3 Java 类与对象详解

类和对象是面向对象编程中最基本、也是最重要的特征之一。从本章开始，将介绍如何进行面向对象的程序开发，以及程序开发的过程中，所需要具备的重要思想是什么？编程思想是很多学习编程的人，学习时间长却学不好的一个主要原因。

❖ **本章内容：**

3.1 Java 类的定义及其实例化

如果你不了解类和对象的概念，请猛击这里：Java 类和对象的概念

类必须先定义才能使用。类是创建对象的模板，创建对象也叫类的实例化。

下面通过一个简单的例子来理解 Java 中类的定义：

```
01. public class Dog{
02.     String name;
03.     int age;
04.
05.     void bark(){  // 汪汪叫
06.         System.out.println("汪汪，不要过来");
07.     }
08.
09.     void hungry(){  // 饥饿
10.         System.out.println("主人，我饿了");
11.     }
12. }
```

对示例的说明：

- public 是类的修饰符，表明该类是公共类，可以被其他类访问。修饰符将在下节讲解。
- class 是定义类的关键字。
- Dog 是类名称。
- name、age 是类的成员变量，也叫属性；bark()、hungry() 是类中的函数，也叫方法。

一个类可以包含以下类型变量：

- 局部变量：在方法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，方法结束后，变量就会自动销毁。
- 成员变量：成员变量是定义在类中、方法体之外的变量。这种变量在创建对象的时候实例化（分配内存）。成员变量可以被类中的方法和特定类的语句访问。
- 类变量：类变量也声明在类中，方法体之外，但必须声明为 static 类型。static 也是修饰符的一种，将在下节讲解。

构造方法

在类实例化的过程中自动执行的方法叫做构造方法，它不需要你手动调用。构造方法可以在类实例化的过程中做一些初始化的工作。

构造方法的名称必须与类的名称相同，并且没有返回值。

每个类都有构造方法。如果没有显式地为类定义构造方法，Java 编译器将会为该类提供一个默认的构造方法。

下面是一个构造方法示例：

```
public class Dog{
    String name;
    int age;

    // 构造方法，没有返回值
    Dog(String name1, int age1){
        name = name1;
        age = age1;
        System.out.println("感谢主人领养了我");
    }

    // 普通方法，必须有返回值
    void bark(){
        System.out.println("汪汪，不要过来");
    }

    void hungry(){
        System.out.println("主人，我饿了");
    }

    public static void main(String arg[]){
        // 创建对象时传递的参数要与构造方法参数列表对应
        Dog myDog = new Dog("花花", 3);
    }
}
```

运行结果：

感谢主人领养了我

说明：

- 构造方法不能被显式调用。
- 构造方法不能有返回值，因为没有变量来接收返回值。

创建对象

对象是类的一个实例，创建对象的过程也叫类的实例化。对象是以类为模板来创建的。

在 Java 中，使用 new 关键字来创建对象，一般有以下三个步骤：

- 声明：声明一个对象，包括对象名称和对象类型。
- 实例化：使用关键字 new 来创建一个对象。
- 初始化：使用 new 创建对象时，会调用构造方法初始化对象。

例如：

```
Dog myDog;  // 声明一个对象
myDog = new Dog("花花", 3);  // 实例化
```

也可以在声明的同时进行初始化：

```
01.  Dog myDog = new Dog("花花", 3);
```

访问成员变量和方法

通过已创建的对象来访问成员变量和成员方法，例如：

```
01.  // 实例化
02.  Dog myDog = new Dog("花花", 3);
03.  // 通过点号访问成员变量
04.  myDog.name;
05.  // 通过点号访问成员方法
06.  myDog.bark();
```

下面的例子演示了如何访问成员变量和方法：

```
public class Dog{
    String name;
    int age;

    Dog(String name1, int age1){
        name = name1;
        age = age1;
        System.out.println("感谢主人领养了我");
    }

    void bark(){
        System.out.println("汪汪，不要过来");
    }

    void hungry(){
        System.out.println("主人，我饿了");
    }

    public static void main(String arg[]){
        Dog myDog = new Dog("花花", 3);
        // 访问成员变量
        String name = myDog.name;
        int age = myDog.age;
        System.out.println("我是一只小狗，我名字叫" + name + "，我" + age + "岁了");
        // 访问方法
        myDog.bark();
        myDog.hungry();
    }
}
```

