ID == st.getID()){
            while (true){
                System.out.print("1 存款  ");
                System.out.print("2 取款  ");
                System.out.print("3 修改密码  ");
                System.out.print("4 查询  ");
                System.out.print("5 退出");
                int n=Integer.parseInt(KB.scan());
                switch (n){
                case 1:
                    st.saveMoney(); break;
```

```
                case 2:
                    st.getMoney();  break;
                case 3:
                    st.changePWD(); break;
                case 4:
                    st.search();break;
                case 5:
                    System.exit(1);
                }
            }

        else{
            System.out.print("您输入的ID错误! ");
        }
    }
}
```

4. 归纳与提高

观察上述代码，发现主程序段中关于用户操作选择的部分使得主方法段显示过于繁杂，是否可以将此部分功能抽象出来，形成一个独立的方法呢？

对于案例代码修改如下：

```
//文件名：CustDemo2_2.java
import java.io.BufferedReader;
import java.io.InputStreamReader;

class KB{
    //此部分代码与前述代码段相同
}

class Cust {
    //此部分代码与前述代码段相同
}
public  class  CustDemo2_2{
    public static void main(String[] args){
        Cust st = new Cust("Tom",100,"12345",10000);
        System.out.print("请输入您的ID：");
        int ID = Integer.parseInt(KB.scan());
        if(ID == st.getID()){
            run(ID);
        }
        else{
            System.out.print("您输入的ID错误! ");
            }
    }
    static void run(Cust st){
        while (true){
            System.out.print("1 存款  ");
            System.out.print("2 取款  ");
            System.out.print("3 修改密码  ");
            System.out.print("4 查询  ");
            System.out.print("5 退出");
            int n=Integer.parseInt(KB.scan());
            switch (n){
                case 1:
                    st.saveMoney(); break;
                case 2:
```

```
                    st.getMoney();  break;
                case 3:
                    st.changePWD(); break;
                case 4:
                    st.search();break;
                case 5:
                    System.exit(1);
            }
        }
    }
}
```

在主方法段中抽象出的 run()方法中具有一个 Cust 类的形参，说明当有多个 Cust 类的对象时，对象要作为参数在方法间进行传递，这使得方法间的关联关系（也可称为耦合度）增强，这是软件设计过程中应该尽量避免的问题。

每一个到银行存取款的顾客都要进行菜单选择，所以可以把菜单选择看成是 Cust 类对象的一种行为，因此可以把菜单选择操作的方法 run()从主方法中提取出来，放置到 Cust 类中，作为 Cust 类的成员方法。由于 run()是 Cust 类的成员方法，所以不用再将 Cust 类的对象作为参数传递，而且在主方法中要采用“对象.方法名”的形式访问方法。

修改代码如下：

```
//文件名：CustDemo2_3.java
import java.io.BufferedReader;
import java.io.InputStreamReader;

class KB{
    //此部分代码与前述代码段相同
}

class Cust {
    //此部分代码与前述代码段相同
    //下述代码为菜单操作方法
    void  run(){
        while (true){
            System.out.print("1 存款  ");
            System.out.print("2 取款  ");
            System.out.print("3 修改密码  ");
            System.out.print("4 查询  ");
            System.out.print("5 退出");
            int n=Integer.parseInt(KB.scan());
                switch (n){
                    case 1:
                        saveMoney();    break;
                    case 2:
                        getMoney(); break;
                    case 3:
                        changePWD();    break;
                    case 4:
                        search();break;
                    case 5:
                        System.exit(1);
                }
            }
    }
```

```
}
public  class  CustDemo2_3{
    public static void main(String[] args){
        Cust st = new Cust("Tom",100,"12345",10000);
        System.out.print("请输入您的ID：");
        int ID = Integer.parseInt(KB.scan());
        if(ID == st.getID()){
            st.run();
        }
        else{
            System.out.print("您输入的ID错误！");
        }
    }
}
```

当银行账户数较多时，不能采用 st1、st2、st3 等变量去存储对象的引用，可以考虑用对象数组实现。Java 中的数组中既能存储基本数据类型的值，也能存储对象。对象数组和基本数据类型数组在使用方法上几乎是完全一致的，唯一的差别在于对象数组容纳的是对象的引用，而基本数据类型数组容纳的是具体的值。

引入对象数组存储 4 个银行账户的信息，修改代码如下：

