**Java 多态**

多态是同一个行为具有多个不同表现形式或形态的能力。

多态性是对象多种表现形式的体现。

比如我们说"宠物"这个对象，它就有很多不同的表达或实现，比如有小猫、小狗、蜥蜴等等。那么我到宠物店说"请给我一只宠物"，服务员给我小猫、小狗或者蜥蜴都可以，我们就说"宠物"这个对象就具备多态性。

接下来让我们通过实例来了解Java的多态。

**例子**

public interface Vegetarian{}

public class Animal{}

public class Deer extends Animal implements Vegetarian{}

因为Deer类具有多重继承，所以它具有多态性。以上实例解析如下：

* 一个 Deer IS-A（是一个） Animal
* 一个 Deer IS-A（是一个） Vegetarian
* 一个 Deer IS-A（是一个） Deer
* 一个 Deer IS-A（是一个）Object

在Java中，所有的对象都具有多态性，因为任何对象都能通过IS-A测试的类型和Object类。

访问一个对象的唯一方法就是通过引用型变量。

引用型变量只能有一种类型，一旦被声明，引用型变量的类型就不能被改变了。

引用型变量不仅能够被重置为其他对象，前提是这些对象没有被声明为final。还可以引用和它类型相同的或者相兼容的对象。它可以声明为类类型或者接口类型。

当我们将引用型变量应用于Deer对象的引用时，下面的声明是合法的：

Deer d = new Deer();

Animal a = d;

Vegetarian v = d;

Object o = d;

所有的引用型变量d,a,v,o都指向堆中相同的Deer对象。

**虚方法**

我们将介绍在Java中，当设计类时，被重载的方法的行为怎样影响多态性。

我们已经讨论了方法的重载，也就是子类能够重载父类的方法。

当子类对象调用重载的方法时，调用的是子类的方法，而不是父类中被重载的方法。

要想调用父类中被重载的方法，则必须使用关键字super。

/\* 文件名 : Employee.java \*/

public class Employee

{

private String name;

private String address;

private int number;

public Employee(String name, String address, int number)

{

System.out.println("Constructing an Employee");

this.name = name;

this.address = address;

this.number = number;

}

public void mailCheck()

{

System.out.println("Mailing a check to " + this.name

+ " " + this.address);

}

public String toString()

{

return name + " " + address + " " + number;

}

public String getName()

{

return name;

}

public String getAddress()

{

return address;

}

public void setAddress(String newAddress)

{

address = newAddress;

}

public int getNumber()

{

return number;

}

}

假设下面的类继承Employee类：

/\* 文件名 : Salary.java \*/

public class Salary extends Employee

{

private double salary; //Annual salary

public Salary(String name, String address, int number, double

salary)

{

super(name, address, number);

setSalary(salary);

}

public void mailCheck()

{

System.out.println("Within mailCheck of Salary class ");

System.out.println("Mailing check to " + getName()

+ " with salary " + salary);

}

public double getSalary()

{

return salary;

}

public void setSalary(double newSalary)

{

if(newSalary >= 0.0)

{

salary = newSalary;

}

}

public double computePay()

{

System.out.println("Computing salary pay for " + getName());

return salary/52;

}

}

现在我们仔细阅读下面的代码，尝试给出它的输出结果：

/\* 文件名 : VirtualDemo.java \*/

public class VirtualDemo

{

public static void main(String [] args)

{

Salary s = new Salary("Mohd Mohtashim", "Ambehta, UP", 3, 3600.00);

Employee e = new Salary("John Adams", "Boston, MA", 2, 2400.00);

System.out.println("Call mailCheck using Salary reference --");

s.mailCheck();

System.out.println("\n Call mailCheck using Employee reference--");

e.mailCheck();

}

}

以上实例编译运行结果如下：

Constructing an Employee

Constructing an Employee

Call mailCheck using Salary reference --

Within mailCheck of Salary class

Mailing check to Mohd Mohtashim with salary 3600.0

Call mailCheck using Employee reference--

Within mailCheck of Salary class

Mailing check to John Adams with salary 2400.0

例子中，我们实例化了两个Salary对象。一个使用Salary引用s，另一个使用Employee引用。

编译时，编译器检查到mailCheck()方法在Salary类中的声明。

在调用s.mailCheck()时，Java虚拟机(JVM)调用Salary类的mailCheck()方法。

因为e是Employee的引用，所以调用e的mailCheck()方法则有完全不同的结果。

当编译器检查e.mailCheck()方法时，编译器检查到Employee类中的mailCheck()方法。

在编译的时候，编译器使用Employee类中的mailCheck()方法验证该语句， 但是在运行的时候，Java虚拟机(JVM)调用的是Salary类中的mailCheck()方法。

该行为被称为虚拟方法调用，该方法被称为虚拟方法。

Java中所有的方法都能以这种方式表现，借此，重写的方法能在运行时调用，不管编译的时候源代码中引用变量是什么数据类型。