运行结果：

感谢主人领养了我
我是一只小狗，我名字叫花花，我 3 岁了
汪汪，不要过来
主人，我饿了

3.2 Java 访问修饰符（访问控制符）

Java 通过修饰符来控制类、属性和方法的访问权限和其他功能，通常放在语句的最前端。例如：

```
public class className {
    // body of class
}
private boolean myFlag;
static final double weeks = 9.5;
protected static final int BOXWIDTH = 42;
public static void main(String[] arguments) {
    // body of method
}
```

Java 的修饰符很多，分为访问修饰符和非访问修饰符。本节仅介绍访问修饰符，非访问修饰符会在后续介绍。

访问修饰符也叫访问控制符，是指能够控制类、成员变量、方法的使用权限的关键字。

在面向对象编程中，访问控制符是一个很重要的概念，可以使用它来保护对类、变量、方法和构造方法的访问。

Java 支持四种不同的访问权限：

修饰符	说明
public	共有的，对所有类可见。
protected	受保护的，对同一包内的类和所有子类可见。
private	私有的，在同一类内可见。
默认的	在同一包内可见。默认不使用任何修饰符。

public：公有的

被声明为 public 的类、方法、构造方法和接口能够被任何其他类访问。

如果几个相互访问的 public 类分布在不用的包中，则需要导入相应 public 类所在的包。由于类的继承性，类所有的公有方法和变量都能被其子类继承。

下面的方法使用了公有访问控制：

```
public static void main(String[] arguments) {
    // body of method
}
```

Java 程序的 main() 方法必须设置成公有的，否则，Java 解释器将不能运行该类。

protected：受保护的

被声明为 protected 的变量、方法和构造方法能被同一个包中的任何其他类访问，也能够被不同包中的子类访问。

protected 访问修饰符不能修饰类和接口，方法和成员变量能够声明为 protected，但是接口的成员变量和成员方法不能声明为 protected。

子类能访问 protected 修饰符声明的方法和变量，这样就能保护不相关的类使用这些方法和变量。

下面的父类使用了 protected 访问修饰符，子类~~重载~~复写了父类的 bark()方法。

```
public class Dog{
    protected void bark() {
        System.out.println("汪汪，不要过来");
    }
}

class Teddy extends Dog{   // 泰迪
    void bark() {
        System.out.println("汪汪，我好怕，不要跟着我");
    }
}
```

如果把 bark()方法声明为 private，那么除了 Dog 之外的类将不能访问该方法。如果把 bark()声明为 public，那么所有的类都能够访问该方法。如果我们只想让该方法对其所在类的子类可见，则将该方法声明为 protected。

private：私有的

私有访问修饰符是最严格的访问级别，所以被声明为 private 的方法、变量和构造方法只能被所属类访问，并且类和接口不能声明为 private。

声明为私有访问类型的变量只能通过类中公共的 Getter/Setter 方法被外部类访问。

private 访问修饰符的使用主要用来隐藏类的实现细节和保护类的数据。

下面的类使用了私有访问修饰符：

```
public class Dog{
    private String name;
```

```
    private int age;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
}
```

例子中，Dog 类中的 name、age 变量为私有变量，所以其他类不能直接得到和设置该变量的值。为了使其他类能够操作该变量，定义了两对 public 方法，getName()/setName() 和 getAge()/setAge()，它们用来获取和设置私有变量的值。

this 是 Java 中的一个关键字，本章会讲到，你可以点击 Java this 关键字详解 预览。

在类中定义访问私有变量的方法，习惯上是这样命名的：在变量名称前面加“get”或“set”，并将变量的首字母大写。例如，获取私有变量 name 的方法为 getName()，设置 name 的方法为 setName()。这些方法经常使用，也有了特定的称呼，称为 Getter 和 Setter 方法。

默认的：不使用任何关键字

不使用任何修饰符声明的属性和方法，对同一个包内的类是可见的。接口里的变量都隐式声明为 public static final，而接口里的方法默认情况下访问权限为 public。

如下例所示，类、变量和方法的定义没有使用任何修饰符：

```
class Dog{
    String name;
    int age;

    void bark(){   // 汪汪叫
        System.out.println("汪汪，不要过来");
    }

    void hungry(){   // 饥饿
        System.out.println("主人，我饿了");
    }
}
```