```
//文件名：CustDemo2_4.java
import java.io.BufferedReader;
import java.io.InputStreamReader;

class KB{
    //此部分代码与前述代码段相同
}

class Cust {
    //此部分代码与前述代码段相同
}
public  class  CustDemo2_4{
    public static void main(String[] args){
        int i,j=0;
        Cust st[]=new Cust[4];
        st[0]=new Cust("Mike",1000,"111",111);
        st[1]=new Cust("Bob",2000,"222",222);
        st[2]=new Cust("cindy",3000,"333",333);
        st[3]=new Cust("ruby",4000,"444",444);

        boolean  flag=false;      //用于判断是否是合法的账户
        while (true){
            System.out.println("请输入您的ID：");
            int ID=Integer.parseInt(KB.scan());
            for ( i=0;i<4;i++){
                if (ID==st[i].ID){
                    flag=true;
                    j=i;
                }
            }
            if (flag){
                st[j].run();
            }
            else{
                System.out.println("您输入的账号不正确，请重新输入！");
```

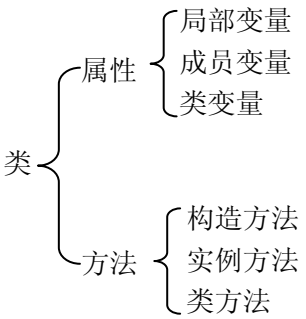
```
                continue;
            }
            System.out.println("是否还有顾客，没有请按N");
            String str=KB.scan();
            if (str.equals("N")||str.equals("n")){
                break;
            }
        }
    }
}
```

在 Java 中，为了数据的安全性往往不直接改动成员变量的值，而是通过声明对象来调用对象的 set()、get()方法，这充分体现出了面向对象的封装特性，对类的内部数据进行隐藏。Java 程序一般将 A 类的属性修饰符设置为 private，要想在 B 类中引用该属性，就可以在 A 类中定义修饰符为 public 的 set()、get()方法以设置和获取 private 型的属性值。读者可以考虑在案例代码中增加 set()、get()方法来设置和获取属性值。

5.8 本 章 小 结

本章主要以银行账户的相关内容为主线，介绍了面向对象的基本概念、构建类的方法、对象的定义及引用方法、静态成员与实例成员的定义方法及区别、方法的重载、类的封装与访问控制等内容。

类主要包括属性和方法：



1. 属性

（1）局部变量：只有离定义的局部变量最近的一对｛｝中的语句可以使用该变量，局部变量为方法和语句块的内部变量，例如：

```
public static void main(){
    int a =0;
}
```

（2）成员变量：也叫全局变量或实例变量，位于方法的外部，是类内部定义的变量，不一定需要赋初值，所有的实例方法都可以使用，例如：

```
public class Hello{
    String str="大家好" ;
    public static void main(String[] args){
        System.out.println(str);
    }
}
```

（3）类变量：用 static 来声明的可供所有对象共享的变量，通过“类名.属性名”调用，例如：

