抽象类和接口和Lambda

内部类

匿名内部类也就是没有名字的内部类,匿名内部类就是重写父类或接口的方法。

正因为没有名字，所以匿名内部类只能使用一次，它通常用来简化代码编写

但使用匿名内部类还有个前提条件：必须继承一个抽象类或实体类或接口

实例1:不使用匿名内部类来实现抽象方法

```
abstract class Person {
    public abstract void eat();
}

class Child extends Person {
    public void eat() {
        System.out.println("eat something");
    }
}

public class Demo {
    public static void main(String[] args) {
        Person p = new Child();
        p.eat();
    }
}
```

运行结果：eat something

可以看到，我们用Child继承了Person类，然后实现了Child的一个实例，将其向上转型为Person类的引用

但是，如果此处的Child类只使用一次，那么将其编写为独立的一个类岂不是很麻烦?

这个时候就引入了匿名内部类

实例2：匿名内部类的基本实现

```
abstract class Person {
    public abstract void eat();
}

public class Demo {
    public static void main(String[] args) {
        Person p = new Person() {
            public void eat() {
                System.out.println("eat something");
            }
        };
        p.eat();
    }
}
```

运行结果：eat something

可以看到，我们直接将抽象类Person中的方法在大括号中实现了

这样便可以省略一个类的书写

并且，匿名内部类还能用于接口上

实例3：在接口上使用匿名内部类

```
interface Person {
    public void eat();
}

public class Demo {
    public static void main(String[] args) {
        Person p = new Person() {
            public void eat() {
                System.out.println("eat something");
            }
        };
        p.eat();
    }
}
```

运行结果：eat something

由上面的例子可以看出，只要一个类是抽象的或是一个接口，那么其子类中的方法都可以使用匿名内部类来实现

最常用的情况就是在多线程的实现上，因为要实现多线程必须继承Thread类或是继承Runnable接口

实例4：Thread类的匿名内部类实现

```
public class Demo {
    public static void main(String[] args) {
        Thread t = new Thread() {
            public void run() {
                for (int i = 1; i <= 5; i++) {
                    System.out.print(i + " ");
                }
            }
        };
        t.start();
    }
}
```

运行结果：1 2 3 4 5

实例5：Runnable接口的匿名内部类实现

```
public class Demo {
    public static void main(String[] args) {
        Runnable r = new Runnable() {
            public void run() {
                for (int i = 1; i <= 5; i++) {
                    System.out.print(i + " ");
                }
            }
        };
        Thread t = new Thread(r);
        t.start();
    }
}
```

运行结果：1 2 3 4 5

为什么引入抽象类?

抽象类：几何图形class Figure

子类: 矩形class Rectangle 圆class Circle三角形class Triangle 等... 属性: 曲边 ,直边, 几条边, 求面积, 求周长...

操作系统(2)
Java(2)
更多

随笔分类
Java基础(10)
数据结构与算法(21)

随笔档案
2017年12月 (9)
2017年11月 (26)

积分与排名
积分 - 563
排名 - 124374

如果没有抽象类,父类提取出来的共性必须是唯一的,实现的,稳定的,那么求面积,求周长,都不能提取,当一个方法的功能是处理几何图形,参数是Figure f .当接收一个图形时,求其周长还要事先把 f 向下转型,很是麻烦.但是如果引入抽象类,可以把求周长的公式提取出来,但是不必实现,当然也无法实现.这样就不用向下转型了.增加了灵活性,而且子类继承了抽象类,必须把抽象类的所有抽象方法必须全实现,才能new实例,这样抽象类带来了另外一个好处:模板,规定子类必须要干什么!

抽象类原则

(1)抽象类用abstract修饰,抽象类不能new出实体

(2)抽象类提取出的共性,能实现的直接实现,不能实现的用abstract修饰,不必实现,由子类继承重写

(3)抽象类应该尽可能提取最多的共性,不用在乎能不能实现.

(4)抽象类不能new出实体,就是用来且必须被继承的,继承者必须重写其全部的抽象方法,才能new出实体,否则,继承者也是个抽象类.

(5)abstract修饰的类被继承才有意义,abstract修饰的方法被重写才能new出实体,而final修饰的类不能被继承,方法不能被重写,所以abstract和final不能同时使用.

(6)private修饰的方法子类无法触及,所以private和abstract也不通用.

