JAVA基础知识（高级类的特性） (2012-12-11 13:22:13)转载▼

标签： 杂谈 分类： JAVA基础

一、static关键字

在Java中，可以将一些成员限制为和类相关的，而不是和实例相关的。实现这个目的的方法是，在类的成员变量例如方法、属性乃至自有代码块前面加上“static”这个修饰符。

实例变量用于描述单个实例的状态，而类变量用于描述类的状态，和具体的实例无关。实例方法用于操作实例变量并维护特定实例的状态，而类方法步步高不要了解单个实例的状态。

在Java中，static最常用到的地方是用在Java应用程序的入口方法main()前面用于修饰main()方法，表示main方法是一个类方法。我们在执行一个应用程序时，解析器将会去寻找这个main()方法，之所以将他设置为类方法，是因为在调用这个main()方法之前，还没有任何对象存在，所以如果不将它设置为类相关的，则没有办法的到任何对象，也就无法调用这个main()方法。

用static修饰的方法中调用的变量只能是static类的变量，而static类型的变量可以用在非static的方法中。虽然static修饰的是类成员，但是同样可以通过实例名来调用static修饰的类成员，就像调用其他的实例成员一样。

static自由块

用static修饰的自由块（在方法体和构造器之外包含一段用“{}”括起来的代码段，这个代码段称为“自由块”或者“游离块”），自由块可以看成一种特殊的方法，这个方法没有方法名，没有输入参数，没有返回值，不能进行方法调用。从这个角度看，在自由块前面也可以加上static关键字来修饰。这样，这个自由块就变成了静态自由块。静态自由块通常用来初始化静态变量。

二、Singleton(单例)设计模式

所谓设计模式（Design Pattern）就是为了满足对优秀、简单而且可重用的解决方案的需要。这就像我们在盖楼的时候，我们不会每次都从零开始来画图纸，而是参考某种已有的模式，然后在其基础上来设计他。

Singleton设计模式的需求背景。在实际应用中，我们可能需要限制生成的对象个数只能为一个。例如，在开发图形应用的时候，在菜单上按下“帮助”菜单，将弹出一个帮助对话框，也就是生成了一个帮助对话框的实例，如果对话框已经出现，则即使再在菜单栏上按下“帮助”菜单，应用程序也不应该再生成新的对话框实例。这个时候，如果还是通过原来的用构造器来生成实例的方式，很难对其生成的实例个数进行控制。通过所谓的“单例(Singleton)设计模式”，可以实现这个功能。

单例设计模式的基本思路是在类的内部定义一个静态变量，当创建第一个实例时设置该变量。应用程序直接使用这个静态变量指向的实例就可以了。为了防止应用程序调用构造器来新建实例，必须限制构造器的访问，也就是将他的访问控制设置为“private”。

概括起来，要实现“单例设计模式”，可以利用下面的方式来完成。

构建一个类，满足：

（1）构造器私有化

（2）用一个私有、静态变量引用实例

（3）提供一个公有、静态方法来获得实例

下面是一个单例设计模式的实现(方法一)：

public class SingletonPattern{

//定义一个私有、静态的引用变量

private static SingletonPattern sp;

//构造器私有

private SingletonPattern(){

}

//提供一个公有、静态方法获得唯一实例

public static SingletonPattern getInstance(){

if(sp==null){//如果还未创建实例

sp = new SingletonPattern();//创建一个实例

}

return sp;//将它返回

}

}

单例实现方法二：

public class SingletonPattern{

//定义一个私有的、静态的引用变量，并实例对象

private static SingletonPattern sp = new SingletonPattern();

//构造器私有化

private SingletonPattern(){

}

//提供一个公有的静态方法获得唯一实例

public static SingletonPattern getInstance(){

return sp;

}

}

三、final关键字

在Java的类、方法和变量前面，可以加上final关键字。final关键字表示他修饰的类、方法或者变量不可改变。

1、final类

final类不可以被继承，例如java.lang.Math。如果你不希望你的类被继承，可以将类定义为final类，这样别的类就无法继承它。

2、final变量

final变量表示该变量不可以被改变，其实就是常量。因为在Java中不能使用const保留字来定义常量，所以在需要使用常量的场合，可以使用final关键字来修饰。例如：

final int CONST\_VAR=12；

如果final变量是简单类型的变量，那么变量的值一旦初始化后，就不能被改变，如果final变量是引用类型的变量，那么你不能改变这个变量的引用，但可以改变这个变量所引用的对象的属性。

示例：final引用变量的用法：

源文件：TestFinal.java

class Aclass{

private int a;

//构造器

public Aclass（）{

a=100;

}

public void setA(int theA){

a=theA;

}

public int getA(){

return a;

}

}

//定义一个类来测试final应用变量

public class TestFinal{

final Aclass aClass = new Aclass();

public static void main(String[] args){

TestFinal tf = new TestFinal();

System.out.println(tf.aClass.getA());//输出100

tf.aClass.setA(1);

System.out.println(tf.aClass.getA());//输出1

}

}

提示：根据Java命名规范，如果在编译的时候，就可以确定final变量的值，那么此常量的名字应该都大写，并且指出完整含义，如果需要用几个单词描述，那么单词之间用“\_”来分割，例如：

static final int MAX\_TAX\_RATE=20；

相反的，在下面情况下可以不使用上面的命名规则：final变量无法在编译的时候就获得值，而必须在执行的时候才能得到值。例如，我们在上述示例的final变量Aclass就是一个例子：

final Aclass aClass = new Aclass();

