笔记总链接：http://bbs.itheima.com/thread-200600-1-1.html

4、继承

4.4 子类的实例化过程

子类中所有的构造函数默认都会访问父类中空参数的构造函数。

因为每一个构造函数的第一行都有一条默认的语句super();。

为什么子类实例化的时候要访问父类中的构造函数呢？

那是因为子类继承了父类，获取到了父类中内容（属性），所以在使用父类内容之前，要先看父类是如何对自己的内容进行初始化的。

P.S.

1、当父类中没有空参数的构造函数时，子类的构造函数必须通过this或者super语句指定要访问的构造函数。

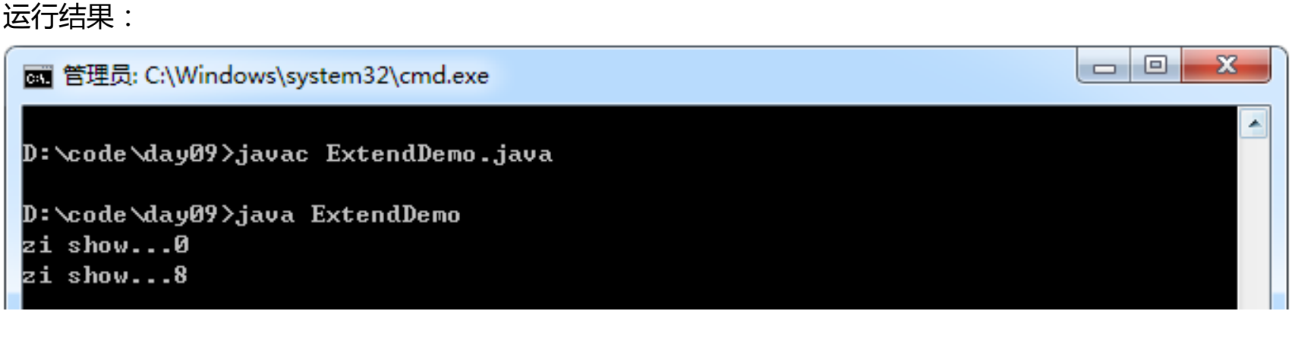
2、子类构造函数中如果使用this调用了本类构造函数，那么默认的super();就没有了，因为super和this都只能定义在第一行，所以只能有一个。但是可以保证的是，子类中肯定会有其他的构造函数访问父类的构造函数。

3、super语句必须要定义在子类构造函数的第一行！因为父类的初始化动作要先完成。

示例：

```
class Fu{
        Fu(){
                super();
                //调用的是子类的show方法，此时其成员变量num还未进行显示初始化
                show();
                return;
        }
        void show(){
               System.out.println("fu show" );
        }
}
class Zi extends Fu{
         int num = 8;
         Zi(){
                super();
                //通过super初始化父类内容时，子类的成员变量并未显示初始化，等super()父类初始化完毕后，才进行子类的成员变量显示初始化
                return;
         }
         void show(){
               System.out.println("zi show..." + num);
         }
}
class ExtendDemo{
         public static void main(String[] args){
               Zi z = new Zi();
               z.show();
         }
}
```
复制代码

运行结果：



管理员: C:\Windows\system32\cmd.exe

```
D:\code\day09>javac ExtendDemo.java

D:\code\day09>java ExtendDemo
zi show...0
zi show...8
```

总结：

一个对象实例化过程，以Person p = new Person();为例：

1. JVM会读取指定的路径下的Person.class文件，并加载进内存，并会先加载Person的父类（如果有直接的父类的情况下）。
2. 在内存中开辟空间，并分配地址。
3. 并在对象空间中，对对象的属性进行默认初始化。
4. 调用对应的构造函数进行初始化。
5. 在构造函数中，第一行会先到调用父类中构造函数进行初始化。
6. 父类初始化完毕后，再对子类的属性进行显示初始化。
7. 再进行子类构造函数的特定初始化。
8. 初始化完毕后，将地址值赋值给引用变量。

