

第一章 继承

多个类中存在相同属性和行为时，将这些内容抽取到单独一个类中，那么多个类就无需再定义这些属性和行为，只要**继承**那一个类即可，其中多个类可以称为**子类**，单独那一个类称为**父类**、**超类** (superclass) 或者**基类**

- **继承**：就是子类继承父类的**属性和行为**，使得子类对象具有与父类相同的属性、相同的行为。子类可以直接访问父类中的**非私有**的属性和行为
- 继承的优点
 - 提高了**代码的复用性**
 - 类与类之间产生了关系，是**多态的前提**

1.1 继承的格式

通过 `extends` 关键字，可以声明一个子类继承另外一个父类

```
class 父类 {  
    ...  
}  
class 子类 extends 父类 {  
    ...  
}
```

1.2 继承后的成员变量

1.2.1 成员变量不重名

如果子类父类中出现**不重名**的成员变量，这时的访问时**没有影响**的

```
class Fu {  
    //父类中的成员变量  
    int num1 = 5;  
}  
  
public class Zi extends Fu {  
    //子类中的成员变量  
    int num2 = 6;  
    //子类中的成员方法  
    public void show() {  
        //访问父类中的num1  
        System.out.println("Fu num1 =" + num1);  
        //访问子类中的num2  
        System.out.println("Zi num2 =" + num2);  
    }  
}  
  
public class DemoExtends {  
    public static void main(String[] args) {  
        //创建子类对象  
        Zi z = new Zi();  
    }  
}
```

```

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

```

```

DemoExtends x
"D:\Professional Program Files\jdk-14\bin\java.exe" "-javaagent:D:\Professional Program Files\JetBrains\Intel
Fu num1 =5
Zi num2 =6

进程已结束，退出代码 0

```

1.2.2 成员变量重名

如果子类父类中出现**重名**的成员变量，这时的访问是**有影响的**

```

public class Fu {
    //父类中的成员变量
    int num = 5;
}

public class Zi extends Fu {
    //子类中的成员变量
    int num = 6;
    //子类中的成员方法
    public void show() {
        //访问父类中的num
        System.out.println("Fu num =" + num);
        //访问子类中的num
        System.out.println("Zi num =" + num);
    }
}

public class DemoExtends {
    public static void main(String[] args) {
        //创建子类对象
        Zi z = new Zi();
        //调用子类中的show方法
        z.show();
    }
}

```

```

DemoExtends x
"D:\Professional Program Files\jdk-14\bin\java.exe" "-javaagent:D:\Professional Program Files\JetBrains\Intel
Fu num =6
Zi num =6

进程已结束，退出代码 0

```

- 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，修饰父类成员变量，类似之前学过的 `this`

使用格式：

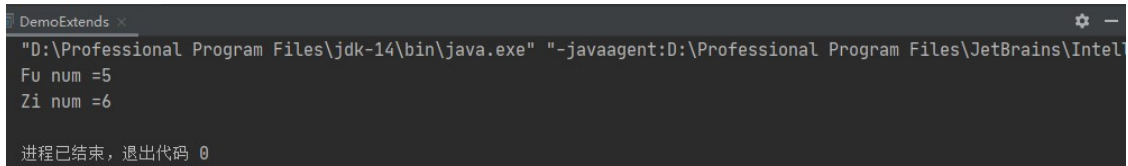
```
super.父类成员变量名
```

子类方法需要修改，代码如下

```

public class Zi extends Fu {
    //子类中的成员变量
    int num = 6;
    //子类中的成员方法
    public void show() {
        //访问父类中的num
        System.out.println("Fu num =" + super.num);
        //访问子类中的num
        System.out.println("Zi num =" + num);
    }
}

```



```

DemoExtends x
"D:\Professional Program Files\jdk-14\bin\java.exe" "-javaagent:D:\Professional Program Files\JetBrains\Intel
Fu num =5
Zi num =6
进程已结束，退出代码 0