访问控制和继承

请注意以下方法继承（不了解继承概念的读者可以跳过这里，或者点击 Java 继承和多态 预览）的规则：

- 父类中声明为 public 的方法在子类中也必须为 public。
- 父类中声明为protected的方法在子类中要么声明为protected，要么声明为public。不能声明为private。
- ~~父类中默认修饰符声明的方法，能够在子类中声明为 private~~。子类复写方法时不能有比父类更严格的权限
- 父类中声明为 private 的方法，不能够被继承。

如何使用访问控制符

访问控制符可以让我们很方便的控制代码的权限：

- 当需要让自己编写的类被所有的其他类访问时，就可以将类的访问控制符声明为 public。
- 当需要让自己的类只能被自己的包中的类访问时，就可以省略访问控制符。
- 当需要控制一个类中的成员数据时，可以将这个类中的成员数据访问控制符设置为 public、protected，或者省略。

3.3 Java 变量的作用域

在 Java 中，变量的作用域分为四个级别：类级、对象实例级、方法级、块级。

类级变量又称全局级变量或静态变量，需要使用 static 关键字修饰，你可以与 C/C++ 中的 static 变量对比学习。类级变量在类定义后就已经存在，占用内存空间，可以通过类名来访问，不需要实例化。

对象实例级变量就是成员变量，实例化后才会分配内存空间，才能访问。

方法级变量就是在方法内部定义的变量，就是局部变量。

块级变量就是定义在一个块内部的变量，变量的生存周期就是这个块，出了这个块就消失了，比如 if、for 语句的块。块是指由大括号包围的代码，例如：

```
{
    int age = 3;
    String name = "www.youknow.net";
    // 正确，在块内部可以访问 age 和 name 变量
    System.out.println( name + "已经" + age + "岁了");
}
// 错误，在块外部无法访问 age 和 name 变量
System.out.println( name + "已经" + age + "岁了");
```

说明：

- 方法内部除了能访问方法级的变量，还可以访问类级和实例级的变量。
- 块内部能够访问类级、实例级变量，如果块被包含在方法内部，它还可以访问方法级的变量。
- 方法级和块级的变量必须被显示地初始化，否则不能访问。

演示代码：

```
public class Demo{
    public static String name = "你懂的";  // 类级变量
    public int i; // 对象实例级变量

    // 属性块，在类初始化属性时候运行
    {
        int j = 2;// 块级变量
    }

    public void test1() {
        int j = 3;  // 方法级变量
        if(j == 3) {
            int k = 5;  // 块级变量
        }
        // 这里不能访问块级变量，块级变量只能在块内部访问
        System.out.println("name=" + name + ", i=" + i + ", j=" + j);
    }

    public static void main(String[] args) {
        // 不创建对象，直接通过类名访问类级变量
        System.out.println(Demo.name);

        // 创建对象并访问它的方法
        Demo t = new Demo();
        t.test1();
    }
}
```

运行结果：

```
你懂的
name=你懂的, i=0, j=3
```

3.4 Java this 关键字详解

this 关键字用来表示当前对象本身，或当前类的一个实例，通过 this 可以调用本对象的所有方法和属性。例如：

```
public class Demo{
    public int x = 10;
    public int y = 15;

    public void sum(){
        // 通过 this 点取成员变量
        int z = this.x + this.y;
        System.out.println("x + y = " + z);
```

```
    }

    public static void main(String[] args) {
        Demo obj = new Demo();
        obj.sum();
    }
}
```

运行结果：

```
x + y = 25
```

上面的程序中，obj 是 Demo 类的一个实例，this 与 obj 等价，执行 int z = this.x + this.y;，就相当于执行 int z = obj.x + obj.y;。

注意：this 只有在类实例化后才有意义。

使用 this 区分同名变量

成员变量与方法内部的变量重名时，希望在方法内部调用成员变量，怎么办呢？这时候只能使用 this，例如：

```
public class Demo{
    public String name;
    public int age;
    public Demo(String name, int age){
        this.name = name;
        this.age = age;
    }
    public void say(){
        System.out.println("网站的名字是" + name + "，已经成立了" + age + "年");
    }
    public static void main(String[] args) {
        Demo obj = new Demo("你懂的", 3);
        obj.say();
    }
}
```

