day09面向对象继承多态

- 学习目标
  - 继承概念
  - 继承的关键字 extends
  - 继承后子类父类的特点
    - 成员变量
    - 成员方法
    - 构造方法
  - Java中继承的特点
  - 继承的好处
  - 对象的多态性
  - 多态的必要因素
  - 多态的语法格式
  - 多态中成员的特性
  - 多态中的转型

1. 继承

1.1 生活中的继承

生活中,子继承父的遗产,不是全部,只能是部分

1.2 程序中的继承

在程序中,也能出现继承的关系,让一个类去继承另一个类.

> 比如 A类继承B类

- 出现继承的关系
- A类是B类的子类,或者称为派生类
- B类是A类的父类,或者称为超类,基类
- 子类可以直接拥有父类的成员(不是全部)

1.3 继承的语法格式

继承使用关键字 extends,表示继承的意思

注意: extend 本身含义,扩展,延伸

- 定义格式

```
class B{}
class A extends B{}  // A 继承 B类
```

- 继承的入门案例 -- 子类拥有父类成员

```java
/**
 *  定义的是 Teacher和Manager类的共性内容
 *  姓名年龄
 */
public class Person {
    String name ;
    int age;
}
```

```java
/**
 *  管理类,班主任类
 */
public class Manager extends Person{

}

```

```java
/**
 *  老师类
 *  重复,只做一次
 *  想法:
 *    String name;
 *     int age;
 *  换一个地方定义(共性抽取)
 *  Teacher和Manager共用
 *  使用Person中的成员
 *  Teacher继承Person
 */
public class Teacher extends Person{

}
```

```java
    public static void main(String[] args) {
        /**
         *  创建对象, Person的子类对象
         *  Person : Teacher,Manager
         */
        Teacher teacher = new Teacher();
        //teacher对象,调用成员变量
        //teacher子类对象调用继承的成员变量
        teacher.name = "张三";
        teacher.age = 20;

        Manager manager = new Manager();
        manager.name = "李四";
        manager.age = 22;

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

        System.out.println("manager.name = " + manager.name);
        System.out.println("manager.age = " + manager.age);
    }
```

1.4 继承的好处

上面案例,使用程序中的继承,使用继承的好处:

- 减少代码量
- 复用性提高
- 继承的存在,导致了面向对象的最后一个特征多态

  继承有弊端: 类和类之间的紧密性更强.(扩展性越差)

2. 继承后成员特点

2.1 继承后成员变量特点

子类和父类中的成员变量同名

- 调用的使用: 子类自己有,使用自己的,子类没有,使用父类

```
public class Fu{
    String s = "父类";
}

public class Zi extends Fu{
    String s = "子类";
}

public static void main(String[] args) {
    //创建对象,子类对象
    Zi zi = new Zi();
    //子类对象,调用成员s
    System.out.println(zi.s);
}
```

2.2 super关键字

super关键字是超级的意思,在子类中调用父类的成员,使用此关键字

```
super.变量   调用父的成员变量
super.方法()  调用的是父类的成员方法
```

this表示当前对象,super表示父类在内存中的存储空间,不是对象

```
public  class Fu{
    String s = "父类";
}
public class Zi extends Fu {
    String s = "子类";

    public void print(){
        String s = "方法";
        System.out.println(s);
        System.out.println(this.s);
        System.out.println(super.s);
    }
```

```
}
public static void main(String[] args) {
    //创建子类对象,调用方法
    Zi zi =  new Zi();
    zi.print();
}
```

2.3 继承后成员方法特点

方法重写override:子类父类出现了一模一样的方法,称为子类重写了父类的方法.又称为覆盖或者复写.

