第3章 面向对象程序设计（基础）

3.1 面向对象的概念

3.1.1 什么是面向对象

面向对象程序设计的思维方式是一种更符合人们思考习惯的方式。面向对象将构成问题的事物分解成各个对象，这些对象是为了描述某个事物在整个问题解决步骤中的行为。

面向对象以对象为核心，强调事件的角色、主体，在宏观上使用面向对象进行把控，而微观上依然是面向过程。如果说面向过程的思想是执行者，那么面向对象的思想就是指挥者。

3.1.2 面向对象的特性

面向对象具有抽象、封装、继承、多态的特性，更符合程序设计中“高内聚、低耦合”的主旨，其编写的代码的可维护性、可读性、复用性、可扩展性远比面向过程思想编写的代码强，但是性能相比面向过程要偏低一些。

封装、继承、多态是面向对象的三大特性，这是任何一门面向对象编程语言都要具备的。

- 封装：指隐藏对象的属性和实现细节，仅对外提供公共的访问方式。
- 继承：继承就是子类继承父类的特征和行为，使得子类对象（实例）具有父类的实例属性和方法，或子类从父类继承方法，使得子类具有父类相同的行为。
- 多态：指的是同一个方法调用，由于对象不同可能会有不同的行为。

3.1.3 类和对象

从编程的角度来说，万物皆对象，可以理解为，现实中存在的任何一个具体的事物都是一个对象，如一张桌子、一个人、一支笔。

而将现实中一类事物抽象化，提取出这一类事物共有的属性和行为，就形成了类。比如

班上的每一位同学都是一个对象，都具备姓名、年龄、学号属性，都拥有学习、睡觉行为。而把班上的同学进行抽象，就是一个学生类。更直白点说，类是抽象的概念，对象是具体的概念。“桌子”“椅子”“计算机”就是类，“某张桌子”“某把椅子”“某台计算机”就是对象。

可以把类看作一个模板，或者蓝图，系统根据类的定义创造出对象。要造一辆汽车，类就是这辆汽车的图纸，它规定了汽车的详细信息，人们根据蓝图将汽车造出来，造出来的汽车就是对象。

3.2 面向对象编程

3.2.1 类的定义


扫一扫

Java 使用 class 关键字定义一个类。一个 Java 文件中可以有多个类，但最多只能有一个 public 修饰的类，并且这个类的类名必须与文件名完全一致。此外，类名需要符合标识符规则，并且遵循大写字母开头的驼峰规则。比如下面的代码就是创建类的代码。

```
package com.yyds.unit3.demo;
public class DemoObject {
}
class Entity1 {
}
class Entity2 {
   }
```

上面虽然创建了三个类，但它们就像空白的图纸，里面什么都没有，这样的类没有任何意义，所以需要定义类的具体信息。

类主要由变量（字段）和方法组成。

- 变量（字段 field）：其定义格式：

```
修饰符  变量类型  变量名  =  [默认值];
```

- 方法（行为 action）：其定义格式：

```
修饰符 返回值类型  方法名(形参列表) {}
```

下面的代码创建一个 Student 类，拥有 name 和 age 变量，以及 eat()和 study()方法，并且这两个方法也称作成员方法。

```
package com.yyds.unit3.demo;
public class Student {
    //变量,此处也称为成员变量
    String name;
    int age;
    //方法,此处也称为成员方法
    void eat(String food) {
```

```
        System.out.println(name + "吃" + food);
    }
    void study() {
        System.out.println(name + "年龄" + age + "岁,在学习 Java");
    }
}
```

在成员方法中,可以随意访问类中定义的成员变量。

注意,某些资料中可能会将成员变量称为“属性”,事实上这是错误的。在 Java 类中,属性与成员变量是有区别的,这一点将在后面介绍。


扫一扫

3.2.2 对象的创建与使用

上面已经创建了一个 Student 类,类是一张图纸,应当使用这张“图纸”创建具有实际意义的对象。在 Java 中,对象(object)也称作 instance(实例),所以也称为实例对象。要创建一个对象,必须先有一个类,然后通过 new 关键字创建一个对象。对象创建的语法格式如下。

```
类名称 对象名称 = new 类名称();
```