4.5 final关键字

final可以修饰类，方法，变量。

final修饰的类不可以被继承。

final修饰的方法不可以被覆盖。

final修饰的变量是一个常量，只能被赋值一次。

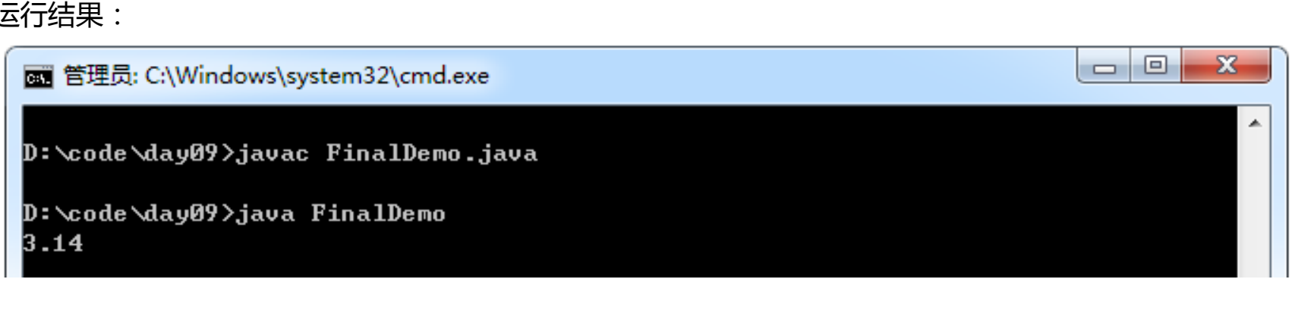
为什么要用final修饰变量，其实，在程序中如果一个数据是固定的。那么直接使用这个数据就可以了，但是这种阅读性差，所以应该给数据起个名称。而且这个变量名称的值不能变化，所以加上final固定。

写法规范：常量所有字母都大写，多个单词，中间用_连接。

示例1：

```
//继承弊端：打破了封装性
class Fu{
      void method(){
      }
}

class Zi extends Fu{
      public static final double PI = 3.14;
      void method(){
          System.out.println(PI);
      }
}

class FinalDemo{
      public static void main(String[] args){
          Zi zi = new Zi();
          zi.method();
      }
}
```
复制代码

运行结果：



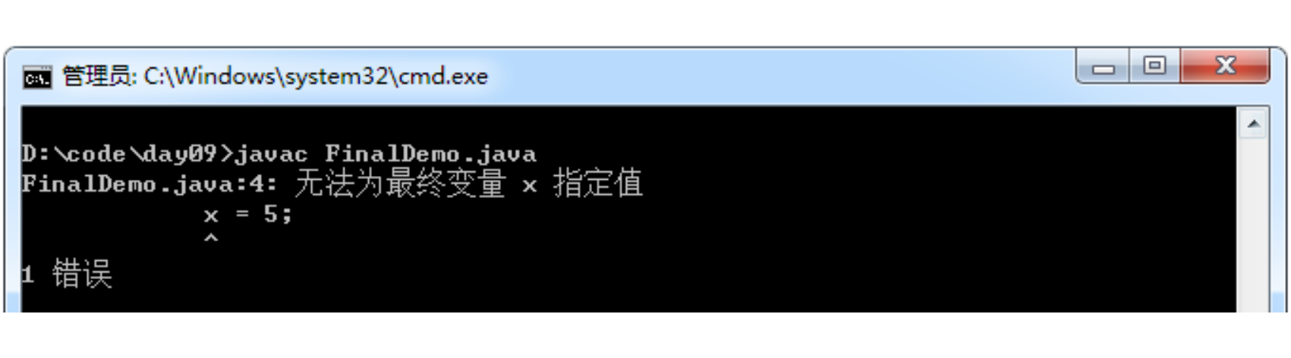
管理员: C:\Windows\system32\cmd.exe

```
D:\code\day09>javac FinalDemo.java

D:\code\day09>java FinalDemo
3.14
```

示例2：

```
class FinalDemo{
         public static void main(String[] args){
              final int x = 4;
              x = 5;
         }
}
```
复制代码

运行结果：



管理员: C:\Windows\system32\cmd.exe

```
D:\code\day09>javac FinalDemo.java
FinalDemo.java:4: 无法为最终变量 x 指定值
               x = 5;
               ^
1 错误
```

