

多态动手动脑

一、怎样判断对象是否可以转换？可以使用**instanceof**运算符判断一个对象是否可以转换为指定的类型，参看实例：**TestInstanceof.java**

```
public class TestInstanceof
{
    public static void main(String[] args)
    {
        //声明hello时使用Object类，则hello的编译类型是Object，Object是所有类的父类
        //但hello变量的实际类型是String
        Object hello = "Hello";
        //String是Object类的子类，所以返回true。
        System.out.println("字符串是否是Object类的实例：" + (hello instanceof Object));
        //返回true。
        System.out.println("字符串是否是String类的实例：" + (hello instanceof String));
        //返回false。
        System.out.println("字符串是否是Math类的实例：" + (hello instanceof Math));
        //String实现了Comparable接口，所以返回true。
        System.out.println("字符串是否是Comparable接口的实例：" + (hello instanceof Comparable));
        String a = "Hello";
        //String类既不是Math类，也不是Math类的父类，所以下面代码编译无法通过
        //System.out.println("字符串是否是Math类的实例：" + (a instanceof Math));
    }
}
```



```
Console x
<terminated> TestInstanceof [Java Application] C:\Progra
字符串是否是Object类的实例: true
字符串是否是String类的实例: true
字符串是否是Math类的实例: false
字符串是否是Comparable接口的实例: true
```

二、下列语句哪一个将引起编译错误?为什么?哪一个会引起运行时错误?为什么?

m=d;

```
d=m;
```

```
d=(Dog)m;
```

```
d=c;
```

```
c=(Cat)m;
```

先进行自我判断，得出结论后，运行TestCast.java实例代码，看看你的判断是否正确

```
class Mammal {}
class Dog extends Mammal {}
class Cat extends Mammal {}

public class TestCast
{
    public static void main(String args[])
    {
        Mammal m;
        Dog d=new Dog();
        Cat c=new Cat();
        m=d;
        //d=m;
        d=(Dog)m;
        //d=c;
        //c=(Cat)m;

    }
}
```

```
d=m;
```

```
d=c;
```

```
c=(Cat)m;
```

这三句话有错。

三、请看以下 **“变态” 的类**（参看示例ParentChildTest.java），运行以下测试代码，并回答如下问题

1.程序运行结果是什么？

2.你如何解释会得到这样的输出？

3.计算机是不会出错的，之所以得到这样的运行结果也是有原因的，那么从这些运行结果中，你能总结出Java的哪些语法特性？

```
public class ParentChildTest {
    public static void main(String[] args) {
        Parent parent=new Parent();
        parent.printValue();
        Child child=new Child();
        child.printValue();

        parent=child;
        parent.printValue();

        parent.myValue++;
        parent.printValue();

        ((Child)parent).myValue++;
        parent.printValue();
    }
}

class Parent{
    public int myValue=100;
    public void printValue() {
        System.out.println("Parent.printValue(),myValue="+myValue);
    }
}

class Child extends Parent{
    public int myValue=200;
    public void printValue() {
        System.out.println("Child.printValue(),myValue="+myValue);
    }
}
```

1、结果

```
Console x
<terminated> ParentChildTest [Java Application] C:\Program Files (
Parent.printValue(),myValue=100
Child.printValue(),myValue=200
Child.printValue(),myValue=200
Child.printValue(),myValue=200
Child.printValue(),myValue=201
```

2、Java语法特性

(1) 当子类与父类拥有一样的方法，并且让一个父类变量引用一个子类对象时，到底调用哪个方法，由对象自己的“真实”类型所决定，这就是说：对象是子类型的，它就调用子类型的方法，是父类型的，它就调用父类型的方法。

这个特性实际上就是面向对象“多态”特性的具体表现。

(2) 如果子类与父类有相同的字段,则子类中的字段会代替或隐藏父类的字段，子类方法中访问的是子类中的字段（而不是父类中的字段）。如果子类方法确实想访问父类中被隐藏的同名字段，可以用super关键字来访问它。如果子类被当作父类使用,则通过子类访问的字段是父类的!