接下来通过代码清单 3.1 创建两个 Student 对象,代码如下所示。

代码清单 3.1 Demo1Student

```
package com.yyds.unit3.demo;
public class Demo1Student {
    public static void main(String[] args) {
        //有点像创建变量,实际上 student1 这个对象名称就可以理解成变量
        Student student1 = new Student();
        //使用对象名.成员变量名操作变量
        student1.age = 18;
        student1.name = "张三";
        //使用对象名.成员方法名调用方法
        student1.study();
        //再创建一个对象
        Student student2 = new Student();
        student2.name = "李四";
        student2.age = 24;
        student2.eat("饼干");
    }
}
```

程序(代码清单 3.1)运行结果如图 3.1 所示。

注意:成员变量和成员方法隶属于对象,不同对象之间的成员变量占用不同的地址空间,互不影响。

```
张三年龄18岁，在学习Java
李四吃饼干
```

图 3.1 程序运行结果


扫一扫

3.2.3 成员变量默认值

为一个类定义成员变量时,可以显式地为其初始化。如果不为成员变量初始化,Java 虚拟机也会默认为成员变量初始化。表 3.1 是 Java 虚拟机为每种数据类型的成员变量赋

予的默认值。

表 3.1 成员变量默认值

数据类型	默认值	数据类型	默认值
整型(short、byte、int、long)	0	布尔型	false
浮点型(float、double)	0.0	引用类型	null
字符型	'\u0000'		

下面再编写代码创建一个 student 对象，并不给它的成员变量赋值，查看运行结果，如代码清单 3.2 所示。

代码清单 3.2 Demo2Student

```
package com.yyds.unit3.demo;
public class Demo2Student {
    public static void main(String[] args) {
        Student student = new Student();
        System.out.println("String 类型默认值: " + student.name);
        System.out.println("int 类型默认值: " + student.age);
    }
}
```

程序(代码清单 3.2)运行结果如图 3.2 所示。

```
String类型默认值：null
int类型默认值：0
```

图 3.2 程序运行结果

成员变量与第 2 章的局部变量不同。局部变量必须赋值才可以使用，而成员变量即使不赋值，Java 虚拟机也会默认为其赋值。

3.2.4 对象内存分析


扫一扫

接下来编写一个方法，该方法接收一个 Student 对象作为参数，为这个对象的年龄加 1。为了方便对比，再编写一个方法，接收 int 类型的参数，该方法的功能是给这个参数加 1，如代码清单 3.3 所示。

代码清单 3.3 Demo3Add

```
package com.yyds.unit3.demo;
public class Demo3Add {
    public static void main(String[] args) {
        Student student = new Student();
        student.age = 18;
        add(student);
        System.out.println("加 1 后 student 的年龄为: " + student.age);
        int num = 18;
        add(num);
        System.out.println("加 1 后的 num 值为: " + num);
    }
    public static void add(int num) {
```

```
            num++;
        }
        public static void add(Student student) {
            student.age++;
        }
    }
```

