第 7 天 面向对象

今日内容介绍

- 面向对象
- 封装

复习:

成员变量与局部变量:

作用范围:成员变量作用域**整个类中**,**局部变量**作用域**整个函数**中或者**语句中.**

在内存中的位置:

成员变量在**堆内存中**,因为对象的存在才在堆内存中存在.

局部变量:存在**栈内存中**.

匿名对象:匿名对象的应用(和有名对象有什么不同)

匿名对象调用属性没有意义,调用方法比较稳妥.

匿名对象使用方式 **1:当对对象方法只调用一次时,可以用匿名对象完成,这样写比较简化.**

如果对一个对象进行多个成员调用,必须给这个对象起个名字.

匿名对象使用方式 **2:可以将匿名对象作为实际参数传递.参考匿名对象应用图片 1，2**

权限修饰符:private 私有权限修饰符,用于修饰类中的成员变量,成员函数,私有化仅在本类中有效,私有仅仅是封装的一种表现形式 图片:封装 1

之所以对外提供访问方式,就是因为可以在访问方式中添加逻辑判断语句,对访问数据进行操作,提高代码的健壮性.图片:封装 2

凡是看到 getXxx 或 setXxx 说明这个类里肯定有个私有的属性

封装原则:将不需要对外提供的内容全部隐藏起来,把属性都隐藏,提供公共方法对其访问

复习构造器:

对象一建立就会调用与之对应的构造函数,构造函数的作用用于给对象初始化,

当一个类中没有构造函数时候,那么系统会默认给该类加一个空参构造函数,在类中自定义了构造函数之后,默认的构造函数就没有了.

构造函数在写法上与运行上与一般函数都有不同:

构造函数是在对象一建立就运行,一般方法是对象调用才运行.

一个对象建立构造函数只运行一次,而一般方法可以被该对象调用多次.

那么问题来了?到底什么时候定义构造函数呢?当分析事物时,该事物具备一些特性或者行为,那么将这些内容定义在构造函数中.比如人的年龄和姓名,一出生就具备

图片:构造器思考?

构造代码块:

```
/*
构造代码块。
作用：给对象进行初始化。
对象一建立就运行，而且优先于构造函数执行。
和构造函数的区别：
构造代码块是给所有对象进行统一初始化，
而构造函数是给对应的对象初始化。

构造代码快中定义的是不同对象共性的初始化内容。

*/
```

1.1 局部变量和成员变量区别

理解清楚了类和对象之后，结合前 5 天的学习知识，发现在描述类的属性和前面学习定义变量差别不大，唯一区别就是位置发生了改变，那么类中定义的变量，和在方法定义的变量有啥差别呢？