调用子类重写的方法,假如子类没有重写,调用父类的方法

```
public class Fu {
    public void print(){
        System.out.println("父类方法print");
    }
}
public class Zi extends Fu {
    /**
     * 要重写父类的方法
     * 直接复制
     */
    public void print(){
        System.out.println("子类方法print::重写");
    }
}
    public static void main(String[] args) {
        //创建子类对象
        Zi zi = new Zi();
        zi.print();
    }
```

2.4 方法重写的意义

继承本质是扩展的意思,延伸的意思,依靠方法的重写来实现

> 方法重写引入案例,理解重写的意义

```
/**
 *  定义手机类
 *  早年代手机
 */
public class Phone {
    /**
     * 定义手机的来电显示功能
     * 数字移动电话
     */
    public void showingCall(){
        System.out.println("来电显示号码");
    }
}

/**
 *  新手机:
```

```
 *  功能变强,但是原有功能继续复用
 *  继承和方法重写的思想
 */
public class Iphone extends Phone {
    //重写父的方法
    //方法名字不变,用户不需要重新知
    public void showingCall(){
        //复用显示号码功能,父类的方法中,已经完成了
        //调用父类的方法
        super.showingCall();
        //新增的功能
        System.out.println("显示大头像");
        System.out.println("归属地");
        System.out.println("意思推销");
    }
}
public static void main(String[] args) {
    //创建手机对象
    Phone phone = new Phone();
    phone.showingCall();
    System.out.println("==========");
    //创建新手机对象
    Iphone iphone = new Iphone();
    iphone.showingCall();
}
```

2.5 方法重写小问题

> 方法重写需要考虑权限的.保证子类方法的权限要大于或者等于父类方法权限

- 父类的方法权限是public

```
class Fu{
    public void print(){}
}
class Zi extends Fu{
    public void print(){} //正确
    protected void print(){} //错误,权限降低
    void print(){} //错误,权限降低
    private void print(){} //错误,权限降低
}
```

- 父类的方法权限是protected

```
class Fu{
    protected void print(){}
}

class Zi extends Fu{
    public void print(){} //正确
    protected void print(){} //正确
    void print(){} //错误,权限降低
    private void print(){} //错误,权限降低
}
```

- 父类方法权限是默认

```
class Fu{
     void print(){}
}

class Zi extends Fu{
    public void print(){} //正确
    protected void print(){} //正确
    void print(){} //正确
    private void print(){} //错误,权限降低
}
```

- 如果父类的方法权限是private, 子类不知道该方法的存在,没有继承的说法

2.6 继承后构造方法特点

构造方法特点: 子类的构造方法中,第一行存在隐式代码 (写不写都存在),代码是**super();** 调用父类的无参数构造方法.

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

- 子类的构造方法,无论重载多少个,第一行肯定也是super();

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

    public Zi(int a){
        super(); //调用父类的无参数构造方法.
        System.out.println("子类构造方法::重载的");
    }
}
```

- 父类没有无参数构造方法,子类的构造方法中,super(传递参数)
- 父类中存在多个构造方法,子类的构造方法只要调用到其中的一个即可

```
public class Fu  {
    /**
     * 父类的构造方法
     * 加上参数,有参数的构造
```

```
     */
    public Fu(int a){
        System.out.println("父类构造方法" + a);
    }

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

public class Zi extends Fu {

    public Zi(){
        //调用父类的无参数构造方法
        //父类中没有无参数构造
        //传递响应的参数
        super(7);
    }

    public Zi(String s){
        //调用父类构造方法,保证调用其中一个
        super("字符串");
    }
}
```

3. 继承特点

3.1 单继承

一个类只能继承一个类,不允许同时继承多个类

```
class A extends B,C{} //不允许的行为
```

单继承存在局限性,解决局限性问题,接口的概念

3.2 多层继承

```
class A extends B{}
class B extends C{}
//多层继承,允许实现
//class C extends Object{}
```

A类可以同是拥有B和C的成员, B只能拥有C的成员

A类中super调用的是B类成员,如果B类没有成员,调用C成员

Object类是java中的皇帝,所有的类都是Object子类

3.3 继承体系