运行结果：

```
网站的名字是你懂的，已经成立了 3 年
```

形参的作用域是整个方法体，是局部变量。在 Demo()中，形参和成员变量重名，如果不使用 this，访问到的就是局部变量 name 和 age，而不是成员变量。在 say() 中，我们没有使用 this，因为成员变量的作用域是整个实例，当然也可以加上 this：

```
public void say(){
    System.out.println("网站的名字是" + this.name + "，已经成立了" + this.age + "年");
}
```

Java 默认将所有成员变量和成员方法与 this 关联在一起，因此使用 this 在某些情况下是多余的。

作为方法名来初始化对象

也就是相当于调用本类的其它构造方法，它必须作为构造方法的第一句。示例如下：

```
public class Demo{
    public String name;
    public int age;
    public Demo(){
        this("你懂的", 3);
    }
    public Demo(String name, int age){
        this.name = name;
        this.age = age;
    }
    public void say(){
        System.out.println("网站的名字是" + name + "，已经成立了" + age + "年");
    }
    public static void main(String[] args) {
        Demo obj = new Demo();
        obj.say();
    }
}
```

运行结果：
网站的名字是你懂的，已经成立了 3 年

值得注意的是：

- 在构造方法中调用另一个构造方法，调用动作必须置于最起始的位置。
- 不能在构造方法以外的任何方法内调用构造方法。
- 在一个构造方法内只能调用一个构造方法。

上述代码涉及到方法重载，即 Java 允许出现多个同名方法，只要参数不同就可以。后续章节会讲解。

作为参数传递

需要在某些完全分离的类中调用一个方法，并将当前对象的一个引用作为参数传递时。例如：

```
public class Demo{
    public static void main(String[] args){
        B b = new B(new A());
    }
}
class A{
    public A(){
        new B(this).print();    // 匿名对象
    }
```

```
    public void print(){
        System.out.println("Hello from A!");
    }
}
class B{
    A a;
    public B(A a){
        this.a = a;
    }
    public void print() {
        a.print();
        System.out.println("Hello from B!");
    }
}
```

运行结果：

```
Hello from A!
Hello from B!
```

匿名对象就是没有名字的对象。如果对象只使用一次，就可以作为匿名对象，代码中 new B(this).print(); 等价于 (new B(this)).print();，先通过 new B(this) 创建一个没有名字的对象，再调用它的方法。

3.5 Java 方法重载

在 Java 中，同一个类中的多个方法可以有相同的名字，只要它们的参数列表不同就可以，这被称为**方法重载(method overloading)**。

参数列表又叫参数签名，包括参数的类型、参数的个数和参数的顺序，只要有一个不同就叫做参数列表不同。

重载是面向对象的一个基本特性。

下面看一个详细的实例。

```
public class Demo{
    // 一个普通的方法，不带参数
    void test(){
        System.out.println("No parameters");
    }
    // 重载上面的方法，并且带了一个整型参数
    void test(int a){
        System.out.println("a: " + a);
    }
    // 重载上面的方法，并且带了两个参数
    void test(int a,int b){
        System.out.println("a and b: " + a + " " + b);
```

```
    }
    // 重载上面的方法，并且带了一个双精度参数
    double test(double a){
        System.out.println("double a: " + a);
        return a*a;
    }

    public static void main(String args[]){
        Demo obj= new Demo();
        obj.test();
        obj.test(2);
        obj.test(2,3);
        obj.test(2.0);
    }
}
```

运行结果：

```
No parameters
a: 2
a and b: 2 3
double a: 2.0
```

通过上面的实例，读者可以看出，重载就是在一个类中，有相同的函数名称，但形参不同的函数。重载的结果，可以让一个程序段尽量减少代码和方法的种类。

说明：

- 参数列表不同包括：个数不同、类型不同和顺序不同。
- 仅仅参数变量名称不同是不可以的。
- 跟成员方法一样，构造方法也可以重载。
- 声明为 final 的方法不能被重载。
- 声明为 static 的方法不能被重载，但是能够被再次声明。

方法的重载的规则：

- 方法名称必须相同。
- 参数列表必须不同（个数不同、或类型不同、参数排列顺序不同等）。
- 方法的返回类型可以相同也可以不相同。
- 仅仅返回类型不同不足以成为方法的重载。

方法重载的实现：

方法名称相同时，编译器会根据调用方法的参数个数、参数类型等去逐个匹配，以选择对应的方法，如果匹配失败，则编译器报错，这叫做**重载分辨**。

3.6 Java 类的基本运行顺序

我们以下面的类来说明一个基本的 Java 类的运行顺序：

```
public class Demo{
    private String name;
    private int age;

    public Demo(){
        name = "你懂的";
        age = 3;
    }
    public static void main(String[] args){
        Demo obj = new Demo();
        System.out.println(obj.name + "的年龄是" + obj.age);
    }
}
```