```

- 父类中的成员变量是非私有的，子类中可以直接访问。若父类中的成员变量私有了，子类是不能直接访问的。通常编码时，遵循封装的原则，使用 `private` 修饰成员变量，在父类中提供公共的 `getter` 和 `setter` 方法用以访问父类的私有成员变量

1.3 继承后的成员方法

1.3.1 成员方法不重名

如果子类父类中出现**不重名**的成员方法，这时的调用是**没有影响**的。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对应的方法

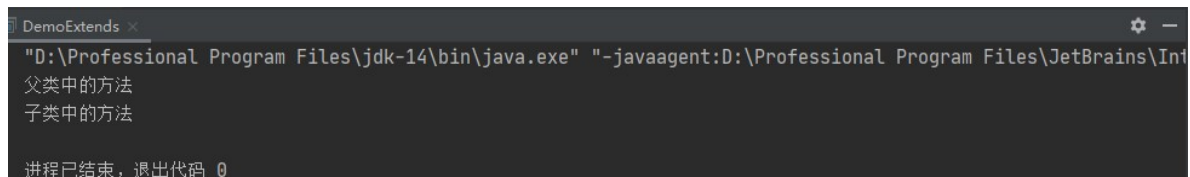
```

public class Fu {
    public void show() {
        System.out.println("父类中的方法");
    }
}

public class Zi extends Fu {
    public void show2() {
        System.out.println("子类中的方法");
    }
}

public class DemoExtends {
    public static void main(String[] args) {
        Zi z = new Zi();
        z.show();
        z.show2();
    }
}

```



```

DemoExtends x
"D:\Professional Program Files\jdk-14\bin\java.exe" "-javaagent:D:\Professional Program Files\JetBrains\Int
父类中的方法
子类中的方法
进程已结束，退出代码 0

```

1.3.2 成员方法重名——重写 (Override)

如果子类父类中出现**重名**的成员方法，此时的访问是一种特殊情况，叫做**方法重写**（`override`）

- **方法重写**：子类中出现与父类一模一样的方法时（返回值类型，方法名和参数列表都相同），会出现覆盖效果，也称为重写或者复写。**声明不变，重新实现。**

```
public class Fu {
    public void show() {
        System.out.println("父类中的方法");
    }
}

public class Zi extends Fu {
    public void show() {
        System.out.println("子类中的方法");
    }
}

public class DemoExtends {
    public static void main(String[] args) {
        Zi z = new Zi();
        z.show();
    }
}
```

```
DemoExtends x
"D:\Professional Program Files\jdk-14\bin\java.exe" "-javaagent:D:\Professional Program Files\JetBrains\IntelliJ IDEA\bin\jetbrains-agent.jar" -jar D:\Professional Program Files\JetBrains\IntelliJ IDEA\bin\jetbrains-agent.jar
子类中的方法

进程已结束，退出代码 0
```

- 重写的应用

子类可以根据需要，定义特定于自己的行为，即学习了父类的功能名称，又根据子类的需要，重新实现父类方法，从而进行扩展增强

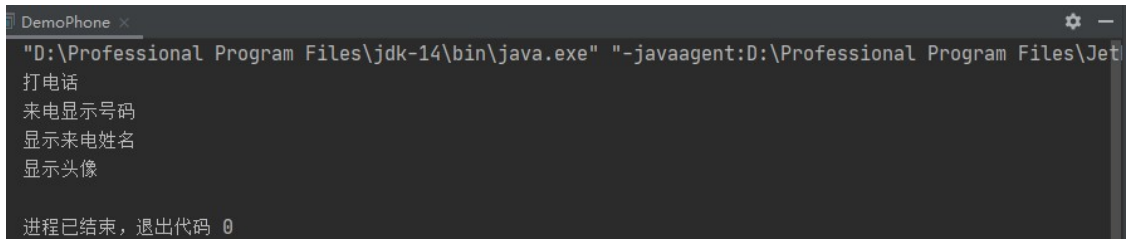
```
public class Phone {
    public void call() {
        System.out.println("打电话");
    }
    public void showNum() {
        System.out.println("来电显示号码");
    }
}

//智能手机类
public class NewPhone extends Phone {
    //重写父类的来电显示号码功能，并增加自己的显示姓名和头像功能
    @Override
    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头像功能
        System.out.println("显示来电姓名");
        System.out.println("显示头像");
    }
}
```

```

public class DemoPhone {
    public static void main(String[] args) {
        NewPhone phone = new NewPhone();
        //调用父类继承而来的方法
        phone.call();
        //调用子类重写的方法
        phone.showNum();
    }
}