程序(代码清单 3.3)运行结果如图 3.3 所示。

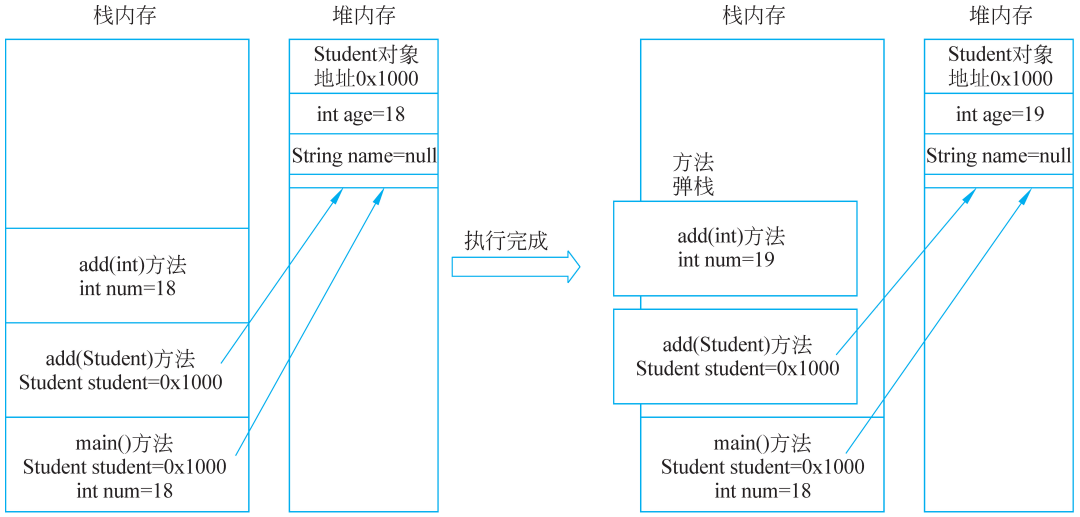
```
加1后student的年龄为：19
加1后的num值为：18
```

图 3.3　程序运行结果

为什么 Student 对象的 age 变量成功增加，而 num 却没有增加呢？如果熟悉第 2 章数组的内存分析，那么这里的问题就不难理解。原因是 Student 和 num 存储的位置不同。

num 是基本类型，存储在栈内存中，main()方法和 add()方法是两个不同的栈帧。由于 Java 只有值传递的特点，在调用 add()方法时虽然把 num 传递给了方法，但两个栈帧中的 num 却是完全不同的变量，因此在 add()方法中改变 num 的值并不会影响到 main()方法。

而 Student 是引用类型，存储在堆内存中，栈内存中只存储“值”，这个值是 Student 对象在堆内存中的地址。虽然 Java 只有值传递，main()方法和 add()方法两个栈帧中的 Student 变量是不同的变量，但它们的值相同。由于引用类型数据的特点，操作值最终都需要到堆内存中进行，因此在 add()方法中对堆内存中 age 的操作，在 main()方法中也能感知到，如图 3.4 所示。



main( )方法和add(Student)方法中都有Student变量，这两个是不同的变量，但是它们都指向堆内存中同一个地址：0x1000，因此，当在add( )方法中修改了age时，实际上是修改了堆内存中的age，在main( )方法中获取age时，获取的也是同一个堆内存地址的age。当main( )方法和add(int)方法操作的是基本数据类型时，因为基本数据类型不会存储到堆内存，所以尽管在add( )方法中修改了num的值，但在main( )方法中并不能获取到add()方法修改后的num值。

执行后，在add(Student)方法中，将堆内存中的age修改成19。在main( )方法中获取Student的age，获取到的值也会是19。而在add(int)方法中将num修改成19，在main( )方法中获取不到修改后的值，依然是18。

图 3.4　Java 对象在内存中的存储

这里还须注意，成员变量和局部变量的存储位置是不同的。尽管成员变量也可能是基本数据类型，但它的生命周期是跟随对象的，也会随着对象一并存储到堆内存中，而局部变量的基本数据类型只会存储到栈内存中。

3.2.5 匿名对象

通过使用 new 关键字创建对象，这种对象分为两种：匿名对象与非匿名对象。何为匿名对象，何为非匿名对象呢？匿名对象就是没有名字的对象，是定义对象的一种简写方式。

当方法的参数只需要一个对象，或者仅想调用一下某个对象的成员方法时，大可不必为这个对象单独创建一个变量，此时就可以使用匿名对象，如代码清单 3.4 所示。

代码清单 3.4　Demo4Student

```
package com.yyds.unit3.demo;
public class Demo4Student {
    public static void main(String[] args) {
        //直接传一个对象进去即可，并不需要创建变量接收
        invokeMethod(new Student());
    }
    public static void invokeMethod(Student student) {
        student.study();
        student.eat("饼干");
    }
}
```