基本运行顺序是：
先运行到第 9 行，这是程序的入口。
然后运行到第 10 行，这里要 new 一个 Demo，就要调用 Demo 的构造方法。
就运行到第 5 行，注意：可能很多人觉得接下来就应该运行第 6 行了，错！初始化一个类，必须先初始化它的属性。
因此运行到第 2 行，然后是第 3 行。
属性初始化完过后，才回到构造方法，执行里面的代码，也就是第 6 行、第 7 行。
然后是第 8 行，表示 new 一个 Demo 实例完成。
然后回到 main 方法中执行第 11 行。
然后是第 12 行，main 方法执行完毕。

作为程序员，应该清楚程序的基本运行过程，否则糊里糊涂的，不利于编写代码，也不利于技术上的发展。

3.7 Java 包装类、拆箱和装箱详解

虽然 Java 语言是典型的面向对象编程语言，但其中的八种基本数据类型并不支持面向对象编程，基本类型的数据不具备“对象”的特性——不携带属性、没有方法可调用。 沿用它们只是为了迎合人类根深蒂固的习惯，并的确能简单、有效地进行常规数据处理。

这种借助于非面向对象技术的做法有时也会带来不便，比如引用类型数据均继承了 Object 类的特性，要转换为 String 类型（经常有这种需要）时只要简单调用 Object 类中定义的 toString()即可，而基本数据类型转换为 String 类型则要麻烦得多。为解决此类问题 ，Java 为每种基本数据类型分别设计了对应的类，称之为包装类(Wrapper Classes)，也有教材称为外覆类或数据类型类。

基本数据类型	对应的包装类
byte	Byte
short	Short
int	Integer
long	Long
char	Character
float	Float
double	Double
boolean	Boolean

每个包装类的对象可以封装一个相应的基本类型的数据，并提供了其它一些有用的方法。包装类对象一经创建，其内容（所封装的基本类型数据值）不可改变。

基本类型和对应的包装类可以相互装换：

- 由基本类型向对应的包装类转换称为装箱，例如把 int 包装成 Integer 类的对象；
- 包装类向对应的基本类型转换称为拆箱，例如把 Integer 类的对象重新简化为 int。

包装类的应用

八个包装类的使用比较相似，下面是常见的应用场景。

1) 实现 int 和 Integer 的相互转换

可以通过 Integer 类的构造方法将 int 装箱，通过 Integer 类的 intValue 方法将 Integer 拆箱。例如：

```
01. public class Demo {
02.     public static void main(String[] args) {
03.         int m = 500;
04.         Integer obj = new Integer(m);  // 手动装箱
05.         int n = obj.intValue();  // 手动拆箱
06.         System.out.println("n = " + n);
07.
08.         Integer obj1 = new Integer(500);
09.         System.out.println("obj 等价于 obj1？" + obj.equals(obj1));
10.     }
11. }
```

运行结果：

n = 500

obj 等价于 obj1？true

2) 将字符串转换为整数

Integer 类有一个静态的 paseInt() 方法，可以将字符串转换为整数，语法为：

```
parseInt(String s, int radix);
```

s 为要转换的字符串，radix 为进制，可选，默认为十进制。

下面的代码将会告诉你什么样的字符串可以转换为整数：

```
public class Demo {
    public static void main(String[] args) {
        String str[] = {"123", "123abc", "abc123", "abcxyz"};

        for(String str1 : str){
            try{
                int m = Integer.parseInt(str1, 10);
                System.out.println(str1 + " 可以转换为整数 " + m);
            }catch(Exception e){
                System.out.println(str1 + " 无法转换为整数");
            }
        }
    }
}
```

运行结果：

123 可以转换为整数 123

123abc 无法转换为整数

abc123 无法转换为整数

abcxyz 无法转换为整数

3) 将整数转换为字符串

Integer 类有一个静态的 toString() 方法，可以将整数转换为字符串。例如：

```
public class Demo {
    public static void main(String[] args) {
        int m = 500;
        String s = Integer.toString(m);
        System.out.println("s = " + s);
    }
}
```

