抽象类的应用（Timer定时器）

Timer类的schedule方法有个参数是TimerTask，它就是一个抽象类

```
package com.hqyj;

import java.util.TimerTask;

//继承抽象类
public class MyTimerTask extends TimerTask {

    //实现抽象方法，打印当前当前时间
    @Override
    public void run() {
        System.out.println(System.currentTimeMillis());

    }



}
```

```
package com.hqyj;

import java.util.Timer;

public class TimerDemo {

    public static void main(String[] args) {
        Timer timer = new Timer();
        //使用我们自己继承的子类创建对象，作为方法的参数
        timer.schedule(new MyTimerTask(), 5000);

    }

}
```

接口

接口是一种更特殊的抽象，当抽象类全是抽象方法的时候，就用接口（interface）代替。

常见的接口：　电器接口（插座，　USB...）；人机交互接口(电梯按钮，自动售货机...)；软件或程序接口（药店收银系统调用社保中心的接口刷医保卡，前后端分离的软件前端调用后台程序接口获得数据，类的public的方法也是一种接口..）

定义接口

用interface关键字

interface 没有构造方法，也不能实例化

接口里面的属性都是public static final(可以省略)，这样修饰的属性相当于是常量，常量变量名一般用大写

接口方法默认public（省略），不允许static和final

```
package com.hqyj;

//用interface关键字定义文件叫接口（等效于只有抽象方法的抽象类）
//interface 没有构造方法，也不能实例化
public interface Lift {
    //接口里面的属性都是public static final(可以省略)，这样修饰的属性相当于是常量，常量变量名一般用大写
    public static final String BRAND =  "Otis";

    //修饰符默认就是public（省略），不允许static和final
    public void down();
    void up();

}
```

接口实现

关键字implements，习惯于把接口的实现类的类名加上Impl后缀

必须实现所有的接口方法（抽象方法）

```
package com.hqyj;

//implements 实现接口的关键字
public class LiftImpl implements Lift{

    @Override
    public void down() {
        System.out.println("电梯向下运行");

    }

    @Override
    public void up() {
        System.out.println("电梯向上运行");

    }

}
```

类可以实现多个接口，也可以继承类的同时实现接口

```java
public interface Shape {

    //计算面积
    int calcArea();
    //计算周长
    int calcPerimeter();
}

public interface Color {
    //填充颜色
    void fill(String color);
}
```

```java
//一个类同时实现多个接口，接口之间用逗号分隔
//实现类中除了接口方法要实现外，其它的属性方法和构造器的写法跟class写法一致
public class Rectangle implements Shape,Color{
    private int length;
    private int width;
    private String color;

    public Rectangle() {
        super();
    }

    public Rectangle(int length, int width) {
        super();
        this.length = length;
        this.width = width;
    }

    @Override
    public void fill(String color) {
        this.color = color;
    }

    @Override
    public int calcArea() {
        return length*width;
    }

    @Override
    public int calcPerimeter() {
        return (length+width)*2;
    }

    public int getLength() {
        return length;
    }

    public void setLength(int length) {
        this.length = length;
    }

    public int getWidth() {
        return width;
    }
```

```
    public void setWidth(int width) {
        this.width = width;
    }

    public String getColor() {
        return color;
    }


}
```

```
//同时可以继承类和实现接口，implements要放到extends后面
public class Triangle extends TriShape implements Shape, Color{
    private String color;
    private int height;

    public Triangle(int botom, int border2, int border3, int height) {
        super(botom, border2, border3);
        this.height = height;
    }

    @Override
    public void fill(String color) {
        this.color = color;
    }

    @Override
    public int calcArea() {
        return (int)(0.5*height*botom);
    }

    @Override
    public int calcPerimeter() {
        return botom+border2+border3;
    }

}
```

内部类

一个源文件里面定义两个类，这个不叫内部类

一个文件中定义多个类，只有一个是public

```
package com.hqyj;

public class Outer {

}

class Inner{

}
```

内部类是一个类里面包含另一个类。 编译后会生成Outer.class和Outer$Inner.class

```
package com.hqyj;

public class Outer {



    class Inner{

    }
}
```

内部类可以访问外部类的成员， 但是外部类不能访问内部类的成员

如果出现内部类的局部变量和内部类的成员变量，以及外部类的成员变量同名时，访问方式

- add(a, b);//当前方法内的局部变量a
- add(this.a, b);//内部类的成员变量a
- add(Outer.this.a, b); //外部类的成员变量a

```
package com.hqyj;

public class Outer {
    private int a;
    private int b;

    public Outer() {
        super();
    }

    public Outer(int a, int b) {
        super();
        this.a = a;
        this.b = b;
    }

    private int add(int m, int n) {
        return m+n;
    }

    public void test() {
        //callAdd(); //外部类不能直接访问内部类的方法
        //System.out.println(s1); //外部类也不能访问内部类的属性
        Inner inner = new Inner();
        System.out.println("调用Inner类的callAdd(): " + inner.callAdd());
    }

    public static void main(String[] args) {
        Outer outer = new Outer(3, 2);
        outer.test();
    }

    //定义内部类
    public class Inner{
        public String s1 = "abc";
        public int callAdd() {
```

```
            return add(a, b);//可以访问外部类的方法和属性
        }
    }
}
```

在其它类里实例化内部类

```
        //在其它类里实例化内部类
        Inner inner = new Outer(3, 2).new Inner();
        System.out.println(inner.callAdd());
```

静态内部类

静态内部类才能有静态方法

外部类可以通过类名调用静态内部类的静态方法

静态内部类也可以实例化，也可以有非静态的方法

```
public class Outer {
    private static String s = "外部内的静态属性";

    public static void mehtodStatic() {
        System.out.println("外部类的静态方法");
    }

    public static void main(String[] args) {

        //外部类可以通过类名调用静态内部类的静态方法
        InnerStatic.callOuterMethod();

        //静态内部类也可以实例化，也可以有非静态的方法
        InnerStatic innerStatic = new InnerStatic();
        innerStatic.nonStaticMethod();
    }


    //只有静态的内部类，才能定义静态的方法
    public static class InnerStatic{
        public static void callOuterMethod() {
            System.out.println(s);
            mehtodStatic();
        }

        public void nonStaticMethod() {
            System.out.println("静态内部类的非静态方法");
        }
    }
}
```

匿名内部类

接口

```
package com.hqyj;

public interface ButtonListener {
    void run();
}
```

方法的参数是上面接口类型，调用方法时实现接口，这个实现是没有类名，所以称为匿名内部类

```
package com.hqyj;

public class InnerAnonymous {

    public void clickButton(ButtonListener bl) {
        bl.run();
    }

    public static void main(String[] args) {
        InnerAnonymous innerAnonymous = new InnerAnonymous();
        //匿名内部类作为clickButton的参数。实现接口并创建实例
        innerAnonymous.clickButton(new ButtonListener() {

            @Override
            public void run() {
                System.out.println("按钮被点击");

            }

        });
    }


}
```

今日作业

1. 定义订单接口Orders，添加购物车，生成订单，结算....。实现接口OrdersImpl
2. 写一个外部类Cow, 属性年龄，体重， 内部类CowLeg，方法walk，测试代码外部内写个方法去创建内部类的实例，调用walk方法
3. 改写Timer定时器为匿名内部类方式