程序（代码清单 3.4）运行结果如图 3.5 所示。

```
null年龄0岁，在学习Java
null吃饼干
```

图 3.5　程序运行结果

3.3 构造方法

3.3.1 什么是构造方法

构造方法也称作构造器（constructor），用于给对象进行初始化操作，即为对象成员变量赋初始值。构造方法的名称必须与类型相同，并且不能定义返回值，不能出现 return 关键字。构造方法的调用必须通过 new 关键字调用，语法格式如下所示。

```
修饰符　类名（形参列表）{ }
```

事实上，3.2 节说到对象创建方式时，提到的“new 类名()”是不准确的，准确的说法应该是使用 new 关键字调用它的构造方法。


扫一扫

3.3.2 构造方法的使用

Java 中要求每一个类必须有构造方法，当不在类中定义构造方法时，编译器会自动为类提供一个默认的无参数构造方法，一般简称为“无参构造方法”。前面通过 new 关键字创建 Student 对象时，就是在调用它默认的无参构造方法。可以在 out 文件夹中找到 Student.class 文件，在这里打开命令行执行 javap.\Student.class 命令，对字节码文件反编译，如图 3.6 所示。

```
PS F:\教材\code\out\production\code\com\yyds\unit3\demo> javap .\Student.class
Compiled from "Student.java"
public class com.yyds.unit3.demo.Student {
  java.lang.String name;
  int age;
  public com.yyds.unit3.demo.Student();
  void eat(java.lang.String);
  void study();
}
```

图 3.6 反编译 Student

可以看到，Student 类中有一个名为 Student 的方法，该方法就是编译器默认提供的空参构造方法。

如果手动为其提供了构造方法，那么编译器就不会再为该类提供默认构造方法了。接下来给 Student 分别提供无参构造方法和有参构造方法，代码如下所示。

```
package com.yyds.unit3.demo;
public class Student {
    //变量,此处也称为成员变量
    String name;
    int age;
    //无参构造方法
    public Student() {
        System.out.println("无参构造执行了");
    }
    //有参构造方法
    public Student(String stuName, int stuAge) {
        name = stuName;
        age = stuAge;
        System.out.println("有参构造执行了");
    }
    //方法,此处也称为成员方法
    void eat(String food) {
        System.out.println(name + "吃" + food);
    }
    void study() {
        System.out.println(name + "年龄" + age + "岁,在学习 Java");
    }
}
```

默认情况下，当系统没有为类提供任何一个构造方法时，只能使用系统默认的无参构造方法创建对象；当只为类提供了无参构造方法时，就只能使用无参构造方法创建对象了；当为类同时提供了无参构造方法和有参构造方法时，不但可以使用无参构造方法创建对象，还可以使用有参构造方法创建对象，如代码清单 3.5 所示。

代码清单 3.5　Demo5Constructor

```
package com.yyds.unit3.demo;
public class Demo5Constructor {
    public static void main(String[] args) {
        //无参构造方法创建对象
        Student student1 = new Student();
        //有参构造方法创建对象
        Student student2 = new Student("张三", 23);
    }
}
```

程序（代码清单 3.5）运行结果如图 3.7 所示。

```
无参构造执行了
有参构造执行了
```

图 3.7　程序运行结果

可以看到，new 关键字的作用不仅仅是创建对象，它还会调用到对应的构造方法，并执行该构造方法中的方法体。

3.3.3　构造方法的重载


扫一扫

在 Java 中，构造方法也可以重载。构造方法重载是方法重载中的一个典型特例。当创建一个对象时，JVM 会自动根据当前对方法的调用形式在类的定义中匹配形式相符合的构造方法，匹配成功后执行该构造方法。接下来给 Student 创建多个构造方法，如下所示。

```
package com.yyds.unit3.demo;
public class Student {
    String name;
    int age;
    public Student() {
        System.out.println("无参构造执行了");
    }
    public Student(String stuName, int stuAge) {
        name = stuName;
        age = stuAge;
        System.out.println("有参构造执行了");
    }
    public Student(int age) {
        this.age = age;
    }
    public Student(String name) {
```

```
            this.name = name;
        }
    }
```