运行结果：

s = 500

自动拆箱和装箱

上面的例子都需要手动实例化一个包装类，称为手动拆箱装箱。Java 1.5(5.0) 之前必须手动拆箱装箱。

Java 1.5 之后可以自动拆箱装箱，也就是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，这将大大方便程序员的代码书写。例如：

```
01. public class Demo {
02.     public static void main(String[] args) {
03.         int m = 500;
04.         Integer obj = m;  // 自动装箱
05.         int n = obj;  // 自动拆箱
06.         System.out.println("n = " + n);
07.
08.         Integer obj1 = 500;
09.         System.out.println("obj 等价于 obj1？" + obj.equals(obj1));
10.     }
11. }
```

运行结果：
n = 500
obj 等价于 obj1？true

自动拆箱装箱是常用的一个功能，读者需要重点掌握。

3.8 再谈 Java 包

在 Java 中，为了组织代码的方便，可以将功能相似的类放到一个文件夹内，这个文件夹，就叫做包。

包不但可以包含类，还可以包含接口和其他的包。

目录以"\"来表示层级关系，例如 E:\Java\workspace\Demo\bin\p1\p2\Test.java。

包以"."来表示层级关系，例如 p1.p2.Test 表示的目录为 \p1\p2\Test.class。

如何实现包

通过 package 关键字可以声明一个包，例如：

```
01. package p1.p2;
02. public class Test {
03.     public Test(){
04.         System.out.println("我是Test类的构造方法");
05.     }
06. }
```

表明 Test 类位于 p1.p2 包中。
包的调用
在 Java 中，调用其他包中的类共有两种方式。

1) 在每个类名前面加上完整的包名
程序举例：

```
public class Demo {
    public static void main(String[] args) {
        java.util.Date today=new java.util.Date();
        System.out.println(today);
    }
}
```

运行结果：

```
Wed Dec 03 11:20:13 CST 2014
```

2) 通过 import 语句引入包中的类
程序举例：

```
import java.util.Date;
// 也可以引入 java.util 包中的所有类
// import java.util.*;

public class Demo {
    public static void main(String[] args) {
        Date today=new Date();
        System.out.println(today);
    }
}
```

运行结果与上面相同。

实际编程中，没有必要把要引入的类写的那么详细，可以直接引入特定包中所有的类，例如 import java.util.*;。

类的路径

Java 在导入类时，必须要知道类的绝对路径。

首先在 E:\Java\workspace\Demo\src\p0\ 目录（E:\Java\workspace\Demo\src\ 是项目源文件的根目录）下创建 Demo.java，输入如下代码：

```
package p0;
import p1.p2.Test;

public class Demo{
    public static void main(String[] args){
        Test obj = new Test();
    }
}
```

再在 E:\Java\workspace\Demo\src\p1\p2 目录下创建 Test.java，输入如下代码：

```
package p1.p2;

public class Test {
    public Test(){
        System.out.println("我是 Test 类的构造方法");
    }
}
```

假设我们将 classpath 环境变量设置为 .;D:\Program Files\jdk1.7.0_71\lib，源文件 Demo.java 开头有 import p1.p2.Test; 语句，那么编译器会先检查 E:\Java\workspace\Demo\src\p0\p1\p2\ 目录下是否存在 Test.java 或 Test.class 文件，如果不存在，会继续检索 D:\Program Files\jdk1.7.0_71\lib\p1\p2\ 目录，两个目录下都不存在就会报错。显然，Test.java 位于 E:\Java\workspace\Demo\src\p1\p2\ 目录，编译器找不到，会报错，怎么办呢？

可以通过 javac 命令的 classpath 选项来指定类路径。

打开 CMD，进入 Demo.java 文件所在目录，执行 javac 命令，并将 classpath 设置为 E:\Java\workspace\Demo\src，如下图所示：

```
C:\Users\mozhiyan>E:

E:\>cd \Java\workspace\Demo\src\p0\

E:\Java\workspace\Demo\src\p0>javac -classpath E:\Java\workspace\Demo\src Demo.java

E:\Java\workspace\Demo\src\p0>
```

运行 Java 程序时，也需要知道类的绝对路径，除了 classpath 环境变量指定的路径，也可以通过 java 命令的 classpath 选项来增加路径，如下图所示：

```
C:\Users\mozhiyan>E:

E:\>cd E:\Java\workspace\Demo\src

E:\Java\workspace\Demo\src>java -classpath E:\Java\workspace\Demo\src p0.Demo
我是Test类的构造方法

E:\Java\workspace\Demo\src>
```