```
public class Hello{
    static String str="大家好" ;
    public static void main(String[] args){
        System.out.println(Hello.str);
    }
}
```

2. 方法

（1）构造方法：构造方法主要对变量初始化，相当于给变量赋值。使用构造方法应注意以下几点：

① 构造方法的名字和类的名字要一模一样。

② 构造方法无返回值，也无须写 void。

③ 使用时用 new 操作符调用，创建对象。

当构造方法同名，但参数不同时，称为构造方法的重载。当我们在调用方法的时候，虚拟机会自动寻找参数相同的方法。不同的参数主要体现在以下几种形式：

① 参数个数不同。

② 参数类型不同。

③ 参数的顺序不同。

（2）类方法和实例方法的区别。

用 static 修饰的方法是类方法，也称为静态方法，目前我们接触到的 main()方法是一个类方法。静态方法只能调用静态成员变量和方法，一个类的静态方法可以不创建该类的对象而通过“类名.方法名”的形式进行调用。因为静态方法是属于类的方法，不属于某个具体对象，所以静态方法不能访问非静态成员。

实例方法也称为非静态方法，不用 static 修饰，需要创建类的对象后，用“对象名.方法”的方式调用方法。

Java 中的访问控制符的作用是说明被声明的内容（类、属性、方法和构造方法）的访问权限。就像发布的文件一样，在文件中标注机密，就是说明该文件只可以被某些人阅读。

访问控制在面向对象技术中处于很重要的地位，合理地使用访问控制符，可以通过降低类和类之间的耦合性（关联性）来降低整个项目的复杂度，也便于整个项目的开发和维护。具体的实现就是通过访问控制符将类中会被其他类调用的内容开放出来，而把不希望别人调用的内容隐藏起来，这样一个类开放的信息变得比较有限，从而降低了整个项目开放的信息；另外因为不被别人调用的功能被隐藏起来，在修改类内部隐藏的内容时，只要最终的功能没有改变，即使改变功能的实现方式，项目中其他的类也不需要更改，这样可以提高代码的可维护性，便于项目代码的修改。

在 4 种访问控制中，public 一般称作公共权限，其限制最小，也可以说没有限制。使用 public 修饰的内容可以在其他所有位置访问，只要能访问到对应的类，就可以访问到类内部 public 修饰的内容。一般在项目中开放的方法和构造方法使用 public 修饰，开放给项目使用的类也使用 public 修饰。protected 一般称作继承权限，使用 protected 修饰的内容可以被同一

个包中的类访问，也可以被不同包内部的子类访问，一般用于修饰只开放给子类的属性、方法和构造方法。无访问控制符一般称作包权限，无访问控制符修饰的内容可以被同一个包中的类访问，一般用于修饰项目中一个包内部的功能类，这些类的功能只是辅助其他的类实现，而为包外部的类提供功能。private 一般称作私有权限，其限制最大，类似于文件中的绝密，使用 private 修饰的内容只能在当前类中访问，而不能被类外部的任何内容访问，一般修饰不开放给外部使用的内容，修改 private 的内容一般对外部的实现没有影响。

理论练习题

一、判断题

1．Java 源程序由类定义组成，每个程序可以定义若干个类，但只有一个主类。（　　）

2．即使一个类中未显式定义构造方法，也会有一个默认的构造方法，默认的构造方法是无参的，方法体为空。（　　）

3．Java 语言中的数组元素只能是基本数据类型而不能为对象类型。（　　）

4. 构造方法用于创建类的实例对象，构造方法名应与类名相同，返回类型为 void。（　　）

5．可以用 new 来创建一个类的实例，即“对象”。（　　）

6．Java 中类的构造方法只能有一个。（　　）

7．类变量在内存中只有一个副本，被该类的所有对象共享。每当创建一个实例，就会为实例变量分配一次内存，实例变量可以在内存中有多个副本，互不影响。（　　）

8．类中说明的方法可以定义在类体外。（　　）

9．实例方法中不能引用类变量。（　　）

10．创建对象时系统将调用适当的构造方法给对象初始化。（　　）

11．使用运算符 new 创建对象时，赋给对象的值实际上是一个引用值。（　　）

12． 对象赋值实际上是同一个对象具有两个不同的名字，它们都有同一个引用值。（　　）

13．对象可作方法参数，对象数组不能作方法参数。（　　）

14．class 是定义类的唯一关键字。（　　）

15．类的 public 类型的成员变量不可以被继承。（　　）

二、填空题

1．类的修饰符分为_____、_____。

2．程序中定义类使用的关键字是_____，每个类的定义由类头定义、类体定义两部分组成，其中类体部分包括_____和_____。

3．main 方法的声明格式是_____。

4．创建一个类的对象的运算符是_____。

5．java 源文件中最多只能有一个_____类，其他类的个数不限。

6．类方法不能直接访问其所属类的_____变量和_____方法，只可直接访问其所属类的_____变量和_____方法。

7．类成员的访问控制符有_____、_____、_____和默认 4 种。

8．protected 类型的类成员可被同一_____、同一包中的_____和不同包中的_____的代

码访问引用。

9．下面是一个类的定义：