4.6 抽象类

4.6.1 抽象类概述

抽象定义：

抽象就是从多个事物中将共性的、本质的内容抽取出来。

例如：狼和狗共性都是犬科，犬科就是抽象出来的概念。

抽象类：

Java中可以定义没有方法体的方法，该方法的具体实现由子类完成，该方法称为抽象方法，包含抽象方法的类就是抽象类。

抽象方法的由来：

多个对象都具备相同的功能，但是功能具体内容有所不同，那么在抽取过程中，只抽取了功能定义，并未抽取功能主体，那么只有功能声明，没有功能主体的方法称为抽象方法。

例如：狼和狗都有吼叫的方法，可是吼叫内容是不一样的。所以抽象出来的犬科虽然有吼叫功能，但是并不明确吼叫的细节。

4.6.2 抽象类的特点

抽象类和抽象方法必须用abstract关键字来修饰。

抽象方法只有方法声明，没有方法体，定义在抽象类中。

格式：修饰符 abstract 返回值类型 函数名(参数列表)；

抽象类不可以被实例化，也就是不可以用new创建对象。

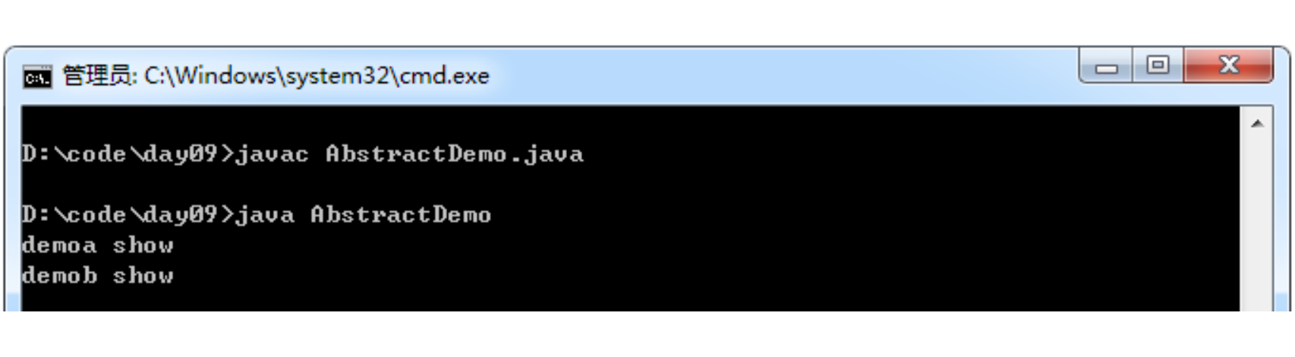
原因如下：

1. 抽象类是具体事物抽取出来的，本身是不具体的，没有对应的实例。例如：犬科是一个抽象的概念，真正存在的是狼和狗。
2. 而且抽象类即使创建了对象，调用抽象方法也没有意义。
3. 抽象类通过其子类实例化，而子类需要覆盖掉抽象类中所有的抽象方法后才可以创建对象，否则该子类也是抽象类。

示例：

```
abstract class Demo{
      abstract /*抽象*/ void show();
}

class DemoA extends Demo{
      void show(){
          System.out.println("demoa show" );
      }
}

class DemoB extends Demo{
      void show(){
          System.out.println("demob show" );
      }
}

class AbstractDemo{
      public static void main(String[] args){
          DemoA demoA = new DemoA();
          demoA.show();

          DemoB demoB = new DemoB();
          demoB.show();
      }
}
```
复制代码

运行结果：



管理员: C:\Windows\system32\cmd.exe

```
D:\code\day09>javac AbstractDemo.java

D:\code\day09>java AbstractDemo
demoa show
demob show
```