四、请使用**javap**查看编译器为**TestPolymorphism.java**生成的字节码指令，然后通过互联网搜索资料，尝试从底层开始理解**Java**编译器是如何为多态代码生成字节码指令，在程序运行过程中，多态特性又是如何实现。

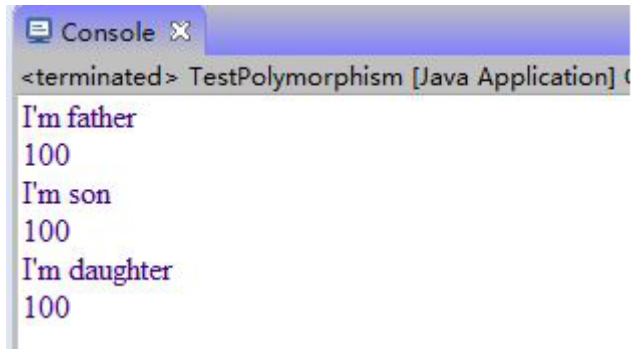
```
class Parent1
{
    public int value=100;
    public void Introduce()
    {
        System.out.println("I'm father");
    }
}
class Son extends Parent1
{
    public int value=101;
    public void Introduce()
    {
        System.out.println("I'm son");
    }
}
```

```

class Daughter extends Parent1
{
    public int value=102;
    public void Introduce()
    {
        System.out.println("I'm daughter");
    }
}
public class TestPolymorphism
{
    public static void main(String args[])
    {
        Parent1 p=new Parent1();
        p.Introduce(); //子类的方法
        System.out.println(p.value); //父类的变量对象
        p=new Son();
        p.Introduce();
        System.out.println(p.value);
        p=new Daughter();
        p.Introduce();
        System.out.println(p.value);
    }
}

```

结果:



```

Console X
<terminated> TestPolymorphism [Java Application] C
I'm father
100
I'm son
100
I'm daughter
100

```

五、在实例中理解多态的含义与用途



```
<terminated> Zoo [Java Application] C:\Program Files
我不吃肉谁敢吃肉！
我什么都吃，尤其喜欢香蕉。
我什么都吃，尤其喜欢香蕉。
我什么都吃，尤其喜欢香蕉。
我什么都吃，尤其喜欢香蕉。
我什么都吃，尤其喜欢香蕉。
我什么都吃，尤其喜欢香蕉。
我什么都吃，尤其喜欢香蕉。
我什么都吃，尤其喜欢香蕉。
我什么都吃，尤其喜欢香蕉。
我要减肥，所以每天只吃一点大米。
我要减肥，所以每天只吃一点大米。
我要减肥，所以每天只吃一点大米。
我要减肥，所以每天只吃一点大米。
我要减肥，所以每天只吃一点大米。
```

1、三种动物对应三个类，每个类定义一个`eat()`方法，表示吃饲养员给它们的食物，再设计一个**Feeder**类代表饲养员，其`name`字段保存饲养员名字，三个方法分别代表喂养三种不同的动物，其参数分别引用三种动物对象。

```
package Zool;
public class Zoo
{
    public static void main(String args[])
    {
        Feeder f = new Feeder("小李");
        // 饲养员小李喂养一只狮子
        f.feedLion(new Lion());
        // 饲养员小李喂养十只猴子
        for (int i = 0; i < 10; i++)
        {
            f.feedMonkey(new Monkey());
        }
        // 饲养员小李喂养5只鸽子
        for (int i = 0; i < 5; i++)
        {
            f.feedPigeon(new Pigeon());
        }
    }
}
```

```

    }
}
class Feeder
{
    public String name;
    public Feeder(String name)
    {
        this.name = name;
    }
    public void feedLion(Lion l)
    {
        l.eat();
    }
    public void feedPigeon(Pigeon p)
    {
        p.eat();
    }
    public void feedMonkey(Monkey m)
    {
        m.eat();
    }
}
class Lion
{
    public void eat()
    {
        System.out.println("我不吃肉谁敢吃肉!");
    }
}
class Monkey
{
    public void eat()
    {
        System.out.println("我什么都吃，尤其喜欢香蕉。");
    }
}
class Pigeon
{
    public void eat()
    {
        System.out.println("我要减肥，所以每天只吃一点大米。");
    }
}

```

2、第一次程序重构：引入继承，简化Feeder类

```
package Zoo2;
public class Zoo
{
    public static void main(String args[])
    {
        Feeder f = new Feeder("小李");
        //饲养员小李喂养一只狮子
        f.feedAnimal(new Lion());
        //饲养员小李喂养十只猴子
        for (int i = 0; i < 10; i++)
        {
            f.feedAnimal(new Monkey());
        }
        //饲养员小李喂养5只鸽子
        for (int i = 0; i < 5; i++)
        {
            f.feedAnimal(new Pigeon());
        }
    }
}
class Feeder
{
    public String name;
    Feeder(String name)
    {
        this.name = name;
    }
    public void feedAnimal(Animal an)
    {
        an.eat();
    }
}
abstract class Animal
{
    public abstract void eat();
}
class Lion extends Animal
{
    public void eat()
    {
        System.out.println("我不吃肉谁敢吃肉!");
    }
}
class Monkey extends Animal
{
    public void eat()
    {