回忆以前学习时变量的定义方式，和位置，以及现在定义类中属性的特点。总结下面几点异同

区别一：定义的位置不同

定义在类中的变量是成员变量

定义在方法中或者{}语句里面的变量是局部变量

区别二：在内存中的位置不同

成员变量存储在对内存的对象中

局部变量存储在栈内存的方法中

区别三：声明周期不同

成员变量随着对象的出现而出现在堆中，随着对象的消失而从堆中消失

局部变量随着方法的运行而出现在栈中，随着方法的弹栈而消失

区别四：初始化不同

成员变量因为在堆内存中，所有默认的初始化值

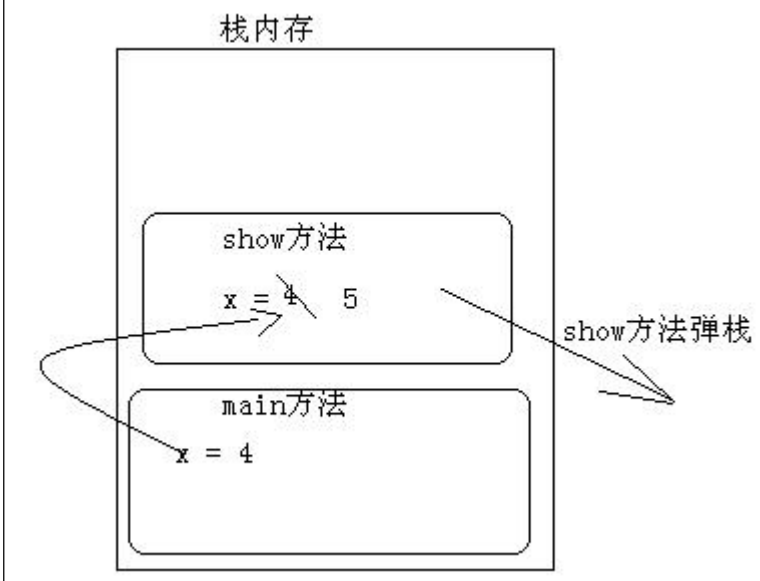
局部变量没有默认的初始化值，必须手动的给其赋值才可以使用。

1.2 基本类型和引用类型作为参数传递

引用类型数据和基本类型数据作为参数传递有没有差别呢？我们用如下代码进行说明，并配合图解让大家更加清晰

```
class Demo
{
    public static void main(String[] args)
    {
        int x = 4;
        show(x);
        System.out.println("x="+x);
    }
    public static void show(int x)
    {
        x = 5;

    }
}
```



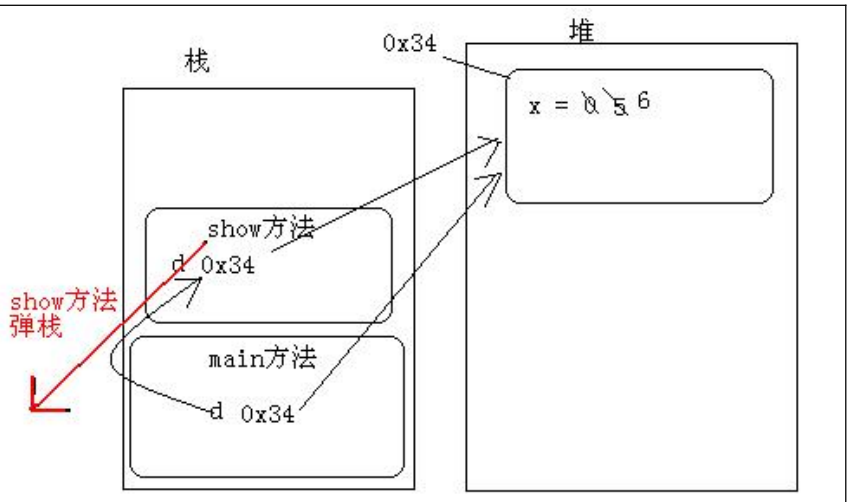
基本类型作为参数传递时，其实就是将基本类型变量 x 空间中的值复制了一份传递给调用的方法 show()，当在 show()方法中 x 接受到了复制的值，再在 show()方法中对 x 变量进行操作，这时只会影响到 show 中的 x。当 show 方法执行完成，弹栈后，程序又回到 main 方法执

行，main 方法中的 x 值还是原来的值。

```
class Demo
{
    int x ;
    public static void main(String[] args)
    {

        Demo d = new Demo();
        d.x = 5;
        show(d);

    System.out.println("x="+d.x);
    }
    public static void show(Demo d)
    {
        d.x = 6;
    }
}
```



当引用变量作为参数传递时，这时其实是将引用变量空间中的内存地址(引用)复制了一份传递给了 show 方法的 d 引用变量。这时会有两个引用同时指向堆中的同一个对象。当执行 show 方法中的 d.x=6 时，会根据 d 所持有的引用找到堆中的对象，并将其 x 属性的值改为 6.show

方法弹栈。

由于是两个引用指向同一个对象，不管是哪一个引用改变了引用的所指向的对象的中的值，其他引用再次使用都是改变后的值。

第 2 章 封装

2.1 封装概述

提起封装，大家并不陌生。前面我们学习方法时，就提起过，将具体功能封装到方法中，学习对象时，也提过将方法封装在类中，其实这些都是封装。

封装，它也是面向对象思想的特征之一。面向对象共有三个特征：封装，继承，多态。接下来我们具体学习封装。

- 封装表现：
  - 1、方法就是一个最基本封装体。
  - 2、类其实也是一个封装体。
- 从以上两点得出结论，封装的好处：
  - 1、提高了代码的复用性。
  - 2、隐藏了实现细节，还要对外提供可以访问的方式。便于调用者的使用。这是核心之一，也可以理解为就是封装的概念。
  - 3、提高了安全性。

2.2 封装举例

机箱：

一台电脑，它是由 CPU、主板、显卡、内存、硬盘、电源等部件组长，其实我们将这些部件组装在一起就可以使用电脑了，但是发现这些部件都散落在外面，很容造成不安全因素，于是，使用机箱壳子，把这些部件都装在里面，并在机箱壳上留下一些插口等，若不留插口，大家想想会是什么情况。

