day09【继承、super、this、抽象类】

今日内容

- 三大特性——继承
- 方法重写
- super关键字
- this关键字
- 抽象类

教学目标

- [ ] 能够解释类名作为参数和返回值类型
  - [ ] 类名作为方法的参数 传递的是地址值
  - [ ] 类名作为返回值类型 返回的是地址值
- [ ] 能够写出类的继承格式
- [ ] 能够说出继承的特点
- [ ] 能够说出子类调用父类的成员特点
- [ ] 能够说出方法重写的概念
- [ ] 能够说出super可以解决的问题
- [ ] 描述抽象方法的概念
- [ ] 写出抽象类的格式
- [ ] 写出抽象方法的格式
- [ ] 能够说出父类抽象方法的存在意义
- [ ] 能够完成发红包案例的代码逻辑

第一章 继承

1.1 概述

目标:

- 继承的概述

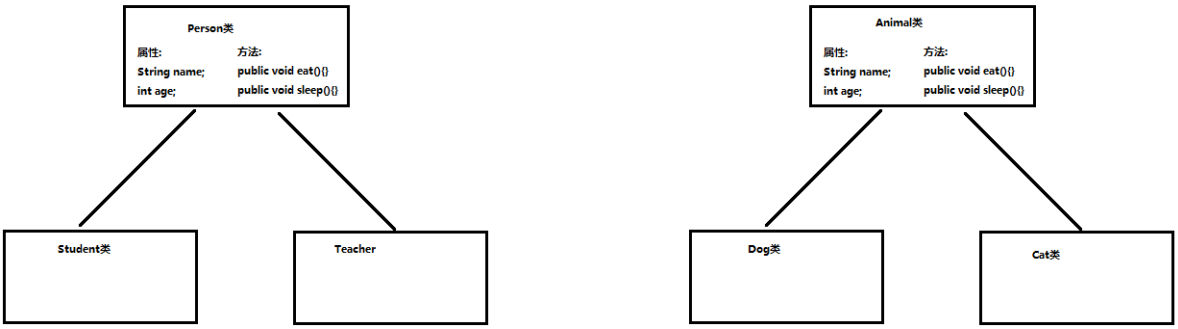
步骤:

- 继承的由来
- 继承的概述
- 继承的好处

讲解:

由来

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，那么多个类无需再定义这些属性和行为，只要继承那一个类即可。如图所示：



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**。

继承描述的是事物之间的所属关系，这种关系是：`is-a` 的关系。例如，图中学生属于人，老师属于人。可见，父类更通用，子类更具体。我们通过继承，可以使多种事物之间形成一种关系体系。

定义

- **继承**：就是子类继承父类的**属性**和**方法**，使得子类对象具有与父类相同的属性、相同的方法。子类可以直接访问父类中的**非私有**的属性和非私有方法。

好处

1. 提高**代码的复用性**。
2. 类与类之间产生了关系，是**多态的前提**。

小结:

- 继承的概述: 继承就是子类继承父类的属性和方法
- 继承之后子类就具有和父类相同的属性和方法,所以子类可以直接访问父类的非私有属性和非私有方法

1.2 继承的格式

目标:

- 如何实现继承呢??

步骤:

- 继承的格式

讲解:

通过 `extends` 关键字，可以声明一个子类继承另外一个父类，定义格式如下：

```
class 父类 {
    ...
}

class 子类 extends 父类 {
    ...
}
```

继承演示，代码如下：

```
/*
 * 定义员工类Employee，做为父类
 */
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");
    }
}

/*
 * 定义讲师类Teacher 继承 员工类Employee
 */
class Teacher 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 ExtendDemo01 {
    public static void main(String[] args) {
        // 创建一个讲师类对象
        Teacher t = new Teacher();

        // 为该员工类的name属性进行赋值
        t.name = "小明";

        // 调用该员工的printName()方法
        t.printName(); // name = 小明

        // 调用Teacher类继承来的work()方法
        t.work();  // 尽心尽力地工作
    }
}
```

小结:

- 继承的格式

1.3 继承后的特点——成员变量

目标:

- 当类之间产生了关系后，其中各类中的成员变量，又产生了哪些影响呢？

步骤:

- 父子类中的成员变量不重名
- 父子类中的成员变量重名

讲解:

成员变量不重名

如果子类父类中出现**不重名**的成员变量，这时的访问是**没有影响的**。代码如下：

```
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); // 继承而来，所以直接访问。
        // 访问子类中的num2
        System.out.println("Zi num2="+num2);
    }
}
class ExtendDemo02 {
    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();
    }
}

演示结果：
Fu num = 5
Zi num2 = 6
```

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**。代码如下：

```
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;
    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);
    }
}
class ExtendsDemo03 {
    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();
    }
}
```

演示结果：

```
Fu num = 6
Zi num = 6
```

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，类似于之前学过的`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：

```
class Zi extends Fu {
    // Zi中的成员变量
    int num = 6;
    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=" + this.num);
    }
}
演示结果：
Fu num = 5
Zi num = 6
```

> 小贴士：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。通常编码时，我们遵循封装的原则，使用private修饰成员变量，那么如何访问父类的私有成员变量呢？对！可以在父类中提供公共的getXxx方法和setXxx方法。

小结:

```
继承之后的特点----->成员变量
    父子类成员变量不重名:无任何影响
    父子类成员变量重名:遵守就近原则
        就近原则:首先先在局部位置查找,如果找到了就直接使用,如果没有找到,
        就去子类成员位置查找,如果子类成员位置有就直接使用,如果没有找到,
        就去父类的成员位置查找,如果找到了就直接使用,如果找不到,就报错

    如果在子类的方法中需要同时访问父子类同名的成员变量:
        访问父类: super.成员变量名
        访问子类: this.成员变量名

    如果父类中的成员变量使用了private修饰,那么想要访问父类的成员变量就得使用set和get方法
```

1.4 继承后的特点——成员方法

目标:

- 当类之间产生了关系，其中各类中的成员方法，又产生了哪些影响呢？

步骤:

- 成员方法不重名
- 成员方法重名

讲解:

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相应的方法。代码如下：

```
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 ExtendsDemo04{
    public static void main(String[] args) {
        Zi z = new Zi();
        //子类中没有show方法，但是可以找到父类方法去执行
        z.show();
        z.show2();
    }
}
```

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，这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**。

代码如下：

```
class Fu {
    public void show() {
        System.out.println("Fu show");
    }
}
class Zi extends Fu {
    //子类重写了父类的show方法
    public void show() {
        System.out.println("Zi show");
    }
}
public class ExtendsDemo05{
    public static void main(String[] args) {
        Zi z = new Zi();
        // 子类中有show方法，只执行重写后的show方法
        z.show();  // Zi show
    }
}
```

小结:

继承后的特点-----成员方法
父子类出现不重名的成员方法：无任何影响
父子类出现重名的成员方法:优先调用子类中定义的

1.5 方法的重写以及应用场景

目标:

- 方法的重写以及应用场景

步骤:

- 方法的重写
- 方法重写的应用场景

讲解:

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，这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**。
- **方法重载**: 在同一个类中,出现多个同名的方法,但是参数列表不同,与返回值类型,权限修饰符无关

重写方法的引用场景:

子类可以根据需要，定义特定于自己的行为。既沿袭了父类的功能名称，又根据子类的需要重新实现父类方法，从而进行扩展增强。比如新的手机增加来电显示头像的功能，代码如下：

```
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 {

    //重写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");
    }
}

public class ExtendsDemo06 {
    public static void main(String[] args) {
        // 创建子类对象
        NewPhone np = new NewPhone();

        // 调用父类继承而来的方法
        np.call();

        // 调用子类重写的方法
        np.showNum();

    }
}
```

> 小贴士：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。

小结:

- 重写的概述:父子类中出现一模一样的方法(返回值类型,方法名,参数列表)
  重写的应用场景:
  1.当父类中定义的方法无法满足子类的需求
  完全覆盖
  部分覆盖

1.6重写的注意事项:

目标:

- 重写的注意事项

步骤:

- 重写的注意事项

讲解:

1. 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。

   private(本类中) < (default 默认)(同一个包) < protected(本类和子类) < public(当前项目中)
2. 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。
3. 重写的方法可以使用@Override注解来标识
4. 父子类中出现了一模一样的静态方法,这不叫方法的重写,只是长得像方法的重写

小结:

方法的重写的注意事项：
    1. 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。
       private(本类中) < (default 默认)(同一个包) < protected(本类和子类) < public(当前项目中)
    2. 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。
    3. 重写的方法可以使用@Override注解来标识
        如果方法是重写的方法就可以使用@Override注解标识
        如果方法不是重写的方法就不可以使用@Override注解标识

    4. 父子类中出现了一模一样的静态方法,这不叫方法的重写,只是长得像方法的重写

1.6 继承后的特点——构造方法

目标:

当类之间产生了关系，其中各类中的构造方法，又产生了哪些影响呢？

步骤:

- 创建父子类
- 研究构造方法

讲解:

首先我们要回忆两个事情，构造方法的定义格式和作用。

1. 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。
2. 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，才可以给子类使用。代码如下：

```
class Fu {
  private int n;
  Fu(){
    System.out.println("Fu()");
  }
}
class Zi extends Fu {
  Zi(){
    // super ( ) ，调用父类构造方法
    super();
    System.out.println("Zi ( ) ");
  }
}
public class ExtendsDemo07{
  public static void main (String args[]){
    Zi zi = new Zi();
  }
}
```

输出结果：

```
Fu ( )
```

```
Zi ( )
```

小结:

- 父类的构造方法是不能被继承的
- 构造方法是用来初始化成员变量的,子类中的构造方法默认会调用父类的空参构造方法
- 子类中调用父类的空参构造方法: super()
- 子类中调用父类的有参构造方法: super(实参);

1.6 super和this

目标:

- super和this关键字

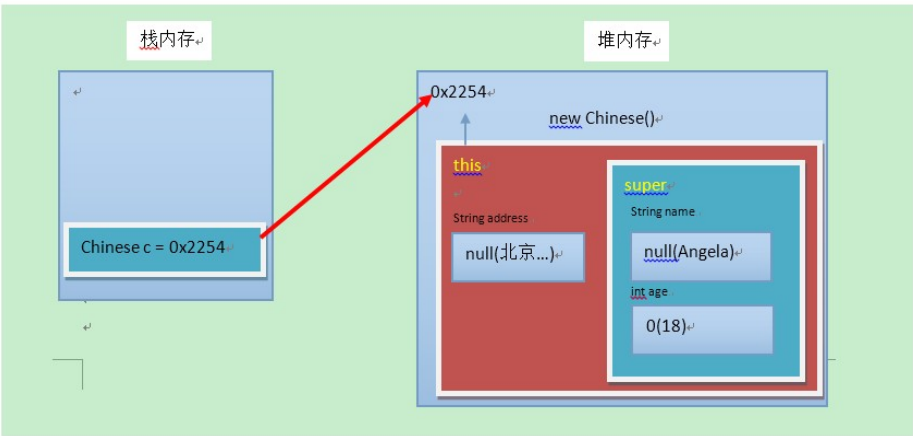
步骤:

- 分析继承的子类对象的内存图
- super的三种用法
- this的三种用法

讲解:

父类空间优先于子类对象产生

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，一定先调用父类的构造方法。理解图解如下：



super和this的含义

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

super和this的用法

1. 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
```

用法演示，代码如下：

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

class Cat extends Animal {
    public void eat() {
        System.out.println("cat : eat");
    }
    public void eatTest() {
        this.eat();   // this  调用本类的方法
        super.eat();  // super 调用父类的方法
    }
}

public class ExtendsDemo08 {
    public static void main(String[] args) {
        Animal a = new Animal();
        a.eat();
        Cat c = new Cat();
        c.eatTest();
    }
}
```

输出结果为：

```
animal : eat
cat : eat
animal : eat
```

2. 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。

小结:

略

1.7 继承的特点

目标:

- 继承的特点

步骤:

- Java只支持单继承，不支持多继承。
- Java支持多层继承(继承体系)。

讲解:

1. Java只支持单继承，不支持多继承。

```
//一个类只能有一个父类，不可以有多个父类。
class C extends A{}      //ok
class C extends A , B...  //error
```

2. Java支持多层继承(继承体系)。

```
class A{}
class B extends A{}
class C extends B{}
```

3. 子类拥有父类的成员变量和成员方法
4. 子类只能直接访问父类的非私有成员,不能直接访问私有成员(成员变量和成员方法),间接访问私有成员
    1. 访问父类的私有成员变量 通过set\get方法
    2. 访问父类的私有成员方法 通过其他公共的方法

小结:

- 注意:顶层父类是Object类。所有的类默认继承Object，作为父类。

第二章 抽象类

2.1 概述

目标:

- 抽象类和抽象方法的概念

步骤:

- 抽象类的概述
- 抽象方法的概述

讲解:

由来

父类中的方法，被它的子类们重写，子类各自的实现都不尽相同。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**。

抽象方法与抽象类

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

小结:

略

2.2 abstract使用格式

目标:

- abstract使用格式

步骤:

- abstract修饰抽象方法
- abstract修饰抽象类
- 抽象的使用

讲解:

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）;
```

代码举例：

```
public abstract void run();
```

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 类名字 {

}
```

代码举例：

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

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的子类都不能创建对象，失去意义。

代码举例：

```
public class Cat extends Animal {
    public void run (){
        System.out.println("小猫在墙头走~~~");
    }
}

public class CatTest {
     public static void main(String[] args) {
        // 创建子类对象
        Cat c = new Cat();

        // 调用run方法
        c.run();
    }
}
输出结果：
小猫在墙头走~~~
```

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**。

小结:

- 抽象类定义的格式
- 抽象方法的定义格式
- 抽象的使用
- ```
  抽象类:
      概述:有抽象方法的类就是抽象类,或者使用abstract修饰的类就是抽象类
      格式:
          public abstract class 类名{
  ```
  ```

```
        }

抽象方法:
    概述:使用abstract修饰的方法就是抽象方法,没有方法体
    格式:
        权限修饰符 abstract 返回值类型 方法名(参数列表);


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
     抽象类是不能创建对象的
```

2.3 注意事项

目标:

- abstract注意事项

步骤:

- abstract注意事项

讲解:

关于抽象类的使用，以下为语法上要注意的细节，虽然条目较多，但若理解了抽象的本质，无需死记硬背。

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。

   > 理解：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，是供子类创建对象时，初始化父类成员使用的。

   > 理解：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。

   > 理解：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。

   > 理解：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。

小结:

略

第三章 继承的综合案例

目标:

- 继承的综合案例

步骤:

- 需求介绍
- 案例分析
- 代码实现

讲解:

3.1 综合案例：群主发普通红包

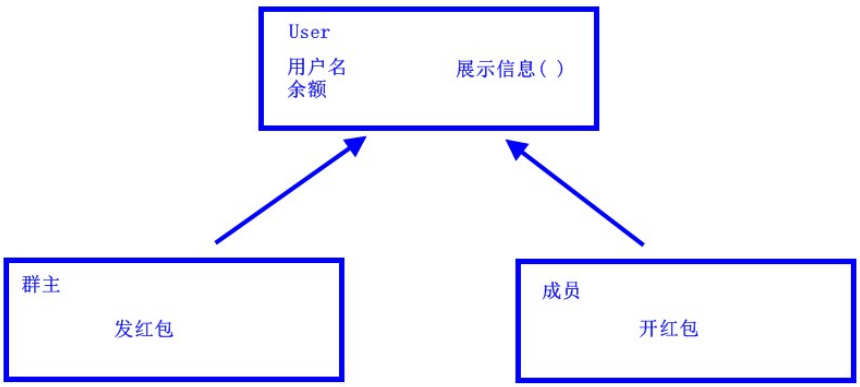
群主发普通红包。某群有多名成员，群主给成员发普通红包。普通红包的规则：

1. 群主的一笔金额，从群主余额中扣除，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，完成案例中所有类的定义以及指定类之间的继承关系，并完成发红包的操作。

3.2 案例分析

根据描述分析，得出如下继承体系：



3.3 案例实现

定义用户类：