(7)抽象类也有构造器,但是这个构造器并不是为了new出对象,而是给继承者的构造器调用的.

```
抽象方法和普通方法 形式上的区别:
public abstruct void abstructFun();
public          void fun() {}
```

接口

```
/*接口的权限是public或friendly,能继承一个或多个接口,不能继承类,不含构造器,不含初始化块*/
interface JieKou extends 接口1,接口2... {
   /* 接口的所有成员都是public ,就像C语言的const修饰的变量,不再改变,就是"模板" */
   public static final int VAR = 20;
   /*static 省空间,final 是规范就不要再修改,必须定义时直接初始化*/
   //int VAR = 20;
   /*普通方法必须是抽象的 public abstruct,不用实现 */
   void f();
   /*类方法 public static ,必须实现 */
   static void g() {
      System.out.println("我是类方法");
   }
   /*默认方法也必须实现 public */
   default void print(String... msgs) {
      for(String msg : msgs) {
         System,out.println(msg);
      }
   }
   /*其他成员:内部类,内部接口,枚举类*/
}
```

接口不能new出实体,接口可以作为引用变量,指向类new出的实体.

<1>定义引用变量,也可以进行强制类型转换

<2>调用接口的常量

<3>被其他类实现

一个类可以多继承很多接口,但是只能继承一个类.

class extends 类 implements 接口1,接口2... {

}

一个类继承了接口,必须实现接口中所有的抽象方法,不然仍旧只能当抽象类用.

继承接口的类重写接口的抽象方法时,权限只能是public,因为子类重写父类时,只能用相同或者更大的权限.

抽象类和接口的区别

接口和抽象类很像，它们都具有如下特征。

- 接口和抽象类都不能被实例化，它们都位于继承树的顶端，用于被其他类实现和继承。
- 接口和抽象类都可以包含抽象方法，实现接口或继承抽象类的普通子类都必须实现这些抽象方法。

但接口和抽象类之间的差别非常大，这种差别主要体现在二者设计目的上。下面具体分析二者的差别。

接口作为系统与外界交互的窗口，接口体现的是一种规范。对于接口的实现者而言，接口规定了实现者必须向外提供哪些服务（以方法的形式来提供）；对于接口的调用者而言，接口规定了调用者可以调用哪些服务，以及如何调用这些服务（就是如何来调用方法）。当在一个程序中使用接口时，接口是多个模块间的耦合标准；当在多个应用程序之间使用接口时，接口是多个程序之间的通信标准。

从某种程度上来看，接口类似于整个系统的“总纲”，它制定了系统各模块应该遵循的标准，因此一个系统中的接口不应该经常改变。一旦接口被改变，对整个系统甚至其他系统的影响将是辐射式的，导致系统中大部分类都需要改写。

抽象类则不一样，抽象类作为系统中多个子类的共同父类，它所体现的是一种模板式设计。抽象类作为多个子类的抽象父类，可以被当成系统实现过程中的中间产品，这个中间产品已经实现了系统的部分功能（那些已经提供实现的方法），但这个产品依然不能当成最终产品，必须有更进一步的完善，这种完善可能有几种不同方式。

除此之外，接口和抽象类在用法上也存在如下差别。

- 接口里只能包含抽象方法和默认方法，不能为普通方法提供方法实现；抽象类则完全可以包含普通方法。
- 接口里不能定义静态方法；抽象类里可以定义静态方法。
- 接口里只能定义静态常量，不能定义普通成员变量；抽象类里则既可以定义普通成员变量，也可以定义静态常量。
- 接口里不包含构造器；抽象类里可以包含构造器，抽象类里的构造器并不是用于创建对象，而是让其子类调用这些构造器来完成属于抽象类的初始化操作。
- 接口里不能包含初始化块；但抽象类则完全可以包含初始化块。
- 一个类最多只能有一个直接父类，包括抽象类；但一个类可以直接实现多个接口，通过实现多个接口可以弥补 Java 单继承的不足。

接口的使用:

某个方法需要完成一个行为,但是这个行为暂时不知道是什么,只有执行方法时,才知道.比如,一个方法时处理数组,遍历数组的时候,对数组元素执行什么操作暂时不确定,需要临时规定行为,解决方法就是定义一个接口.

```
1 import java.util.*;
2 public class TestJava {
3     public static void main(String[] args) {
4         int[] a = new int[10];
5         for(int i = 0;i < a.length;++i)
6             a[i] = i;
7 
8         for(int i = 0;i < a.length;++i)
9             System.out.print(a[i]);
10 
11         TestFun tf = new TestFun();
12         tf.f(a,new Set0());
13 
14         for(int i = 0;i < a.length;++i)
15             System.out.print(a[i]);
16 
17         tf.f(a,new Set1());
18         for(int i = 0;i < a.length;++i)
19             System.out.print(a[i]);
20     }
21 }
22 
23 interface JieKou {
24     void set(int[] a);
25 }
```

```
26
27 class Set0 implements JieKou {
28     public void set(int[] a) {
29         for(int i = 0;i < a.length;++i) {
30             a[i] = 0;
31         }
32     }
33 }
34
35 class Set1 implements JieKou {
36     public void set(int[] a) {
37         for(int i = 0;i < a.length;++i) {
38             a[i] = 1;
39         }
40     }
41 }
42
43 class TestFun {
44     void f(int[] a,JieKou jk) {
45         jk.set(a);
46     }
47 }
```

Lambda表达式

如何动态的传入一段代码,作为具体的处理行为?