总结：机箱其实就是隐藏了办卡设备的细节，对外提供了插口以及开关等访问内部细节的方式。

2.3 私有 private

了解到封装在生活的体现之后，又要回到 Java 中，细说封装的在 Java 代码中的体现，先从描述 Person 说起。

描述人。Person

属性：年龄。

行为：说话：说出自己的年龄。

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

    public void show() {
        System.out.println("age=" + age + ",name" + name);
    }
}

public class PersonDemo {
    public static void main(String[] args) {
        // 创建 Person 对象
        Person p = new Person();
```

```
            p.age = -20; // 给 Person 对象赋值
            p.name = "人妖";
            p.show(); // 调用 Person 的 show 方法
        }
    }
```

通过上述代码发现，虽然我们用 Java 代码把 Person 描述清楚了，但有个严重的问题，就是 Person 中的属性的行为可以任意访问和使用。这明显不符合实际需求。

可是怎么才能不让访问呢？需要使用一个 Java 中的关键字也是一个修饰符 private(私有，权限修饰符)。只要将 Person 的属性和行为私有起来，这样就无法直接访问。

```
class Person {

    private int age;

    private String name;



    public void show() {

        System.out.println("age=" + age + ",name" + name);

    }

}
```

年龄已被私有，错误的值无法赋值，可是正确的值也赋值不了，这样还是不行，那肿么办呢？按照之前所学习的封装的原理，隐藏后，还需要提供访问方式。只要对外提供可以访问的方法，让其他程序访问这些方法。同时在方法中可以对数据进行验证。

一般对成员属性的访问动作：赋值(设置 set)，取值(获取 get)，因此对私有的变量访问的方式可以提供对应的 setXxx 或者 getXxx 的方法。

```
class Person {
    // 私有成员变量
    private int age;
    private String name;
```

```
    // 对外提供设置成员变量的方法
    public void setAge(int a) {
        // 由于是设置成员变量的值，这里可以加入数据的验证
        if (a < 0 || a > 130) {
            System.out.println(a + "不符合年龄的数据范围");
            return;
        }
        age = a;
    }

    // 对外提供访问成员变量的方法
    public void getAge() {
        return age;
    }
}
```

- 总结：

  类中不需要对外提供的内容都私有化，包括属性和方法。

  以后再描述事物，属性都私有化，并提供 setXxx getXxx 方法对其进行访问。

- 注意：私有仅仅是封装的体现形式而已。

2.4 this 关键字

2.4.1 成员变量和局部变量同名问题

当在方法中出现了局部变量和成员变量同名的时候，那么在方法中怎么区别局部变量成员变量呢？可以在成员变量名前面加上 this.来区别成员变量和局部变量

```
class Person {
    private int age;
    private String name;

    public void speak() {
        this.name = "小强";
        this.age = 18;
        System.out.println("name=" + this.name + ",age=" + this.age);
    }
}
```

```
class PersonDemo {
	public static void main(String[] args) {
		Person p = new Person();
		p.speak();
	}
}
```

2.4.2 对象的内存解释

我们已经学习了如何把生活中的事物使用 Java 代码描述，接下来我们分析对象在内存中的分配情况。这里需要画图一步一步演示，严格按照画图流程讲解内存对象创建使用过程。

