# 1.静态代码块练习

1.在Frock类中声明私有的静态属性currentNum[int类型]，初始值为100000，作为衣服出厂的序列号起始值。

2.声明公有的静态方法getNextNum;作为生成上衣唯一序列号的方法。每调用一次，将currentNum增加100，并作为返回值。

3.在TestFrock类的main方法中，分两次调用getNextNum方法，获取序列号并打印输出。

4.在Frock类中声明serialNumber(序列号)属性，并提供对应的get方法;

5.在Frock类的构造器中，通过调用getNextNum方法为Frock对象获取唯一序列号，赋给serialNumber属性。

6.在TestFrock类的main方法中，分别创建三个Frock对象，并打印三个对象的序列号，验证是否为按100递增。

```
public class TestFrock {
    public static void main(String[] args) {

        int nextNum1 = Frock.getNextNum();
        System.out.println(nextNum1);//100100
        int nextNum2 = Frock.getNextNum();
        System.out.println(nextNum2);//100200

        System.out.println("========");
        Frock frock = new Frock();
        int serialNumber = frock.getSerialNumber();
        System.out.println(serialNumber);//100300


    }

}
class Frock{
    //静态属性是所有对象共享的
    private static int currentNum = 100000;
    private int serialNumber;

    public Frock() {
        this.serialNumber = getNextNum();
    }

    public int getSerialNumber() {
        return serialNumber;
```

```
    }
    public static int getNextNum(){
        currentNum += 100;
        // 为什么不能直接写：return currentNum + 100;
        return currentNum;
    }
}
```

## 2. 抽象方法 练习题

按要求实现下列问题:

1.动物类Animal包含了抽象方法shout();

2.Cat 类继承 了Animal，并实现方法shout,打印“猫会瞄哺叫”

3.Dog类继承了Animal,并实现方法shout，打印“狗会汪汪叫”

4.在测试类中 实例化对象 Animal cat =new Cat(),并调用cat的shout方法

5.在测试类中实例化对象Animal dog=new Dog0.并调用dog的shout方法

```
public class Homework03 {
    public static void main(String[] args) {
        Animal cat = new Cat();
        cat.shout();
        Animal dog = new Dog();
        dog.shout();
    }
}
abstract class Animal {
    //当该方法被声明为抽象方法时，类就应该是抽象类
    abstract void shout();
}

class Cat extends Animal {
    // 需要重写父类的所有抽象方法
    @Override
    void shout() {
        System.out.println("猫会喵喵叫");
    }
}

class Dog extends Animal {
    @Override
```

```
    void shout() {
        System.out.println("狗会汪汪叫");
    }
}
```

## 3.匿名内部类的使用

1.计算器接口具有work方法，功能是运算，有一个手机类Cellphone，定义方法testWork测试计算功能，调用计算接口的work方法。

2.要求调用CellPhone对象的testWork方法，使用上匿名内部类。

```
/**
 * @auther yt
 * @address https://www.cnblogs.com/y-tao/
 */
public class Homework04 {
    public static void main(String[] args) {
        //1.匿名内部类是
        /*
        new Calculator() {
            @Override
            public double work(double n1, double n2) {
                return n1 + n2;
            }
        }
         */
        //1.同时也是一个对象，他的编译类型是Calculator,而运行类型就是匿名内部类

        // 匿名内部类经典的应用场景就是作为参数列表
        CellPhone cellPhone = new CellPhone();
        //参数是实现了接口的匿名内部类，动态绑定到重写的work方法
        cellPhone.testWork(new Calculator() {
            @Override
            public double work(double n1, double n2) {
                return n1 + n2;
            }
        },2,3);

        //计算两个数的乘积
        //动态的改变计算方式
        cellPhone.testWork(new Calculator() {
            @Override
            public double work(double n1, double n2) {
                return n1 * n2;
```

```
            }        }, 2,4);


    }
}

//编写接口
interface Calculator{
    // work方法是完成计算，自己设计
    // 至于该方法完成怎样的计算，交给匿名内部类去完成
    public double work(double n1, double n2);
}

class CellPhone{
    // 定义方法testWork测试计算功能，调用计算接口的work方法。
    //因为接口中work方法有两个参数，所有还要传入两个参数
    // 当我们调用testWork方法时，直接传入一个实现了Calculator接口的匿名内部类即可
    public void testWork(Calculator calculator, double n1, double n2){
        double result = calculator.work(n1, n2);//动态绑定机制
        System.out.println("计算的结果是=" + result);
    }
}
```