(1) 动态处理代码封装成一个接口a的抽象方法.定义一个继承接口的类a,重写抽象方法.传入类,在使用动态处理代码的方法中通过类名.方法使用代码.

(2)直接把接口的引用作为方法参数,void f(int [ ] a, new JieKou { 重写抽象方法}); 匿名内部类

(3)使用Lambda表达式

Lambda表达式支持将代码块作为方法参数,Lmabda表达式可以更简洁的创建函数式接口的一个实体.

函数式接口: 可以有多个默认方法,类方法,只有一个抽象方法的接口叫函数式接口.

使用匿名内部类.

```
 1 import java.util.*;
 2 public class TestJava {
 3     public static void main(String[] args) {
 4         int[] a = new int[10];
 5
 6       TestFun tf = new TestFun();
 7       tf.f(a,new JieKou() {
 8           public void set(int[] a) {
 9               for(int i = 0;i < a.length;++i)
10                 a[i] = 2;
11           }
12         });
13      for(int i = 0;i < a.length;++i)
14             System.out.print(a[i]);
15     }
16 }
17
18 interface JieKou {
19     void set(int[] a);
20 }
21
22 class TestFun {
23     void f(int[] a,JieKou jk) {
24         jk.set(a);
25     }
26 }
```

使用Lambda表达式

Lambda表达式:只有一个形参时,可以省略圆括号,只有一条语句时,可以省略{ }, 若方法体是 c = a + b; return c; 可以简写成 c = a + b; 它会自动返回.

Lambda与匿名内部类的相同点和区别

```
interface Displayable {
    void Display(int i);
    default void f() {
        System.out.println("我是接口的默认方法");
    }
}

public class TestLambda {
    private String name = new String("哈哈哈");
    void test() {
        Displayable dis = i->{
        System.out.println("我是接口的抽象方法,我要显示"+i);
        System.out.println("我要访问name,name是:"+name);
      };
      dis.f();
      dis.Display(3);
    }

    public static void main(String[] args) {
        TestLambda tl = new TestLambda();
        tl.test();
    }
}
```

```
问题: 方法A需要处理一个数组,怎样处理临时起意,不确定.
(1)把处理方法封装成接口
interface JieKou {
    处理方法process;
}
(2)写方法A
A(数组引用,JieKou引用)
(3)使用方法A
     1.匿名内部类
     A(数组引用,接口引用() {重写处理方法process});
     2.Lambda表达式
     A(数组引用,(process的参数列表)->{方法体})
两种方法都避免了为了一个动态处理代码块创建一个类,优化了项目.
注意:Lambda表达式返回一个引用.接口是函数式接口时,才能用Lambda表达式简写.
Lambda表达式就是为函数式接口创建实体的.
Runnable是个函数式接口,只含有一个抽象方法
Runnable r = ()->{System.out.println("Lambda表达式实现函数式接口的实体")};
Object r = ()->{System.out.println("Lambda表达式实现函数式接口的实体")};//error
因为Object不是一个函数式接口,用函数式接口强制类型转换后可以用Object引用指向.
Object r = (Runnable)()->{System.out.println("Lambda表达式实现函数式接口的实体")};//OK
将Lambda表达式赋值给一个函数式接口的引用变量
将Lambda表达式作为函数式接口的形参
使用函数式接口对Lamable表达式进行强制类型转换.
```

匿名内部类是接口就能创建实体,但是Lambda表达式只能为函数式接口创建实体.

匿名内部类还可以为抽象类,实体类创建实体.

匿名内部类在实现抽象方法时可以调用接口的默认方法,但是Lambda表达式不可以.

分类: Java基础

标签: Java6

LevelIsBubble
关注 - 3
粉丝 - 0

« 上一篇：思考对于各种容器对相等的理解
» 下一篇：二叉查找树

posted @ 2017-11-27 20:08 LevelIsBubble 阅读(8) 评论(0) 编辑 收藏

» 更多知识库文章...