```



```

DemoPhone x
"D:\Professional Program Files\jdk-14\bin\java.exe" "-javaagent:D:\Professional Program Files\JetI
打电话
来电显示号码
显示来电姓名
显示头像

进程已结束，退出代码 0

```

- 这里重写时，用到 `super.父类成员方法`，表示调用父类的成员方法
- 注意事项
 - 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
 - 子类方法覆盖父类方法，返回值类型、函数名和参数列表都要一模一样

1.4 继承后的构造方法

- 构造方法的名字是与类名一致的，所以子类是无法继承父类构造方法的
- 构造方法的作用是初始化成员变量的，所以子类的初始化过程中，必须先执行父类的初始化动作。子类的构造方法中默认有一个 `super()`，表示调用父类的构造方法，父类成员变量初始化后，可以给子类使用

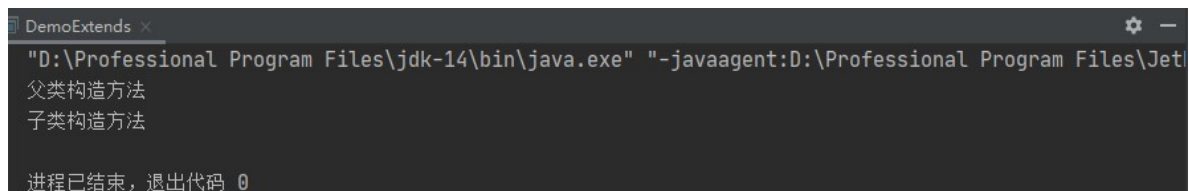
```

public class Fu {
    public Fu() {
        System.out.println("父类构造方法");
    }
}

public class Zi extends Fu {
    public Zi() {
        //调用父类构造方法
        super();
        System.out.println("子类构造方法");
    }
}

public class DemoExtends {
    public static void main(String[] args) {
        Zi z = new Zi();
    }
}

```



```

DemoExtends x
"D:\Professional Program Files\jdk-14\bin\java.exe" "-javaagent:D:\Professional Program Files\JetI
父类构造方法
子类构造方法

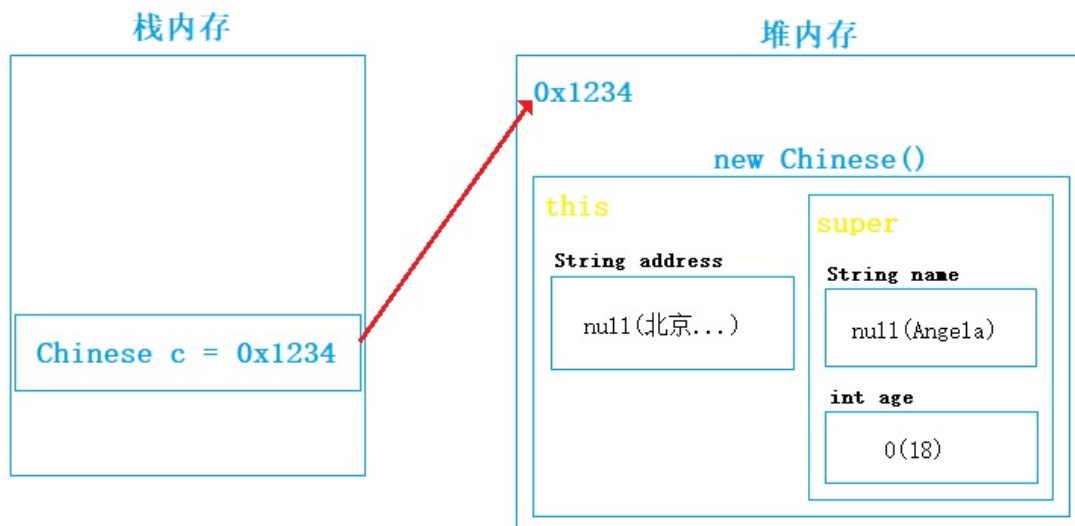
进程已结束，退出代码 0