```
class Person {
	private int age;
	public int getAge() {
		return this.age;
	}
	public void setAge(int age) {
		this.age = age;
	}
}
public class PersonDemo {
	public static void main(String[] args) {
		Person p = new Person();
		p.setAge(30);
		System.out.println("大家好，今年我" + p.getAge() + "岁");
	}
}
```

下图为程序中内存对象的创建使用过程。

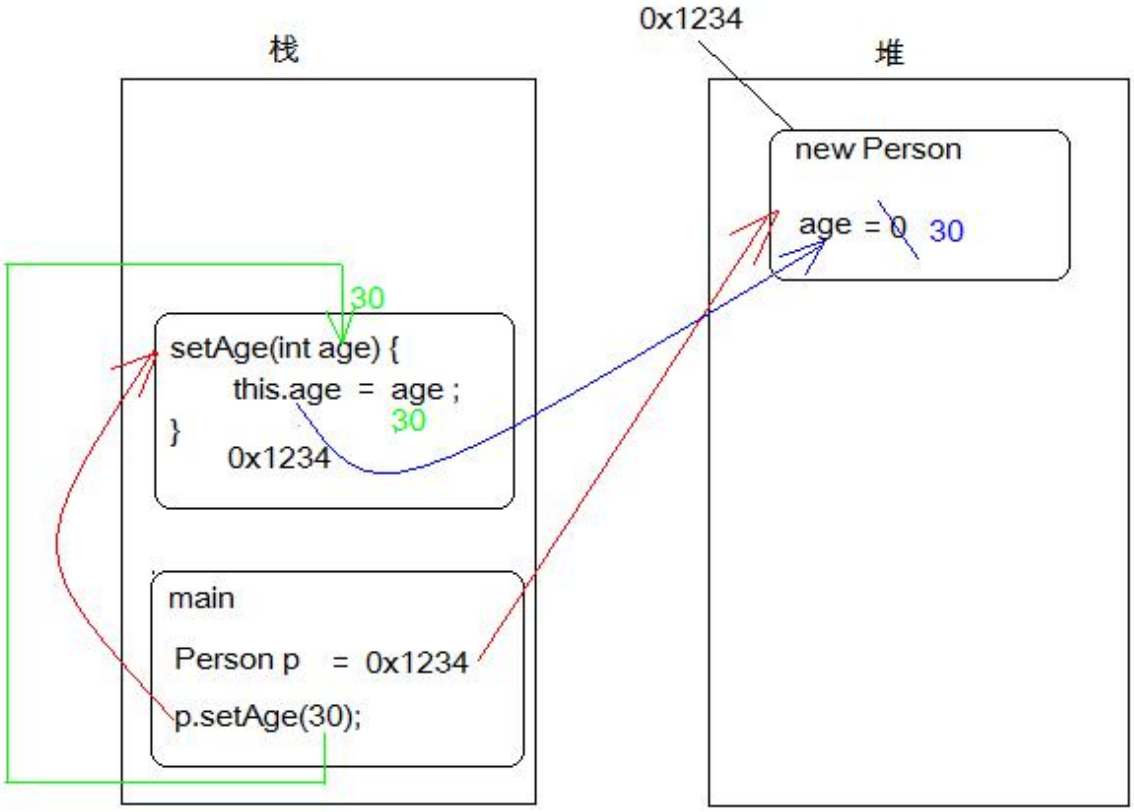


图 1-1 内存对象创建使用过程

程序执行流程说明：

1、先执行 main 方法（压栈），执行其中的 Person p = new Person()；

2、在堆内存中开辟空间，并为其分配内存地址 0x1234，紧接着成员变量默认初始化(age = 0)；将内存地址 0x1234 赋值给栈内中的 Person p 变量

3、继续执行 p.setAge(30)语句，这时会调用 setAge(int age)方法，将 30 赋值为 setAge 方法中的“age”变量；执行 this.age = age 语句，将 age 变量值 30 赋值给成员变量 this.age 为 30；

4、setAge()方法执行完毕后（弹栈），回到 main()方法，执行输出语句 System.out.println()，控制台打印 p 对象中的 age 年龄值。

- 注意：

- this 到底代表什么呢？this 代表的是对象 ,具体代表哪个对象呢？哪个对象调用了 this 所在的方法，this 就代表哪个对象。
- 上述代码中的 p.setAge(30)语句中 ,setAge(int age)方法中的 this 代表的就是 p 对象。

2.4.3 this 的应用

学习 this 的用法之后，现在做个小小的练习。

需求：在 Person 类中定义功能，判断两个人是否是同龄人

```
class Person {
    private int age;
    private String name;

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public void speak() {
        System.out.println("name=" + this.name + ",age=" + this.age);
    }

    // 判断是否为同龄人
    public boolean equalsAge(Person p) {
        // 使用当前调用该 equalsAge 方法对象的 age 和传递进来 p 的 age 进行比较
        // 由于无法确定具体是哪一个对象调用 equalsAge 方法，这里就可以使用 this 来代替
```