4.6.3 抽象类举例代码讲解

需求：

公司中程序员有姓名，工号，薪水，工作内容。

项目经理除了有姓名，工号，薪水，还有奖金，工作内容。

分析：

在这个问题领域中，通过名词提炼法：

程序员：

属性：姓名，工号，薪水。

行为：工作。

经理：

属性：姓名，工号，薪水，奖金。

行为：工作。

程序员和经理不存在着直接继承关系。

但是，程序员和经理却具有共性内容，可以进行抽取，因为他们都是公司的雇员。

可以将程序员和经理进行抽取，建立体系。

代码：

```
//描述雇员。
abstract class Employee{
      private String name ;
      private String id ;
      private double pay ;

     Employee(String name,String id, double pay){
           this.name = name;
           this.id = id;
           this.pay = pay;
     }

      public abstract void work();
}

//描述程序员
class Programmer extends Employee{
     Programmer(String name,String id, double pay){
           super(name,id,pay);
     }

      public void work(){
          System.out.println("code..." );
      }
}

//描述经理
class Manager extends Employee{
     private int bonus ;

     Manager(String name,String id, double pay,int bonus){
           super(name,id,pay);
           this.bonus = bonus;
     }

     public void work(){
          System.out.println("manage" );
     }
}
```
复制代码

4.6.4 抽象类相关问题

抽象类中是否有构造函数？

答：有，用于给子类对象进行初始化。

抽象关键字abstract不可以和哪些关键字共存？

答：private、static、final。

抽象类中可不可以没有抽象方法？

答：可以，但是很少见。目的就是不让该类创建对象，AWT的适配器对象就是这种类。通常这个类中的方法有方法体，但是却没有内容。

示例：

```
abstract class Demo{
    void show1(){}
    void show2(){}
}
```
复制代码

抽象类和一般类的区别？

答：

相同点：

抽象类和一般类都是用来描述事物的，都在内部定义了成员。

不同点：

1. 一般类有足够的信息描述事物。
   抽象类描述事物的信息有可能不足。
2. 一般类中不能定义抽象方法，只能定义非抽象方法。
   抽象类中可定义抽象方法，同时也可以定义非抽象方法。
3. 一般类可以被实例化。
   抽象类不可以被实例化。

抽象类一定是个父类吗？

答：是的，因为需要子类覆盖其方法后才可以对子类实例化。

4.7 接口

当一个抽象类中的方法都是抽象的时候，这时可以将该抽象类用另一种形式定义和表示，就是接口。

格式：interface {}

接口中的成员修饰符是固定的：

成员常量：public static final

成员函数：public abstract

由此得出结论，接口中的成员都是公共的权限。

接口是对外暴露的规则。

接口是程序的功能扩展。

P.S.