```
public class ______ {
  int x, y;
  Myclass ( int i, ______) {     // 构造方法
    x=i;
    y=j;
  }
}
```

10．下面程序的运行结果是_____。

```
public class D{
    public static void main(String args[]){
        int d=21;
        Dec dec=new Dec( );
        dec.decrement(d);
        System.out.println(d);
    }
}
class Dec{
    public void decrement(int decMe){
        decMe = decMe – 1;
    }
}
```

11．下面程序的运行结果是_____。

```
public class Q6{
    public static void main(String args[ ]){
        Holder h=new Holder( );
        h.held=100;
        h.bump(h);
        System.out.println(h.held);
    }
}
class Holder{
    public int held;
    public void bump(Holder theHolder){
        theHolder.held --;
    }
}
```

三、选择题

1．以下关于 application 的说明，正确的是（　　）。

```
1 public class StaticStuff {
2     static int x=15;
3     static { x*=3; }
4     public static void main(String args[]) {
5         System.out.println("x="+x);
6     }
7     static {x/=3;}
8 }
```

A．3 号行与 7 号行不能通过编译，因为缺少方法名和返回类型

B．7 号行不能通过编译，因为只能有一个静态初始化器

C．编译通过，执行结果为：x=15

D．编译通过，执行结果为：x=3

2．类 Text1 定义如下：

```
public class Test1{
   public float aMethod(Float a, float b){ }
    ***
}
```

将以下（　　）方法插入行　***　是不合法的。

A．public float aMethod(float a,float b,float c){ }

B．public float aMethod(float c,float d){ }

C．public int aMethod(int a,int b){ }

D．public float aMethod(int a,int b,int c){ }

3．以下关于构造方法的描述错误的是（　　）。

A．构造方法的返回类型只能是 void 型

B．构造方法是类的一种特殊方法，它的方法名必须与类名相同

C．构造方法的主要作用是完成对类的对象的初始化工作

D．一般在创建新对象时，系统会自动调用构造方法

4．在 Java 中，一个类可同时定义许多同名的方法，这些方法的形式参数个数、类型或顺序各不相同，传回的值也可以不相同。这种面向对象程序的特性称为（　　）。

A．隐藏　　B．覆盖　　C．重载　　D．Java 不支持此特性

5．假设 A 类有如下定义，设 a 是 A 类的一个实例，下列语句调用错误的是（　　）。

```
public class  A {
   int  i;
   static  String  s;
   void  method1() {   }
   static  void  method2()  {   }
}
```

A．System.out.println(a.i);　　B．a.method1();

C．A.method1();　　D．A.method2();

6．设 x、y 为已定义的类名，下列声明 x 类的对象 x1 的语句中正确的是（　　）。

A．static x x1;　　B．public x x1=new x(int 123);

C．y x1;　　D．x x1=x( );

7．已知有下列类的说明，下列语句正确的是（　　）。

```
public class Test{
   private float f = 1.0f;
   int m = 12;
   static int n=1;
   public static void main (String args[ ]) {
     Test t = new Test( );
   }
}
```

A．t.f;　　B．this.n;　　C．Test.m;　　D．Test.f;

上机实训题

1．创建 MyProject3 项目并创建 Person 类，设置 name、sex 及 age 成员域。设置带参构造方法及无参构造方法，设置 toString（该名称可自定义）方法将类的 3 个成员域转化成字符串便于显示输出。创建主类 CreatPerson，通过 Person 类创建对象，显示输出该对象的各种属性。

2．创建 MaxArray 类，并利用该类的对象求一维数组中的最大值。

3．创建 Circle 类并添加静态属性 r（成员变量），并定义一个常量 PI=3.142，在类 Circle 中添加两种方法，分别计算周长和面积；编写主类 CreatCircle，利用类 Circle 输出 r=2 时圆的周长和面积。