```
            /*
             * if(this.age == p.age) { return true; } return false;
             */
            return this.age == p.age;
        }
    }
```

第 3 章 继承

3.1 继承的概念

在现实生活中，继承一般指的是子女继承父辈的财产。在程序中，继承描述的是事物之间的所属关系，通过继承可以使多种事物之间形成一种关系体系。例如公司中的研发部员工和维护部员工都属于员工，程序中便可以描述为研发部员工和维护部员工继承自员工，同理，JavaEE 工程师和 Android 工程师继承自研发部员工，而维网络维护工程师和硬件维护工程师继承自维护部员工。这些员工之间会形成一个继承体系，具体如下图所示。

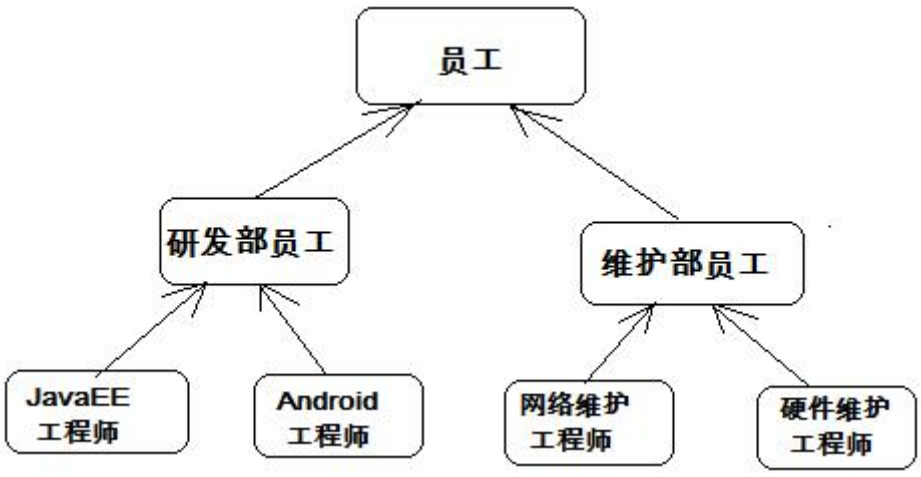


图 1-2 员工继承关系图

在 Java 中，类的继承是指在一个现有类的基础上去构建一个新的类，构建出来的新类被称作子类，现有类被称作父类，子类会自动拥有父类所有可继承的属性和方法。

3.2 继承的格式&使用

在程序中，如果想声明一个类继承另一个类，需要使用 extends 关键字。

格式：

```
class 子类 extends 父类 {}
```

接下来通过一个案例来学习子类是如何继承父类的，如下所示。Example01.java

```
/*
 * 定义员工类 Employee
 */
class Employee {
	String name; // 定义 name 属性
	// 定义员工的工作方法
	public void work() {
		System.out.println("尽心尽力地工作");
	}
}

/*
 * 定义研发部员工类 Developer 继承 员工类 Employee
 */
class Developer extends Employee {
	// 定义一个打印 name 的方法
	public void printName() {
		System.out.println("name=" + name);
	}
}

/*
 * 定义测试类
 */
public class Example01 {
	public static void main(String[] args) {
		Developer d = new Developer(); // 创建一个研发部员工类对象
		d.name = "小明"; // 为该员工类的 name 属性进行赋值
		d.printName(); // 调用该员工的 printName()方法
		d.work(); // 调用 Developer 类继承来的 work()方法
	}
}
```

运行结果如下图所示。

图 1-3 运行结果

在上述代码中，Developer 类通过 extends 关键字继承了 Employee 类，这样 Developer 类便是 Employee 类的子类。从运行结果不难看出，子类虽然没有定义 name 属性和 work() 方法，但是却能访问这两个成员。这就说明，子类在继承父类的时候，会自动拥有父类的成员。

3.3 继承的好处&注意事项

继承的好处：

1、继承的出现提高了代码的复用性，提高软件开发效率。

2、继承的出现让类与类之间产生了关系，提供了多态的前提。

在类的继承中，需要注意一些问题，具体如下：

- 1、在 Java 中，类只支持单继承，不允许多继承，也就是说一个类只能有一个直接父类，例如下面这种情况是不合法的。

```
class A{}
class B{}
class C extends A,B{}   // C 类不可以同时继承 A 类和 B 类
```

- 2、多个类可以继承一个父类，例如下面这种情况是允许的。

```
class A{}
class B extends A{}
class C extends A{}    //  类 B 和类 C 都可以继承类 A
```

- 3、在 Java 中，多层继承是可以的，即一个类的父类可以再去继承另外的父类，例如 C 类继承自 B 类，而 B 类又可以去继承 A 类，这时，C 类也可称作 A 类的子类。下面这种情况是允许的。