```



```

        System.out.println("我什么都吃，尤其喜欢香蕉。");
    }
}
class Pigeon extends Animal
{
    public void eat()
    {
        System.out.println("我要减肥，所以每天只吃一点大米。");
    }
}

```

3、第二次程序重构，修改**feedAnimals**方法，让它接收一个**Animal**数组……

```

package Zoo3;
public class Zoo {
    public static void main(String args[]) {
        Feeder f = new Feeder("小李");
        Animal[] ans = new Animal[16];
        //饲养员小李喂养一只狮子
        ans[0] = new Lion();
        //饲养员小李喂养十只猴子
        for (int i = 0; i < 10; i++) {
            ans[1 + i] = new Monkey();
        }
        //饲养员小李喂养5只鸽子
        for (int i = 0; i < 5; i++) {
            ans[11 + i] = new Pigeon();
        }
        f.feedAnimals(ans);
    }
}
class Feeder {
    public String name;
    Feeder(String name) {
        this.name = name;
    }
    public void feedAnimals(Animal[] ans) {
        for (Animal an : ans) {
            an.eat();
        }
    }
}
abstract class Animal {
    public abstract void eat();
}
class Lion extends Animal {

```

```

        public void eat() {
            System.out.println("我不吃肉谁敢吃肉！");
        }
    }
}
class Monkey extends Animal {
    public void eat() {
        System.out.println("我什么都吃，尤其喜欢香蕉。");
    }
}
class Pigeon extends Animal {
    public void eat() {
        System.out.println("我要减肥，所以每天只吃一点大米。");
    }
}
}

```

4、第三次重构，修改**feedAnimals**方法，让其接收一个元素数目可变的对象容器。

```

package Zoo4;
import java.util.Vector;
public class Zoo {
    public static void main(String args[]) {
        Feeder f = new Feeder("小李");
        Vector<Animal> ans = new Vector<Animal>();
        //饲养员小李喂养一只狮子
        ans.add(new Lion());
        //饲养员小李喂养十只猴子
        for (int i = 0; i < 10; i++) {
            ans.add(new Monkey());
        }
        //饲养员小李喂养5只鸽子
        for (int i = 0; i < 5; i++) {
            ans.add(new Pigeon());
        }
        f.feedAnimals(ans);
    }
}
class Feeder {
    public String name;
    Feeder(String name) {
        this.name = name;
    }
    public void feedAnimals(Vector<Animal> ans) {
        for (Animal an : ans) {
            an.eat();
        }
    }
}

```

```

}
abstract class Animal {
    public abstract void eat();
}
class Lion extends Animal {
    public void eat() {
        System.out.println("我不吃肉谁敢吃肉！");
    }
}
class Monkey extends Animal {
    public void eat() {
        System.out.println("我什么都吃，尤其喜欢香蕉。");
    }
}
class Pigeon extends Animal {
    public void eat() {
        System.out.println("我要减肥，所以每天只吃一点大米。");
    }
}
}

```

5、从这个示例中可以看到，通过在编程中应用多态，可以使我们的代码具有更强的适用性。当需求变化时，多态特性可以帮助我们减少需要改动的地方到最低限度。

多态编程有两种主要形式：

- （1）继承多态：示例程序使用的方法
- （2）接口多态：使用接口代替抽象基类。

使用多态最大的好处是：

当你要修改程序并扩充系统时，你需要修改的地方较少，对其它部分代码的影响较小！千万不要小看这两个“较”字！程序规模越大，其优势就越突出。

六、用多态的方法模拟**ATM**操作流程

```

//王荣荣 2016/11/18
import java.util.Scanner;
class PersonalAccount{
    private String passWord="123456";//密码
    private String number;//银行卡号
}

```

```

private int money=0;
public int getMoney() {
    return money;
} //余额
public void setPw(String s) {
    passWord=s;
} //设置密码
public void addMoney(int x) {
    money+=x;
} //加钱
public void minusMoney(int x) {
    money-=x;
} //减钱
public boolean whetherPwTrue(String s) { //密码是否正确
    if(s.equals(passWord))
        return true;
    else return false;
}
}

abstract class PATM {
    abstract boolean withdraw(int x); //取款
    abstract void save(int x); //存款
    abstract boolean transfer(String s, int x); //转账
    abstract boolean ifPass(String s); //判断输入的密码是否正确
    abstract int getRest(); //查询余额
    abstract void setPassword(String s); //设置密码
}

class ATM extends PATM {
    private String numbers[] = {"123451", "123452",
        "123453", "123454", "123455"}; //数据库中已有的账户卡号
    private PersonalAccount account = new PersonalAccount();
    public boolean withdraw(int x) {
        if(x > account.getMoney())
            return false;
        else {
            account.minusMoney(x);
            return true;
        }
    }
    public void save(int x) {
        account.addMoney(x);
    }
    public boolean transfer(String s, int x) {
        //转账
        //先判断转到账户号是否存在
        //再判断余额是否足够
        boolean flag = false;