## 4.局部内部类

内部类
1.编一个类A，在类中定义局部内部类B，B中有一个私有常量name，有一个方法show()打印常量name。进行测试。
2.进阶: A中也定义一个私有的变量name，在show方法中打印测试

```
package com.yt.homework;

/**
 * @auther yt
 * @address https://www.cnblogs.com/y-tao/
 */
public class Homework05 {
    public static void main(String[] args) {
        A a = new A();
        a.test01();

    }
}
```

```
//局部内部类是定义在外部类的局部位置，比如通常在方法中，并且有类名。
class A {
    private String name = "jack";

    public void test01(){
        class B {
            private String name;

            public void setName(String name) {
                this.name = name;
            }

            //外部类名.this表示外部类的一个对象(谁调用就是谁的对象)
            public void show(){
                System.out.println("内部类B " + name + "   外部类A  " + A.this.
            }
        }

        B b = new B();//创建内部类对象要在局部内部类后面
        b.setName("tom");
        b.show();

    }

}
```

## 5.接口(知识点多)

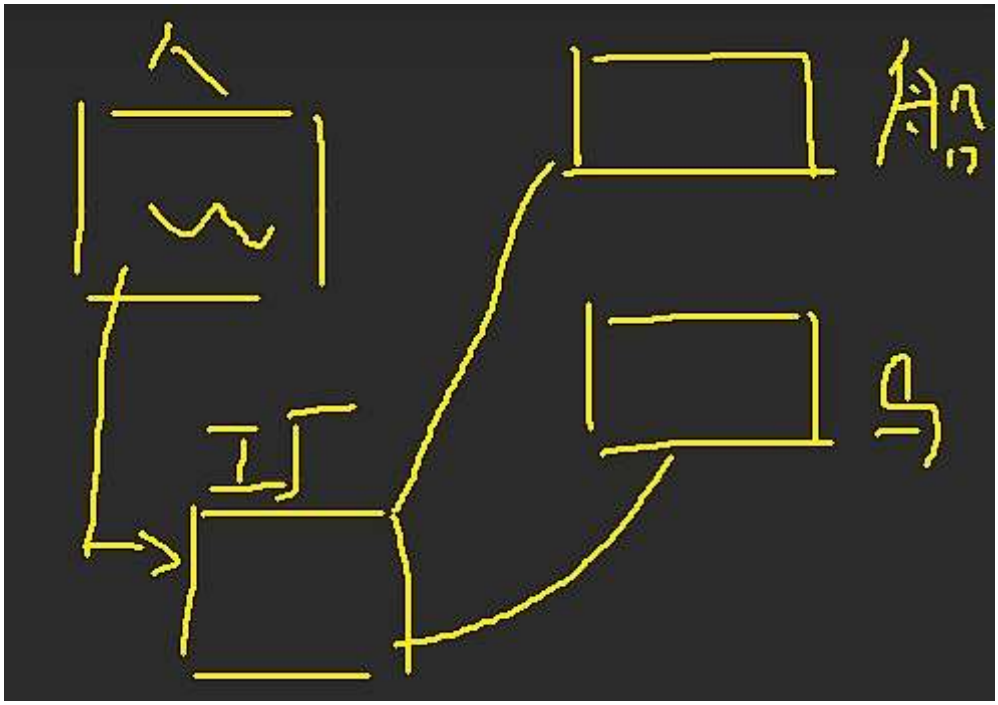
1.有一个交通工具接口类Vehicles,有work接口；

2.有Horse类和Boat类分别实现Vehicles；

3.创建交通工具工厂类,有两个方法分别获得交通工具Horse和Boat；

4.有Person类，有name和Vehicles属性，在构造器中为两个属性赋值；

5.实例化Person对象“唐僧”，要求一般情况下用Horse作为交通工具，遇到大河时用Boat作为交通工具。



```
package com.yt.homework;

/**
 * @auther yt
 * @address https://www.cnblogs.com/y-tao/
 */
/*
1.有一个交通工具接口类Vehicles,有work接口；
2.有Horse类和Boat类分别实现Vehicles；
3.创建交通工具工厂类,有两个方法分别获得交通工具Horse和Boat；
4.有Person类，有name和Vehicles属性，在构造器中为两个属性赋值；
5.实例化Person对象“唐僧”，要求一般情况下用Horse作为交通工具，遇到大河时用Boat作为交通工
6.自己扩展程序：如果唐僧过火焰山，使用飞机
*/
//编程：需求---> 理解 ----> 代码 -----> 优化
public class Homework06 {
    public static void main(String[] args) {
        Person person = new Person("唐僧", new Horse());
        person.common();//一般情况下
        person.passRiver();//过河
        person.common();//一般情况下
        person.passRiver();//过河
        person.passRiver();//过河
        person.volcano();//火焰山

    }
}

//交通工具类,是一个接口类
interface Vehicles {
```

```
    //接口类的方法默认为抽象类
    public void work();
}

//Horse类，实现接口Vehicles
class Horse implements Vehicles {

    @Override
    public void work() {
        System.out.println("一般情况下，使用小马儿");
    }
}

//Boat类，实现接口Vehicles
class Boat implements Vehicles {
    @Override
    public void work() {
        System.out.println("遇到河流，使用小船船");
    }
}
//扩展程序
class Airplane implements Vehicles {
    @Override
    public void work() {
        System.out.println("使用飞机过火焰山");
    }
}


// 创建交通工具工厂类
class Factory {
    //马儿始终是用一个马儿
    private static Horse horse = new Horse();//单例模式中的饿汉式

    //把构造器私有化
    private Factory(){};