注意 java 命令与 javac 命令的区别，执行 javac 命令需要进入当前目录，而执行 java 命令需要进入当前目录的上级目录，并且类名前面要带上包名。

可以这样来理解，javac 是一个平台命令，它对具体的平台文件进行操作，要指明被编译的文件路径。而 java 是一个虚拟机命令，它对类操作，即对类的描述要用点分的描述形式，并且不能加扩展名，还要注意类名的大小写。

这些命令比较繁杂，实际开发都需要借助 Eclipse，在 Eclipse 下管理包、编译运行程序都非常方便。Eclipse 实际上也是执行这些命令。

包的访问权限

被声明为 public 的类、方法或成员变量，可以被任何包下的任何类使用，而声明为 private 的类、方法或成员变量，只能被本类使用。

没有任何修饰符的类、方法和成员变量，只能被本包中的所有类访问，在包以外任何类都无法访问它。

3.9 Java 源文件的声明规则

当在一个源文件中定义多个类，并且还有 import 语句和 package 语句时，要特别注意这些规则：

- 一个源文件中只能有一个 public 类。
- 一个源文件可以有多个非 public 类。
- 源文件的名称应该和 public 类的类名保持一致。例如：源文件中 public 类的类名是 Employee，那么源文件应该命名为 Employee.java。
- 如果一个类定义在某个包中，那么 package 语句应该在源文件的首行。
- 如果源文件包含 import 语句，那么应该放在 package 语句和类定义之间。如果没有 package 语句，那么 import 语句应该在源文件中最前面。
- import 语句和 package 语句对源文件中定义的所有类都有效。在同一源文件中，不能给不同的类不同的包声明。
- 类有若干种访问级别，并且类也分不同的类型：抽象类和 final 类等。这些将在后续章节介绍。
- 除了上面提到的几种类型，Java 还有一些特殊的类，如内部类、匿名类。

一个简单的例子

在该例子中，我们创建两个类 Employee 和 EmployeeTest，分别放在包 p1 和 p2 中。

Employee 类有四个成员变量，分别是 name、age、designation 和 salary。该类显式声明了一个构造方法，该方法只有一个参数。

在 Eclipse 中，创建一个包，命名为 p1，在该包中创建一个类，命名为 Employee，将下面的代码复制到源文件中：

```
package p1;
```

```
public class Employee{
   String name;
   int age;
   String designation;
   double salary;
   // Employee 类的构造方法
   public Employee(String name){
      this.name = name;
   }
   // 设置 age 的值
   public void empAge(int empAge){
      age =  empAge;
   }
   // 设置 designation 的值
   public void empDesignation(String empDesig){
      designation = empDesig;
   }
   // 设置 salary 的值
   public void empSalary(double empSalary){
      salary = empSalary;
   }
   // 输出信息
   public void printEmployee(){
      System.out.println("Name:"+ name );
      System.out.println("Age:" + age );
      System.out.println("Designation:" + designation );
      System.out.println("Salary:" + salary);
   }
}
```

程序都是从 main 方法开始执行。为了能运行这个程序，必须包含 main 方法并且创建一个对象。

下面给出 EmployeeTest 类，该类创建两个 Employee 对象，并调用方法设置变量的值。

在 Eclipse 中再创建一个包，命名为 p2，在该包中创建一个类，命名为 EmployeeTest，将下面的代码复制到源文件中：

```
package p2;
import p1.*;

public class EmployeeTest{
   public static void main(String args[]){
      // 创建两个对象
      Employee empOne = new Employee("James Smith");
      Employee empTwo = new Employee("Mary Anne");

      // 调用这两个对象的成员方法
```

```
        empOne.empAge(26);
        empOne.empDesignation("Senior Software Engineer");
        empOne.empSalary(1000);
        empOne.printEmployee();

        empTwo.empAge(21);
        empTwo.empDesignation("Software Engineer");
        empTwo.empSalary(500);
        empTwo.printEmployee();
    }
}
```

编译并运行 EmployeeTest 类，可以看到如下的输出结果：

```
Name:James Smith
Age:26
Designation:Senior Software Engineer
Salary:1000.0
Name:Mary Anne
Age:21
Designation:Software Engineer
Salary:500.0
```