1、虽然抽象类中的全局变量和抽象方法的修饰符都可以不用写，但是这样阅读性很差。所以，最好写上。

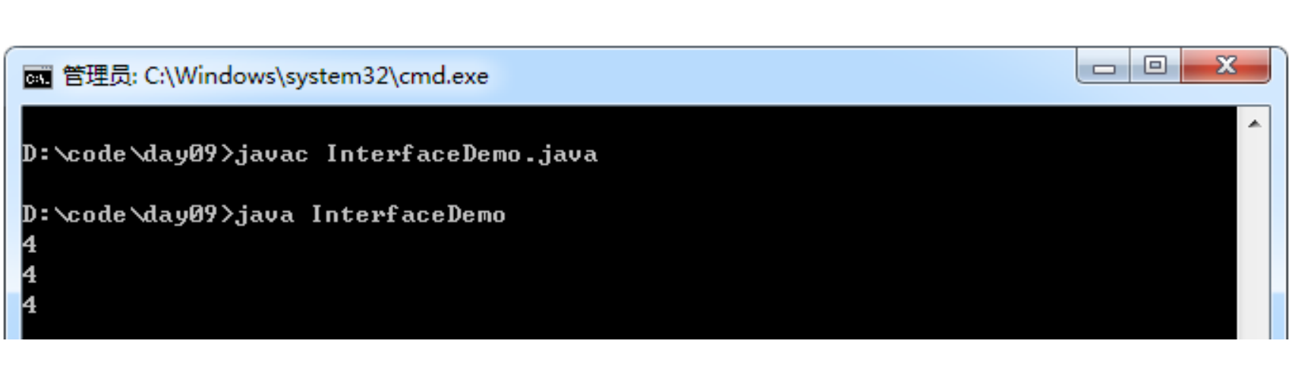
2、类与类之间是继承关系，类与接口直接是实现关系。

3、接口不可以实例化，能由实现了接口并覆盖了接口中所有的抽象方法的子类实例化。否则，这个子类就是一个抽象类。

示例：

```
interface Demo{
         public static final int NUM = 4;
         public abstract void show1();
         public abstract void show2();
}

class DemoImpl implements /*实现*/Demo{
         public void show1(){}
         public void show2(){}
}

class InterfaceDemo{
         public static void main(String[] args){
              DemoImpl d = new DemoImpl();
              System.out.println(d.NUM);
              System.out.println(DemoImpl.NUM);
              System.out.println(Demo.NUM);
        }
}
```
复制代码

运行结果：



管理员: C:\Windows\system32\cmd.exe

```
D:\code\day09>javac InterfaceDemo.java

D:\code\day09>java InterfaceDemo
4
4
4
```

接口的出现将“多继承”通过另一种形式体现出来，即“多实现”。

在java中不直接支持多继承，因为会出现调用的不确定性。

所以，java将多继承机制进行改良，在java中变成了多实现，一个类可以实现多个接口。

接口的出现避免了单继承的局限性。

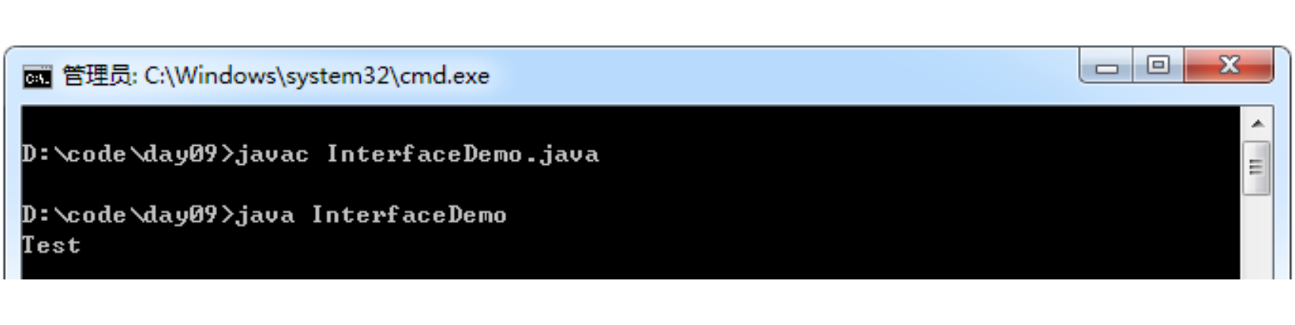
示例：

```
interface A{
         public void show();
}

interface Z{
         public void show();
}

//多实现
class Test implements A,Z{
         public void show(){
```

```
                System.out.println("Test");
        }
}

class InterfaceDemo{
        public static void main(String[] args){
                Test t = new Test();
                t.show();
        }
}
```

复制代码

运行结果：



```
D:\code\day09>javac InterfaceDemo.java

D:\code\day09>java InterfaceDemo
Test
```

一个类在继承另一个类的同时，还可以实现多个接口。

示例1：

```
interface A{
      public void show();
}

interface Z{
      public void show();
}

class Q{
    public void method(){
    }
}

abstract class Test2 extends Q implements A,Z{

}
```

复制代码

示例2：

```
interface CC{
      void show();
}

interface MM{
      void method();
}

//接口与接口之间是继承关系，而且接口可以多继承
interface QQ extends CC,MM{
      public void function();
}

class WW implements QQ{
      //覆盖3个方法
      public void show(){}
      public void method(){}
      public void function(){}
}
```

复制代码

抽象类和接口的异同点？

相同点：

都是不断向上抽取而来的。

不同点：

1. 抽象类需要被继承，而且只能单继承。

   接口需要被实现，而且可以多实现。

2. 抽象类中可以定义抽象方法和非抽象方法，子类继承后，可以直接使用非抽象方法。

   接口中只能定义抽象方法，必须由子类去实现。

3. 抽象类的继承，是is a关系，定义该体系的基本共性内容。

   接口的实现是like a关系。

~END~

~爱上海，爱黑马~