```
class A{}
```

```
class B extends A{}    // 类 B 继承类 A，类 B 是类 A 的子类
class C extends B{}    // 类 C 继承类 B，类 C 是类 B 的子类，同时也是类 A 的子类
```

- 4、在 Java 中，子类和父类是一种相对概念，也就是说一个类是某个类父类的同时，也可以是另一个类的子类。例如上面的这种情况中，B 类是 A 类的子类，同时又是 C 类的父类。

3.4 继承-子父类中成员变量的特点

了解了继承给我们带来的好处，提高了代码的复用性。继承让类与类或者说对象与对象之间产生了关系。那么，当继承出现后，类的成员之间产生了那些变化呢？

类的成员重点学习成员变量、成员方法的变化。

成员变量：如果子类父类中出现不同名的成员变量，这时的访问是没有任何问题。

看如下代码：

```
class Fu
{
	//Fu 中的成员变量。
	int num = 5;
}
class Zi extends Fu
{
	//Zi 中的成员变量
	int num2 = 6;
	//Zi 中的成员方法
	public void show()
	{
		//访问父类中的 num
		System.out.println("Fu num="+num);
		//访问子类中的 num2
		System.out.println("Zi num2="+num2);
	}
}
class Demo
{
	public static void main(String[] args)
	{
```

```
        Zi z = new Zi(); //创建子类对象
        z.show(); //调用子类中的 show 方法
    }
}
```

代码说明：Fu 类中的成员变量是非私有的，子类中可以直接访问，若 Fu 类中的成员变量私有了，子类是不能直接访问的。

当子父类中出现了同名成员变量时，在子类中若要访问父类中的成员变量，必须使用关键字 super 来完成。super 用来表示当前对象中包含的父类对象空间的引用。super 今天不做具体讲解，在课程第 12 天会详细讲解。

```
在子类中，访问父类中的成员变量格式：
    super.父类中的成员变量
```

看如下代码：

```
class Fu
{
    //Fu 中的成员变量。
    int num = 5;
}
class Zi extends Fu
{
    //Zi 中的成员变量
    int num = 6;
    void show()
    {
        //子父类中出现了同名的成员变量时
        //在子类中需要访问父类中非私有成员变量时，需要使用 super 关键字
        //访问父类中的 num
        System.out.println("Fu num="+super.num);
        //访问子类中的 num2
        System.out.println("Zi num2="+this.num);
    }
}
class Demo5
{
    public static void main(String[] args)
    {
        Zi z = new Zi(); //创建子类对象
```

```
            z.show(); //调用子类中的 show 方法
        }
    }
```

3.5 继承-子父类中成员方法特点-重写&应用

- 子父类中成员方法的特点

当在程序中通过对象调用方法时，会先在子类中查找有没有对应的方法，若子类中存在就会执行子类中的方法，若子类中不存在就会执行父类中相应的方法。

看如下代码：

```
class Fu{
    public void show(){
        System.out.println("Fu 类中的 show 方法执行");
    }
}
class Zi extends Fu{
    public void show2(){
        System.out.println("Zi 类中的 show2 方法执行");
    }
}
public   class Test{
    public static void main(String[] args) {
        Zi z = new Zi();
        z.show(); //子类中没有 show 方法，但是可以找到父类方法去执行
        z.show2();
    }
}
```

- 成员方法特殊情况——覆盖

子类中出现与父类一模一样的方法时，会出现覆盖操作，也称为 override 重写、复写或者覆盖。

```
class Fu
{
    public void show()
    {
```

```
            System.out.println("Fu show");
        }
}
class Zi extends Fu
{
        //子类复写了父类的 show 方法
        public void show()
        {
            System.out.println("Zi show");
        }
}
```

- 方法重写（覆盖）的应用：

当子类需要父类的功能，而功能主体子类有自己特有内容时，可以重写父类中的方法，这样，即沿袭了父类的功能，又定义了子类特有的内容。

举例：比如手机，当描述一个手机时，它具有发短信，打电话，显示来电号码功能，后期由于手机需要在来电显示功能中增加显示姓名和头像，这时可以重新定义一个类描述智能手机，并继承原有描述手机的类。并在新定义的类中覆盖来电显示功能，在其中增加显示姓名和头像功能。