在编译的时候，程序不会执行new 动作来实例化一个Aclass，并赋值给aClass，所以就没有使用上面的命名规则来给final变量命名。

3、final方法

final方法表示这个方法不可以在子类中覆盖。例如，下面的例子中，类B编译不会通过，因为它试图覆盖类中的一个final方法。

父类：

public class A{

public final void methodA(){};

}

子类

public class B extends A{

//此处报错，因为它试图覆盖父类中的final方法

public void methodA(){};

}

如果将父类A中的methodA()改成private的，如下：

父类：

public class A{

private final void methodA(){};

}

子类：

public class B extends A{

//此处不报错，因为它只是定义了一个methodA()

public void methodA(){};

}

注意类A的methodA（），它的访问修饰符为private，此时，在子类B中是访问不了父类A的methodA（）方法，因此，在类B中的methodA（）只是新定义的一个方法而不是覆盖父类的方法。

四、抽象(abstract)类

1、abstract概述

在我们编写一个对象的时候，我们会对一些类的方法进行定义，但是并不具体实现，而是将这些方法的实现放到子类中去，这样可以增强类设计的灵活性。这个时候，我们将类定义为abstract的，而将没有实现的方法声明为abstract的。因为abstract类中没有被实现的方法，所有，抽象类不能被实例化。

2、abstract class

抽象类里面并非一定需要抽象方法，当你定义一个类，但是又不想让它直接实例化的时候，可以使用抽象类的方法来实现。反之，如果类中有抽象方法，则一定要将类定义为抽象的类。

在以下任一条件成立时，类必须定义为抽象类：

（1）、类中至少有一个抽象方法

（2）、类继承了父类中的抽象方法，但至少有一个抽象方法没有实现。

（3）、类实现了某个接口，但是没有全部实现接口中的方法。

public abstract class Shape{

public abstract double perimeter();//抽象方法

区分没有实现的方法和空方法体，

public int methodB(){};//空方法体

public int methodA();//没有实现的方法

没有实现的方法可以用abstract来修饰，而空方法体方法却不能使用abstract来修饰，它已经实现的方法，只是在这个方法中什么动作也没有做而已。

}

五、接口(interface)

1、在JAVA中除了可以定义抽象类来实现一定程度的抽象外，还可以定义一种中特殊的“抽象类”--接口(interface)。接口是方法声明和常量值的定义的集合。在抽象类中，有些方法被实现，而有些方法只有方法的声明，而有些方法只有方法的声明，没有方法的具体实现(抽象方法)，而在接口中，所有的方法都没有被实现。和抽象类中的方法不一样，这些没有被实现的方法不需要加上abstract来将他们声明为抽象方法。

2、接口的定义：和类的定义不同，接口的定义不使用关键字"class"，而是使用另一个关键字"interface"，接口定义的基本格式如下:

interface [extends ]{

[]

[]

}

在接口中，可以定义属性，但它的成员属性都会被自动加上public、static和final，而对于接口中的方法，也会自动将它设置为public的。虽然我们可以显示的在属性前面加上public static final 来修饰它，或者在方法定义前加上public 来修饰，但在JAVA编程规范中，并不推荐这样做。下面来看一个接口定义。

public interface VideoD{

public int MIN\_B=10;

void turnOn();

}

3、接口的继承

接口和类相似，也可以由一个父接口派生。接口的继承也是使用"extends"关键字来完成的。接口继承的基本语法如下:

interface

[extends ,]

{

[]

[]

}

和类继承相似，继承的接口也具有了父接口的特性。和类的继承不同的就是，在接口的继承中，可以继承多个接口，只需要将继承的各个接口用逗号“，”分隔开就可以了。

六、多态

1、多态描述

在JAVA中，变量对象是多态的。一个类型为Aclass的变量既可以指向类型为Aclass的对象，又可以指向Aclass的任何子类对象。

假设有一个类Aclass，由它派生的类Bclass，那么，我们可以将子类Bclass的对象赋给父类变量：

Aclass a = new Bclass();

2、虚方法的调用

在多态的情况下，声明为父类的类型的引用变量只能访问父类中定义过的方法，但如果此变量实际引用的是子类的对象，而子类中有进行了方法覆盖时，实际调用的是覆盖后的方法，这种机制被称为虚方法调用。

3、对象造型

各种简单数据类型之间是可以进行互相转换的，有些转换可以通过系统自动完成，而有些转换必须在程序中通过强制转换来完成。而对于引用类型，也有一个相互转换的机制。同样的，在引用类型数据进行转换的时候，分为自动造型和强制造型两种情况。

当从子类转换成父类的时候(或者实现类转换成接口)，造型可以自动完成。

当从父类转换成子类的时候(或者接口转换成实现类)，必须使用强制造型。

需要注意的是，无论是自动造型还是强制造型，都只能在有继承关系的对象之间。并不是任意的父类类型数据都可以被造型为子类类型，只有在多态情况下，原本就是子类类型的对像被声明为父类的类型，才可以通过造型来恢复“真实面目”，否则会在程序运行时出错。

在造型时的时候，往往需要使用instanceof来判断一个对象是否可以进行造型，以避免运行时错误。

例如，Person类有两个子类，Teacher和Student，还有一个单独的类Test，我们来看下面的代码，

Test t = new Test();

Person p = new Student();

Teacher t = (Teacher)p;//将会出现运行时错误

Student s = (Student)t;//因为Test类和Student类之间没有任何的继承关系

//所以在编译的时候就会出错(编译错误，compile error)

此时，可以先使用instanceof来判断要造型的对象是否为可以造型的类型，例如：

Person p = new Student();

if(p instanceof Teacher){

Teacher t = (Teacher)p;

}

http://blog.sina.com.cn/s/blog\_9e62b0a501013klk.html