```

```

        for(int i=0;i<numbers.length;i++)
            if(s.equals(numbers[i])) flag=true;
        if(x>account.getMoney()) flag=false;
        if(x<=account.getMoney()&&flag) account.minusMoney(x);;
        return flag;
    }
    public boolean ifPass(String s) {
        return account.whetherPwTrue(s);
    }
    public int getRest() {
        return account.getMoney();
    }
    public void setPassword(String s) {
        account.setPw(s);
    }
}
public class Atm1 {
    public static void main(String[] args) {
        Scanner in=new Scanner(System.in);
        ATM atm=new ATM();
        int choose=0,num=0;
        String pw="";
        next:while(true){
            System.out.println("是否进入账户（0否1是）：");
            int kk=in.nextInt();
            if(kk==0) break;
            else if(kk!=1){
                System.out.println("输入错误！");
                continue;
            }
            System.out.println("输入账户密码：");
            pw=in.next();
            if(atm.ifPass(pw)){
                while(true){
                    showFace();
                    choose=in.nextInt();
                    switch(choose){
                        case 1:
                            System.out.println("输入存款金额：");
                            num=in.nextInt();
                            atm.save(num);
                            System.out.println("存款成功！");
                            System.out.println("当前余额："+atm.getRest()+"元");
                            break;
                        case 2:
                            System.out.println("请选择：");
                            int a[]={100,500,1000,1500,2000,5000};

```

```

        for(int i=0;i<a.length;i++)
            System.out.println((i+1)+". "+a[i]+"元");
        System.out.println("7. 其他");
        int ch=in.nextInt();
        if(ch>=1&&ch<=6){
            if(atm.withdraw(a[ch-1]))
                System.out.println("取款成功!");
            else
                System.out.println("余额不足!");
        }
        else if(ch==7){
            System.out.println("请输入取款金额: ");
            num=in.nextInt();
            if(atm.withdraw(num))
                System.out.println("取款成功!");
            else
                System.out.println("余额不足!");
        }
        else
            System.out.println("输入有误!");
        System.out.println("当前余额: "+atm.getRest()+"元");
        break;
    case 3:
        System.out.println("账户号: ");
        String s=in.next();
        System.out.println("转账金额: ");
        int i=in.nextInt();
        if(atm.transfer(s, i))
            System.out.println("转账成功!");
        else
            System.out.println("转账失败!");
        System.out.println("当前余额: "+atm.getRest()+"元");
        break;
    case 4:
        System.out.println("输入六位数密码: ");
        String p=in.next();
        atm.setPassword(p);
        break;
    case 5:
        System.out.println("当前余额: "+atm.getRest()+"元");
        break;
    default:
        continue next;
    }
}
else

```

```
        System.out.println("密码错误！");
    }
}
//显示菜单方法
public static void showFace() {
    System.out.println("1. 存款");
    System.out.println("2. 取款");
    System.out.println("3. 转账汇款");
    System.out.println("4. 修改密码");
    System.out.println("5. 查询余额");
    System.out.println("6. 退卡");
    System.out.println("请选择：");
}
}
```

结果:

```
Console X
<terminated> Atm1 [Java Application]
是否进入账户 (0否1是):
1
输入账户密码:
123456
1.存款
2.取款
3.转账汇款
4.修改密码
5.查询余额
6.退卡
请选择:
1
输入存款金额:
900
存款成功!
当前余额: 900元
1.存款
2.取款
3.转账汇款
4.修改密码
5.查询余额
6.退卡
请选择:
2
```


请选择:

1.100元

2.500元

3.1000元

4.1500元

5.2000元

6.5000元

7.其他

2

取款成功!

当前余额: 400元

1.存款

2.取款

3.转账汇款

4.修改密码

5.查询余额

6.退卡

请选择:

3

账户号:

123455

转账金额:

200

转账成功!

当前余额: 200元

1.存款

2.取款

3.转账汇款

4.修改密码

5.查询余额

6.退卡

请选择:

4

输入六位数密码:

123457

1.存款

2.取款

3.转账汇款

4.修改密码

5.查询余额

6.退卡

请选择:

6

是否进入账户(0否1是):

1

输入账户密码:

123456

密码错误!

是否进入账户(0否1是):

1

输入账户密码:

123457

1.存款

2.取款

3.转账汇款

4.修改密码

5.查询余额

6.退卡