由于构造方法的名称必须与类名相同，所以 Student 类中所有的构造方法名称都完全一样，但当使用 new 关键字调用构造方法时，就像调用普通的方法一样，Java 虚拟机（JVM）会根据传递的参数个数和参数类型决定调用哪个构造方法。

这里还需要注意一点，当局部变量和成员变量的变量名相同时，此时编译器无法区分，应当将其中一处的变量名进行改名或者显式使用 this 关键字调用成员变量。

3.4 this 关键字


扫一扫

3.4.1 this 关键字介绍

当创建一个对象成功后，JVM 会动态地分配一个引用，该引用指向的就是当前对象，该引用称作 this。更直白地说，this 关键字就是在成员方法或者构造方法中使用，用来调用当前对象的成员变量、成员方法或构造方法，它代表当前对象。

比如创建两个 Student 对象，分别为 stu1 和 stu2。当 stu1 调用 eat()方法时，eat()方法中的 this 就代表 stu1 这个对象；当 stu2 调用 eat()方法时，eat()方法中的 this 就代表 stu2 这个对象。


扫一扫

3.4.2 this 关键字的使用

this 关键字可以调用成员变量、成员方法、构造方法。需要注意的是，成员方法中不能使用 this 关键字调用构造方法，当使用 this 关键字调用构造方法时，它必须出现在构造方法的第一行。

接下来编写一个案例。定义一个坐标类（Point），用于表示二维空间中的一个坐标位置。通过坐标类的方法，实现计算两个坐标位置之间的距离。首先定义坐标类，如下所示。

```
package com.yyds.unit3.demo;
public class Point {
    double x;
    double y;
    public Point(double x, double y) {
        this.x = x;
        this.y = y;
    }
    public double calcLength(Point p) {
        double xLen = this.x - p.x;
        double yLen = this.y - p.y;
        return Math.sqrt(xLen * xLen+yLen * yLen);
    }
}
```

接下来编写测试程序，如代码清单 3.6 所示。

代码清单 3.6　Demo6This

```
package com.yyds.unit3.demo;
public class Demo6This {
    public static void main(String[] args) {
        Point p1 = new Point(3, 5);
        Point p2 = new Point(6, 1);
        double length = p1.calcLength(p2);
        System.out.println("两个坐标的距离为: " + length);
    }
}
```

程序（代码清单 3.6）运行结果如图 3.8 所示。

```
两个坐标的距离为: 5.0
```

图 3.8　程序运行结果

3.5　static 关键字

3.5.1　静态变量


扫一扫

在类中，将与成员变量同级的用 static 修饰的变量称为静态变量或类变量。静态变量优先于对象存在，随着类的加载就已经存在了，该类的所有实例共用这个静态变量，即共用同一份地址空间。当想调用静态变量时，可以使用对象名.变量名进行调用，但不推荐，建议使用类名.变量名进行调用。

接下来给 Student 类加上一个静态变量，如下所示。

```
package com.yyds.unit3.demo;
public class Student {
    String name;
    int age;
    static String grade;
    public Student() {
        System.out.println("无参构造执行了");
    }
    public Student(String stuName, int stuAge) {
        name = stuName;
        age = stuAge;
        System.out.println("有参构造执行了");
    }
    public Student(int age) {
        this.age = age;
    }
```

```
    public Student(String name) {
        this.name = name;
    }
    //方法,此处也称为成员方法
    void eat(String food) {
        System.out.println(name + "吃" + food);
    }
    void study() {
        System.out.println(name + "年龄" + age + "岁,在学习 Java");
    }
}
```