```
public class User {
    private String name;// 姓名
    private int leftMoney;// 余额

    public User() {
    }

    public User(String name, int leftMoney) {
        this.name = name;
        this.leftMoney = leftMoney;
    }
```

```
    public String getName() {
        return name;
    }

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

    public int getLeftMoney() {
        return leftMoney;
    }

    public void setLeftMoney(int leftMoney) {
        this.leftMoney = leftMoney;
    }

    public void show(){
        System.out.println("我是:"+name+",我的余额是:"+leftMoney);
    }

}
```

定义群主类：

```
import java.util.ArrayList;

public class QunZhu extends User {

    public QunZhu() {
    }

    public QunZhu(String name, int leftMoney) {
        super(name, leftMoney);
    }

    // 发红包
    /*
        明确方法返回值类型:  ArrayList<Integer>
        明确方法名:   faHongBao
        明确方法参数:  红包金额 int  红包个数 int
        明确方法体:
           10元  5包   每个包2元
           10元  3包   3  3  3
     */
    public ArrayList<Integer> faHongBao(int redMoney,int count){
        // 0.创建一个集合用来存储这多个小红包
        ArrayList<Integer> list = new ArrayList<>();


        // 1.获取群主余额
```

```
        int leftMoney = getLeftMoney();

        // 2.判断是否够发红包  余额 >= 红包金额
        if (leftMoney < redMoney){
        // 3.如果不够发红包,就返回null
            return null;
        }else{
        // 4.如果够发红包,就发红包
            //  4.0 群主的余额要减少
            setLeftMoney(getLeftMoney() - redMoney);
        // 4.1 求这多个红包的平均金额
            int avg = redMoney / count;

        // 4.2 求除不尽剩余的金额
            int left = redMoney % count;

        // 4.3 把count-1个平均红包添加到集合中
            for (int i = 0;i<count-1;i++){
                list.add(avg);
            }
        // 4.4 把除不尽剩余的金额 加上 一个平均红包 添加到集合中
            list.add(avg+left);
        }
        // 5. 返回集合
        return list;
    }
}
```

定义成员类：

```
import java.util.ArrayList;
import java.util.Random;

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

    public QunYuan(String name, int leftMoney) {
        super(name, leftMoney);
    }

    // 抢红包的功能
     /*
        明确方法返回值类型:  void
        明确方法名:   qiangHongBao
        明确方法参数:  ArrayList<Integer>
        明确方法体:
     */
     public void qiangHongBao(ArrayList<Integer> list){
         // 随机抽取一个红包  集合是根据索引取元素,所以就随机产生一个索引

         // 创建随机生成器
```

```
            Random r = new Random();

            // 随机产生一个索引 [0,list.size-1]
            int index = r.nextInt(list.size());//[0]  ---0
            System.out.println(index);

            // 根据产生随机索引去list集合中取出红包
            Integer redMoney = list.get(index);

            // 要删除刚刚取出来的红包金额元素
            list.remove(index);

            // 把抢到的红包金额添加到自己的余额中
            setLeftMoney(getLeftMoney() + redMoney);
        }
}
```

定义测试类：

```
public class Demo12 {
    public static void main(String[] args) {
        QunZhu qz = new QunZhu("群主",10);

        QunYuan qy1 = new QunYuan("群员1",0);
        QunYuan qy2 = new QunYuan("群员2",0);
        QunYuan qy3 = new QunYuan("群员3",0);

        // 群主发红包
        ArrayList<Integer> list = qz.faHongBao(10, 3);
        // {4}
        // 群员抢红包
        qy1.qiangHongBao(list);// list集合的长度是3
        qy2.qiangHongBao(list);// list集合的长度是2
        qy3.qiangHongBao(list);// list集合的长度是1

        // 显示信息
        qz.show();
        qy1.show();
        qy2.show();
        qy3.show();

    }
}
```

小结:

略