情境1：继承与多态

教学内容

1.继承的使用

2.继承的实现

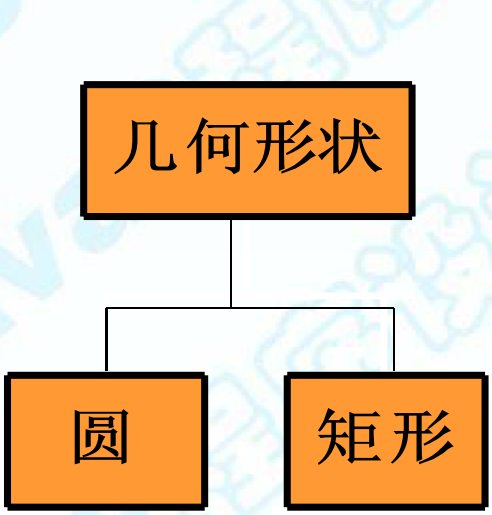
3.方法的重写与变量覆盖

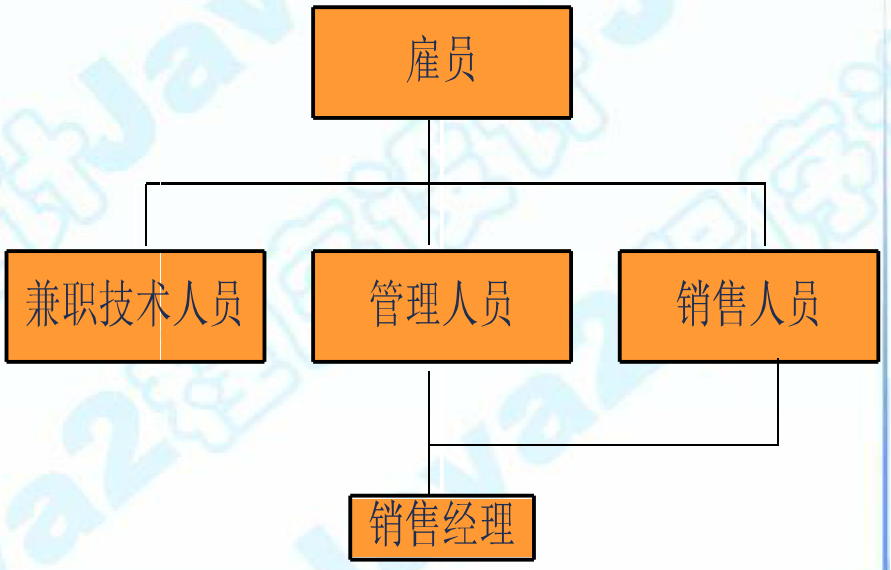
4.对象间的类型转换

5.多态性

继承与多态

- 继承性
  - 子类可以沿用父类的某些特征。子类也可以具有自己独立的属性和操作。
  - 继承分单继承和多继承。

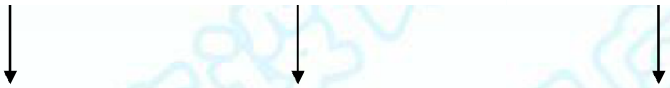




继承性

- 继承的目的：实现代码重用。
- 子类不能继承父类中访问权限为**private**的成员变量和方法。
- 子类可以继承父类方法和成员变量，但它不能继承构造方法。
- 创建子类格式：

[访问权限] **class** 类名 [**extends** 父类]



```
Public  class SubClass extends SuperClass
{
    …
}
```

继承性

例：源程序名**Car Demo .java**，是类的继承。

```
class Car
{
  int car_number;
  void set_number(int car_num)
  {
    car_number=car_num;
   }
  void show_number()
  {
   System.out.println ("My car No. is :"+car_number);
  }
 }
```

```
class Bus extends Car
{
  int capacity;
  void set_capacity(int bus_car_capacity)
  {
    capacity=bus_car_capacity;
  }
  void show_capacity()
  {
    System.out.println("My capacity is: "+ capacity);
  }
}
```

继承性

```
class CarDemo
{
 public static void main(String args[])
 {
 Bus DemoBus = new Bus();
 DemoBus.set_number(6868);
 DemoBus.show_number();
 DemoBus.set_capacity(30);
 DemoBus.show_capacity();
 }
}
```

运行结果：

多态性

- 多态性体现在两个方面：由方法重载实现的静态多态性（编译时多态）和方法重写实现的动态多态性（运行时多态）。
- 方法重写：指方法的含义被重写后替代 。
  - 子类通过隐藏父类的成员变量和重写父类的方法，可以把父类的状态和行为改变为自身的状态和行为。
  - 子类中重写的方法和父类中被重写的方法要具有相同的名字，相同的参数表和相同的返回类型，只是函数体不同。
  - 方法重写时应遵循的原则：

改写后的方法不能比被重写的方法有更严格的访问权限（可以相同）。改写后的方法不能比重写的方法产生更多的例外。

例：源程序名**Rewritex.java,**，是方法的重写的例子。

```
class SuperClass
{
  int x;
  void setF( )
  {
   x=0;
   System.out.println("Super的变量x的值"+x);
   }
}
```