下面编写程序演示静态变量的使用,如代码清单 3.7 所示。

代码清单 3.7　Demo7Static

```
package com.yyds.unit3.demo;
public class Demo7Static {
    public static void main(String[] args) {
        Student s1 = new Student();
        Student s2 = new Student();
        s1.grade = "软件 1 班";
        System.out.println("s1 的 grade 值: " + s1.grade);
        System.out.println("s2 的 grade 值: " + s2.grade);
        System.out.println("使用类名取 grade 值: " + Student.grade);
    }
}
```

程序(代码清单 3.7)运行结果如图 3.9 所示。

```
无参构造执行了
无参构造执行了
s1的grade值：软件1班
s2的grade值：软件1班
使用类名取grade值：软件1班
```

图 3.9　程序运行结果

可以看到,程序中只给 s1 这个对象的 grade 进行了赋值,但实际上 s1、s2 使用类名获取 grade 变量,拿到的都是同一个值,这是因为静态变量是属于类的,全局仅此一份。

在开发中应尽可能避免将“姓名”“班级”这类变量定义成静态的,因为这类变量在逻辑上应当属于对象,每个对象的姓名和班级都可能是不同的。一般来说,static 关键字会与 final 关键字一起使用,用于定义全局的常量。


扫一扫

3.5.2　静态方法

static 关键字也可以修饰方法,用 static 修饰的方法称为静态方法或类方法。静态方法同样是属于类的,优先于对象存在,调用方式与静态变量相同,也是建议使用类名.方法名进行调用。

接下来在 Student 类中定义一个静态方法,如下所示。

```
package com.yyds.unit3.demo;
public class Student {
    //变量,此处也称为成员变量
    String name;
```

```
    int age;
    static String grade;
    public Student() {
        System.out.println("无参构造执行了");
    }
    public Student(String stuName, int stuAge) {
        name = stuName;
        age = stuAge;
        System.out.println("有参构造执行了");
    }
    public Student(int age) {
        this.age = age;
    }
    public Student(String name) {
        this.name = name;
    }
    //方法,此处也称为成员方法
    void eat(String food) {
        System.out.println(name + "吃" + food);
    }
    void study() {
        System.out.println(name + "年龄" + age + "岁,在学习 Java");
    }
    static void goHome() {
        //静态方法中不能调用成员变量和成员方法,不能使用 this
        //System.out.println(this.name + "回家");
        System.out.println("学生回家");
    }
}
```

下面同样编写程序进行演示，如代码清单 3.8 所示。

代码清单 3.8　Demo8Static

```
package com.yyds.unit3.demo;
public class Demo8Static {
    public static void main(String[] args) {
        Student s1 = new Student();
        Student s2 = new Student();
        s1.goHome();
        s2.goHome();
        Student.goHome();
    }
}
```

程序(代码清单 3.8)运行结果如图 3.10 所示。

```
无参构造执行了
无参构造执行了
学生回家
学生回家
学生回家
```

图 3.10　程序运行结果

与静态变量类似，静态方法虽然可以使用对象名调用，但依然不建议。此外，由于静态方法是属于类的，优先于对象存在，也就是说，当调用静态方法时可能程序并没有创建这个类的对象，因此在静态方法中不存在 this 引用，不能使用 this 关键字。

当一个类中几乎所有的核心方法都是静态方法时，通常称之为工具类。工具类中的方法都是工具性质的方法，它们的调用结果应当与对象无关，因此，对于工具类，一般使用 private 关键字修饰它的空参构造方法，让其他类无法创建工具类的对象。关于 private 修饰符，将在第 4 章介绍。


扫一扫

3.5.3 静态代码块

在类中，与成员变量和静态变量同级，使用大括号包裹起来的代码块称作构造代码块，当构造代码块使用 static 关键字修饰时，称作静态代码块。

构造代码块和静态代码块只能定义在类中，与成员变量和静态变量平级，并且可以定义多个。不同的是，构造代码块随着对象的创建而加载，每创建一个对象就会执行一次；而静态代码块随着类的加载而加载，每个类中的静态代码块只会执行一次。