```

1.5 super和this

1.5.1 父类空间优先于子类对象的产生

在每次创建子类对象时，先初始化父类空间，再创建其子类对象本身。目的在于子类对象中包含了其对应的父类空间，便可以包含其父类的成员，如果父类成员非 `private` 修饰，则子类可以随意使用父类成员。代码体现在子类的构造方法调用时，一定要先调用父类的构造方法



1.5.2 super和this的含义

- `super`：代表父类的**存储空间标识**（可以理解为父类的引用）
- `this`：代表**当前对象的引用**（谁调用就代表谁）

1.5.3 super和this的用法

- 访问成员

```
this.成员变量; //本类的
super.成员变量; //父类的

this.成员方法名(); //本类的
super.成员方法名(); //父类的
```

例：

```
public class Animal {
    public void eat() {
        System.out.println("Animal:eat");
    }
}

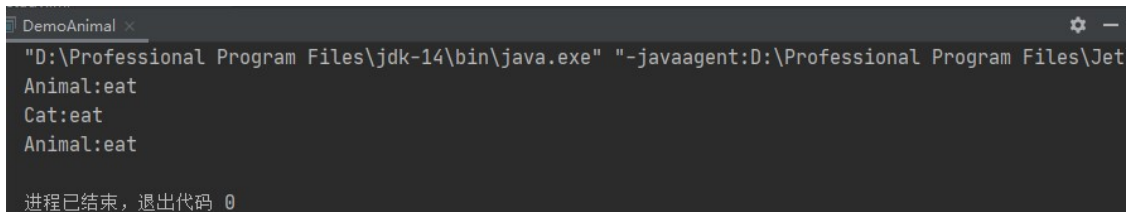
public class Cat extends Animal {
    @Override
    public void eat() {
        System.out.println("Cat:eat");
    }
    public void eatTest() {
```

```

        this.eat();
        super.eat();
    }
}

public class DemoAnimal {
    public static void main(String[] args) {
        Animal a = new Animal();
        a.eat();
        Cat c = new Cat();
        c.eatTest();
    }
}

```



- 访问构造方法

```

this(...); //本类的构造方法
super(...); //父类的构造方法

```

子类的每个构造方法中均有默认的 `super()`，调用父类的空参构造。手动调用父类构造会覆盖默认的 `super()`。

`super()` 和 `this()` 都必须是在构造方法的第一行，所以不能同时出现

1.6 继承的特点

1. Java只支持单继承，不支持多继承
2. Java支持多层继承（继承体系）
3. 子类和父类是一种相对的概念

第二章 抽象类

父类中的方法，被它的子类们重写，子类各自实现都不尽相同。那么父类的方法声明和方法主体，只有声明还有意义，而方法主体则没有存在的意义了。我们把没有方法主体的方法称为**抽象方法**。Java语法规定，包含抽象方法的类就是**抽象类**。

- **抽象方法**：没有方法体的方法
- **抽象类**：包含抽象方法的类

2.1 abstract 使用格式

2.1.1 抽象方法

使用 `abstract` 关键字修饰方法，该方法就成了抽象方法，抽象方法只包含一个方法名，而没有方法体定义格式：

修饰符 **abstract** 返回值类型 **方法名**(参数列表);