```
在子类中，访问父类中的成员方法格式：
    super.父类中的成员方法();
```

看如下代码：

```
public class Test {
        public static void main(String[] args) {
            new NewPhone().showNum();
        }
}

//手机类
class Phone{
        public void sendMessage(){
            System.out.println("发短信");
        }
        public void call(){
            System.out.println("打电话");
```

```
        }
        public void showNum(){
            System.out.println("来电显示号码");
        }
    }

    //智能手机类
    class NewPhone extends Phone{

        //覆盖父类的来电显示号码功能，并增加自己的显示姓名和图片功能
        public void showNum(){
            //调用父类已经存在的功能使用 super
            super.showNum();
            //增加自己特有显示姓名和图片功能
            System.out.println("显示来电姓名");
            System.out.println("显示头像");
        }
    }
```

3.6 方法重写的注意事项

重写需要注意的细节问题：

- 子类方法覆盖父类方法，必须要保证权限大于等于父类权限。

```
class Fu(){
    void show(){}
    public void method(){}
}
class Zi() extends Fu{
    public void show(){}   //编译运行没问题
    void method(){}        //编译错误
}
```

- 写法上稍微注意:必须一模一样:方法的返回值类型 方法名 参数列表都要一样。

总结：当一个类是另一个类中的一种时，可以通过继承，来继承属性与功能。如果父类具备的功能内容需要子类特殊定义时，进行方法重写。

第 4 章 总结

4.1 知识点总结

- 类与对象
  - 类，用于描述多个对象的共同特征，它是对象的模板。
  - 对象，用于描述现实中的个体，它是类的实例。
  - 类的定义：使用关键字 class 来定义 java 中的类
    - 格式：

```
class 类名 {
    //属性
    数据类型 变量名;
    …
    //方法
    修饰符 返回值类型 方法名(参数){    }
    …
}
```

  - 创建对象：
    - 格式：

```
类名 对象名 = new 类名();
```

- 封装（private 关键字）
  - 封装，把对象的属性与方法的实现细节隐藏，仅对外提供一些公共的访问方式
  - 封装的体现：
    - 变量:使用 private 修饰，这就是变量的封装

- 方法:也是一种封装，封装了多条代码
- 类： 也是一种封装，封装了多个方法

- private 关键字，私有的意思
  - 它可以用来修饰类中的成员(成员变量，成员方法)
  - private 的特点：
    - private 修饰的成员只能在当前类中访问，其他类中无法直接访问

- this 关键字
  - this 关键字，本类对象的引用
    - this 是在方法中使用的，哪个对象调用了该方法，那么，this 就代表调用该方法的对象引用
    - this 什么时候存在的？当创建对象的时候，this 存在的
    - this 的作用：用来区别同名的成员变量与局部变量（this.成员变量）

```
public void setName(String name) {
    this.name = name;
}
```

- 继承：是指在一个现有类的基础上去构建一个新的类，构建出来的新类被称作子类，现有类被称作父类，子类会自动拥有父类所有
- 继承的好处：可继承的属性和方法。
  - 提高了代表的可维护性

    提高了代码的复用性

    让类与类之间产生了继承关系
  - 继承的弊端：

类与类之间的耦合度过高

- 继承特点：

java 中类只能够单继承，不能多继承，可以多层继承

```
class Yy extends Object {}

class Fu extends Yy{}

class Zi extends Fu {}
```

所有的类都直接或者间接的继承了 Object 类，Object 类称为祖宗类

- 继承的注意事项：

1，使用关键字 extends 让类与类之间 产生继承关系

2，父类私有的成员，子类不能继承，因为根本看不到

3，不能为了继承某个功能而随意进行继承操作， 必须要符合 is a 的关系

苹果 is a 水果

男人 is a 人

狗 is a 人 ， 这种情况就不能继承了

- 继承中的成员变量关系：

不同名的变量：

子类直接继承使用

同名的变量：

默认访问的是子类自己的成员变量，想访问父类中的同名变量 ,请使用 super.成员变量;

- 继承中的成员方法关系：

不同名的方法：

子类直接继承使用

同名的方法：

默认访问的是子类自己的成员方法，想访问父类中的同名方法，请使用 super.成员方法();

- super:用来表示当前对象中包含的父类对象空间的引用

调用父类的成员变量：

super.成员变量;

调用方法的成员方法:

super.成员方法();