当一个类中存在静态代码块、构造代码块、构造方法时，如果创建这个类的对象，它们三者的执行顺序应该是什么样的？

创建一个对象前，首先 JVM 会将这个类加载到方法区，当类被加载后，就会创建出它的静态变量，并执行静态代码块。之后，对象创建成功，初始化成员变量，并执行构造代码块。需要注意的是，对象其实并不是构造方法创建出来的，而是 new 关键字创建的。当创建了构造方法后，首先就会创建出对象，之后才会调用这个对象的构造方法。

分析完执行流程后，接下来编写一个简单的代码演示一下它们的执行流程，在此之间，在 Student 类中分别添加两个静态代码块和构造代码块，如下所示。

```
package com.yyds.unit3.demo;
public class Student {
    //省略其他代码
    {
        System.out.println("构造代码块 1 被执行了");
    }
    {
        System.out.println("构造代码块 2 被执行了");
    }
    static{
        System.out.println("静态代码块 1 被执行了");
    }
    static{
        System.out.println("静态代码块 2 被执行了");
    }
    public Student() {
        System.out.println("无参构造执行了");
    }
    //省略其他代码
}
```

接着编写代码进行测试，如代码清单 3.9 所示。

代码清单 3.9　Demo9Static

```
package com.yyds.unit3.demo;
public class Demo9Static {
    public static void main(String[] args) {
        Student s1 = new Student();
    }
}
```

程序（代码清单 3.9）运行结果如图 3.11 所示。

```
静态代码块1被执行了
静态代码块2被执行了
构造代码块1被执行了
构造代码块2被执行了
无参构造执行了
```

图 3.11　程序运行结果

静态代码块一般用于做一些全局的初始化，因为它随着类的加载只会执行一次，因此不用担心重复执行问题。比如需要开发一款游戏，那么游戏地图的加载就可以编写到静态代码块中。

到这里能够得出如下结论。

- 同一个类中，成员变量不能赋值给静态变量，静态变量可以赋值给成员变量和静态变量。
- 同一个类中，静态方法不能调用成员变量和成员方法，成员方法可以调用静态或非静态的方法和变量。
- 同一个类中，静态代码块不能调用成员变量和成员方法，构造代码块可以调用静态或非静态的方法和变量。

可能上面的总结有些枯燥难懂，其实只需记住一句话即可：带有 static 关键字的方法、变量、代码块只能调用带有 static 关键字的方法、变量，而不带有 static 关键字的可以随意调用。

3.6　包

3.6.1　包的概念

包是 Java 中重要的类管理方式，开发中会遇到大量同名的类，通过包给类加上一个命名空间，很容易解决类重名的问题，也可以实现对类的有效管理。包对于类，相当于文件夹对于文件的作用。

Java 通过 package 关键字声明一个包，之后根据功能模块将对应的类放到对应的包中即可。

包名的命名要求遵循标识符规则，在企业中一般以反着写企业域名的方式命名，如 com.baidu、com.jd，唯独需要注意的是，包名不能以 java 开头。比如下面就是声明一个包的语法格式。

```
package com.yyds.unit3.demo;
```

3.6.2 类的访问与导包

一般来说，定义的类都需要定义在包下。当要使用一个类时，这个类与当前程序在同一个包中，或者这个类是 java.lang 包中的类时通常可以省略掉包名，直接使用该类。其余情况下使用某一个类，必须导入包。使用 import 关键字导入 Java 中的包，语法格式如下。

```
import 包名.类名;
```

通过 import 关键字指定要导入哪个包下的哪个类，比如 import java.util.Scanner 就导入了 java.util 包下的 Scanner 类，而其他包中的 Scanner 类则不受影响。此外，前面使用到 String 类，而该类在 java.lang 包下，因此可以省略导包。而上面的案例中，Student 类与 Demo 类都在同一个包下，也可以省略导包。