例:

```
public static void method();
```

2.1.2 抽象类

如果一个类包含抽象方法，那么该类必须是抽象类

定义格式:

```
abstract class 类名 {  
  
}
```

例:

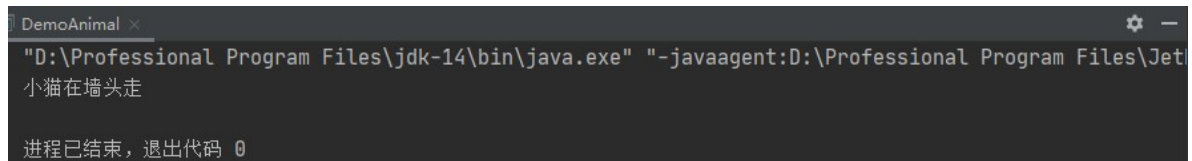
```
public abstract class Animal {  
    public abstract void run();  
}
```

2.1.3 抽象的使用

继承抽象类的子类**必须重写父类所有的抽象方法**，否则，该子类也必须声明为抽象类。最终，必须有子类实现该父类的抽象方法，否则，从最初的父类到最终子类都不能创建对象，失去意义

例:

```
public class Cat extends Animal {  
    @Override  
    public void run() {  
        System.out.println("小猫在墙头走");  
    }  
}  
  
public class CatTest {  
    public static void main(String[] args) {  
        Cat c = new Cat();  
        c.run();  
    }  
}
```



- 此时的方法重写，是子类对父类抽象方法的完成实现，我们将这种方法重写的操作，也叫做**实现方法**

2.2 注意事项

1. 抽象类**不能创建对象**，如果创建，编译无法通过而报错，只能创建其非抽象子类的对象

假设创建了抽象类的对象，调用抽象的方法，而抽象方法没有具体的方法体，没有意义

2. 抽象类中，可以有构造方法，是提供子类创建对象时，初始化父类成员使用的

子类的构造方法中，有默认的 `super()`，需要访问父类构造方法

3. 抽象类中，不一定包含抽象方法，但是有抽象方法的类必定是抽象类

未包含抽象方法的抽象类，目的就是不想让调用者创建该类对象，通常用于某些特殊的类结构设计

4. 抽象类的子类，必须重写抽象父类中**所有的**抽象方法，否则，编译无法通过而报错，除非该子类也是抽象类

假设不重写所有抽象方法，则类中可能包含抽象方法。那么创建对象后，调用抽象的方法没有意义

第三章 继承的综合案例

3.1 综合案例：群主发红包

某群有多名成员，群主给群员发普通红包，普通红包的规则：

1. 群主的一笔金额从群主的余额中扣除，平均分成n等份，让成员领取
2. 成员领取红包后，保存到成员余额中

请根据描述，完成案例中所有的定义以及制定类之间的继承关系，并完成发红包的操作

3.2 案例实现

定义用户类：

```
public class User {
    private String name; //用户名
    private double leftmoney; //账户金额

    public User() {
    }

    public User(String name, double leftmoney) {
        this.name = name;
        this.leftmoney = leftmoney;
    }

    public String getName() {
        return name;
    }

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

    public double getLeftmoney() {
        return leftmoney;
    }

    public void setLeftmoney(double leftmoney) {
```

```

        this.leftmoney = leftmoney;
    }

    //显示用户信息
    public void show() {
        System.out.print("用户名: " + name);
        System.out.println("余额为: " + String.format("%.3f", leftmoney));
    }
}