多态性

```
class SubClass extends
SuperClass
{
 int x;//隐藏了父类的变量x
 void setF( )
   {      //重写了父类的方法 setF()
    x =6;
    System.out.println("Son的变量x的值"+x);
   }
 }
```

多态性

```
class Rewritex
{
 public static void main(String args[])
 {
  SubClass Son = new SubClass ();
  Son.setF();
  SuperClass Father= new SubClass ();
  Father.setF();
 }
}
```

运行结果：

多态性

方法重载

- 方法重载是指同样方法在不同的地方具有不同的含义。
- 方法重载使一个类中可以有多个相同名字的方法。编译器根据参数的不同来静态确定调用相应的方法。
- 方法重载中这些方法的参数必须不同，或是参数的数量不同，或是参数的类型不同。

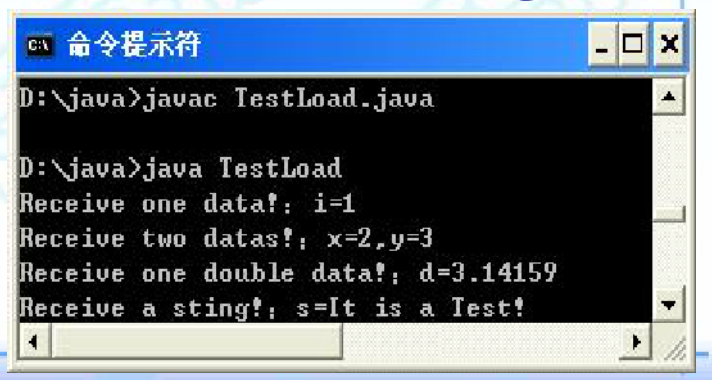
例:源程序名**TestLoad.java**，方法重载的例子。

```
/*定义一个方法重载的类*/
class Load
{
   void receive(int i)  // receive方法，带有int类型参数1个
    {
     System.out.print("Receive one data!：");
     System.out.println("i="+i);
    }
```

```
    void receive(int x,int y)  // receive方法，带有int类型参数2个
     {
       System.out.print("Receive two datas!：");
       System.out.println("x="+x+",y="+y);
     }
    void receive(double d)  // receive方法，带有double类型参数1个
     {
       System.out.print("Receive one double data!：");
       System.out.println("d="+d);
     }
  void receive(String s)  // receive方法，带有string类型参数1个
    {
      System.out.print("Receive a sting!：");
      System.out.println("s="+s);
    }
}
```

```
/*主程序类*/
 public class TestLoad
 {
   public static void main(String args[])
   {
     Load mo=new Load (); //创建对象mo
     mo.receive(1);      //调用receive方法，带int类型参数1个
    mo.receive(2,3);     //调用receive方法，带int类型参数2个
     mo.receive(3.14159); //调用receive方法，带double类型参数1个
     mo.receive("It is a Test!"); //调用receive方法，带string型参数1
   个
    }
   }
```

运行结果：



3.对象间的类型转换

对象状态的确定（instanceof)

- Manager和Contractor都是Employee的子类
- 使用instanceof来测试一个对象的类型。

```
public void method(Employee e) {
    if  (e instanceof  Manager)
        Manager m = (Manager)e;
        System.out.println( " This is the manager of " +
m.department);
    }   else if ( e instanceof Contractor) {
            …
    } //do something as a Contractor
}
```

3.对象间的类型转换

- 用类型转换来恢复一个对象的全部功能。
- 向上类型转换是将子类对象强制类型转换为父类类型，这总是允许的，而且不需要强制类型转换运算符。
- 对于向下类型转换（将父类类型强制转换为子类类型），编译器必须满足类型转换至少是可能的这样的条件。比如，任何将Manager转换成Contractor引用的尝试是肯定不允许的，因为Contractor不是一个Manager。类型转换发生的类必须是当前引用类型的子类。

3.对象间的类型转换

对象类型转换实例

```
class Base {
    public String str = "hello Base...";
    public void hello(){
        System.out.println(str);
    }
}
class Sub extends Base{
    public String str = "hello Sub...";
    public void hello(){
        System.out.println(str);
    }
}
```

```
public class two{
    public static void main(String[] args) {
        Base base = new Base();
        base.hello();              //  hello Base...

        Sub sub = new Sub();
        sub.hello();               // hello Sub...

        Base bs = new Sub();
        bs.hello();                // hello Sub...

        Base s = new Sub();
        Sub s1=(Sub)s;
        s1.hello();                // hello Sub...

        Sub sb = (Sub) new Base();
        sb.hello();                //// error

    }
}
```

课堂习题

Given the following code

```
class Base {}
class Ag extends Base{
      public String getFields(){
       String name =  "Ag";
      return name;
      }
}
public class Avf{
    public static void main(String argv[]){
      Base a = new Ag();
      //Here
    }
}
```

下面哪行代码放在注释处，可以输出 "Ag"?

1) System.out.println(a.getFields());
2) System.out.println(a.name);
3) System.out.println((Base) a.getFields());
4) System.out.println(((Ag) a).getFields());

Thank you!