当要使用到两个名称一样的类时，就需要以包名.类名的方式使用了，包名.类名的形式也称为一个类的全类名。

总体来说，包的使用比较简单，并且现在市面上成熟的开发工具都有很强大的自动导包功能，这里不再赘述。通过代码清单 3.10 演示一下包的使用即可，如下所示。

代码清单 3.10　Demo10Package

```
package com.yyds.unit3.demo;
//其他包下的类使用需要导包
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;
public class Demo10Package {
    public static void main(String[] args) {
        //java.lang 包下，省略导包
        String s = "HelloWorld";
        //Student 与本类在一个包下，省略导包
        Student student = new Student();
        //其他包下的类使用需要导包
        Scanner scanner = new Scanner(System.in);
        List list = new ArrayList();
        //如果使用到同名的两个类，其中一个必须用全类名的方式访问
        java.awt.List list1 = new java.awt.List();
    }
}
```

3.7 本章思政元素融入点

思政育人目标：把马克思主义立场观点方法的教育与科学精神的培养结合起来，培养学生的人文精神，启发学生领悟人生的哲学原理。

思想元素融入点：通过介绍面向对象程序设计中类和对象的概念和思想，引入“人们应当按照现实世界这个本来的面貌理解世界，直接通过对象及其相互关系反映世界”，引入马克思主义哲学之事物具有普遍联系性，要善于分析事物的具体联系，确立整体性、开放性观念，从动态中考察事物的普遍联系，抽取共性，甄别特性，认清事物的普遍联系与发展；同时，也可以联想到哲学中的观点“整体与局部的关系”——大到国家，小到班级，都是一个整体，每个人（即对象）都是整体（即类）的一部分（即一个实例），任何人都应该从整体的利益出发考虑问题，培养学生的全局意识。通过培养学生的人文精神，启发学生领悟人生的哲学原理，把马克思主义立场观点方法的教育与科学精神的培养结合起来，提高学生正确认识、分析和解决问题的能力。因为面向对象程序设计 Java 语言是运用人类的自然思维方式设计的，所以可将人的思维或人生追求引入 Java 语言中。

3.8 本章小结

本章为面向对象编程的基础阶段，对于初次接触面向对象的读者而言存在一些难度。面向对象开发相较于面向过程开发是一种思想上的转变，当熟悉了这种思想后，便不再有难度。

本章首先介绍了类和对象的概念，分别介绍了类的定义方式和对象的创建与使用，以步入面向对象编程的大门。接着，通过分析对象的内存，使读者理解基本类型和引用类型在 JVM 中存储方式上的差别。然后介绍了构造方法，使用户能够更加灵活地创建对象，并对对象中的成员变量进行更便捷的初始化，而当成员变量和局部变量存在重名时，成员变量需要加上 this 关键字，这一点需要注意。再者，介绍了 static 关键字的使用，分别演示了静态变量、静态方法、静态代码块与非静态变量、方法、代码块的区别。还概述了关键字 package 的思想以及介绍了 package 的使用。最后，指出了本章中的一些知识点可融入的思政元素。

通过本章的学习，读者能够掌握面向对象程序设计的基本思想，并且能够运用面向对象程序设计的思想解决一些实际问题。

3.9 习题

1. 什么是类？什么是对象？请举例说明。

2. 面向对象的三大特性是什么？

3. 使用计算器（Calculator）完成加法和减法运算，并能显示出该计算器的品牌和尺寸。

4. 编写一个狗类和猫类，它们都拥有姓名、年龄属性，以及吃饭和睡觉方法，并编写程序创建二者的对象，最后分别调用它们的方法。

5. 创建一个飞机类和汽车类，它们都拥有行驶（run）方法，编写程序分别调用它们的方法。

6. 在构造方法中通过 this 关键字调用另一个构造方法，如果使用 new 关键字创建该类的对象，此时会创建出几个对象？

7. 静态代码块有什么特点？它可用于什么场景？

8. 创建对象完全由构造方法完成，这句话是否正确？

9.（扩展习题）为什么说 Java 中只有值传递？请与 C、C++ 语言对比介绍。

10.（扩展习题）上面提到了方法区，你知道方法区存储什么数据吗？请查阅相关资料加以说明。