```

定义群主类:

```

public class Owner extends User {
    public Owner() {
    }

    public Owner(String name, double leftmoney) {
        super(name, leftmoney);
    }

    //发红包方法
    public ArrayList<Double> send(int money, int count){
        //读取群主账户金额信息
        double leftmoney = getLeftmoney();
        if(leftmoney < money)
            return null;

        //群主账户扣钱
        leftmoney -= money;
        setLeftmoney(leftmoney);

        //设立红包金额列表
        ArrayList<Double> list = new ArrayList<>();
        money = 100 * money;
        for (int i = 0; i < count - 1; i++)
            list.add(money/count/100.0);
        //无法整除的余数, 放到最后一位
        list.add(money/count/100.0 + money%count/100.0);

        return list;
    }

    //收回未领取红包
    public void backMoney(ArrayList<Double> list) {
        int backmoney = 0;
        while(list.size() != 0)
            backmoney += list.remove(0);
        setLeftmoney(backmoney + getLeftmoney());
    }
}

```

定义群员类:

```

public class Member extends User {
    public Member() {
    }
}

```

```

    }

    public Member(String name, double leftmoney) {
        super(name, leftmoney);
    }

    public void openRedPackage(ArrayList<Double> list) {
        System.out.print(getName() + "正在取抽红包: ");

        //读取用户账户金额信息
        double leftmonef = getLeftmoney();

        //红包抽完处理
        if(list.size() == 0){
            System.out.println("抽取失败, 红包已抽完!!!");
            return;
        }

        //随机抽取红包列表
        int index = new Random().nextInt(list.size());
        leftmonef += list.remove(index);
        setLeftmoney(leftmonef);
        System.out.println("抽取成功!!!");
    }
}

```

定义测试类:

```

public class Demo {
    public static void main(String[] args) {
        //创建群主, 群员
        Owner o = new Owner("张三", 200);
        Member m1 = new Member("李四", 10);
        Member m2 = new Member("王二", 15);
        Member m3 = new Member("麻子", 26);
        Member m4 = new Member("井九", 108);
        //创建键盘输入
        Scanner sc = new Scanner(System.in);
        System.out.print("请输入红包金额: ");
        int money = sc.nextInt();
        System.out.print("请输入红包个数: ");
        int count = sc.nextInt();
        //发红包
        ArrayList<Double> list = o.send(money, count);
        //余额不足判断
        if(list == null) {
            System.out.println("余额不足...");
            return;
        }
        System.out.println("-----");
        //打开红包, 返回未领取红包
        m1.openRedPackage(list);
        m2.openRedPackage(list);
        m3.openRedPackage(list);
        m4.openRedPackage(list);
        o.backMoney(list);
        System.out.println("-----");
    }
}

```

```
//展示信息
o.show();
m1.show();
m2.show();
m3.show();
m4.show();

}

}
```

运行结果：

```
Demo x
"D:\Professional Program Files\jdk-14\bin\java.exe" "-javaagent:D:\Professional Program Files\JetBrains\IntelliJ IDEA 2019.3.3\lib\idea_rt.
请输入红包金额: 100
请输入红包个数: 2
-----
李四正在取抽红包：抽取成功!!!
王二正在取抽红包：抽取成功!!!
麻子正在取抽红包：抽取失败，红包已抽完!!!
井九正在取抽红包：抽取失败，红包已抽完!!!
-----
用户名：张三余额为：100.000
用户名：李四余额为：60.000
用户名：王二余额为：65.000
用户名：麻子余额为：26.000
用户名：井九余额为：108.000

进程已结束，退出代码 0
```

```
Demo x
"D:\Professional Program Files\jdk-14\bin\java.exe" "-javaagent:D:\Professional Program Files\JetBrains\IntelliJ IDEA 2019.3.3\lib\idea_rt.
请输入红包金额: 100
请输入红包个数: 33
-----
李四正在取抽红包：抽取成功!!!
王二正在取抽红包：抽取成功!!!
麻子正在取抽红包：抽取成功!!!
井九正在取抽红包：抽取成功!!!
-----
用户名：张三余额为：187.889
用户名：李四余额为：13.030
用户名：王二余额为：18.030
用户名：麻子余额为：29.030
用户名：井九余额为：111.030

进程已结束，退出代码 0
```

- 注：账户金额数据在有余数的情况下存在精度问题，暂时未解决