    //这里将方法做成static,直接使用类调用方法即可
    public static Horse getHorse(){
//        System.out.println("交通工具是 " + horse);
//        return new Horse();
        return horse;
    }

    public static Boat getBoat(){
//        System.out.println("交通工具是 " + boat);
        return new Boat();
    }
```

```
        //扩展程序，工厂创建飞机
    public static Airplane getAirplane() {
        return new Airplane();
    }
}

//Person类
class Person {
    private String name;
    private Vehicles vehicles;

    //在创建人对象时，事先给他分配一个交通工具
    public Person(String name, Vehicles vehicles) {
        this.name = name;
        this.vehicles = vehicles;
    }
    //实例化Person对象“唐僧”，要求一般情况下用Horse作为交通工具，遇到大河时用Boat作为交
    //编程思想：把具体的要求封装成方法
    //思考：在创建对象时，如何不浪费传入的交通工具对象
    public void passRiver() {
        //先得到船
//        Boat boat = Factory.getBoat();
//        boat.work();
        //这个细节很重要
        //判断一下，如果当前vehicles属性是null，就获取小船船
        // 如何防止始终使用的是马 使用instanceOf
//        if (vehicles == null){
        // vehicles instanceof Boat 判断当前的vehicles是不是Boat
        //1.vehicles==null：vehicles instanceof Boat结果为False
        //2.vehicles是一个马儿对象，vehicles instanceof Boat 结果也是False
        //3.vehicles是一个小船儿对象，vehicles instanceof Boat 结果为True
        if (!(vehicles instanceof Boat)){
            vehicles = Factory.getBoat();
        }
        vehicles.work();
    }
    //一般情况下
    public void common(){
        //通过工厂得到马儿
//        Horse horse = Factory.getHorse();
//        horse.work();
        //判断一下，如果当前vehicles属性是null，就获取一匹马儿
//        if (vehicles == null){
        if (!(vehicles instanceof Horse)){
            //使用的是多态
            vehicles = Factory.getHorse();//向上转型
```

```
            }                   //体现使用接口来调用
            vehicles.work();//接口的一种解耦的特性

        }

        //扩展程序
        //过火焰山，使用飞机
        public void volcano(){
            //判断是否有其他交通工具
            if (!(vehicles instanceof Airplane)){
                vehicles = Factory.getAirplane();
            }

            vehicles.work();
        }
    }
```

## 6.成员内部类

有一个Ca类，有属性temperature(温度)，车内有Air(空调)类，有吹风的功能flow,Air会监视车内的温度，如果温度超过40度则吹冷气。如果温度低于0度则吹暖气，如果在这之间则关掉空调。实例化具有不同温度的Car对象，调用空调的flow方法，测试空调吹的风是否正确.

```
package com.yt.homework;

/**
 * @auther yt
 * @address https://www.cnblogs.com/y-tao/
 */
public class Homework07 {
    public static void main(String[] args) {
//        new Car().new Air().flow(44);
        //实例化不同的Car对象
        Car car = new Car(20);
        car.getAir().flow();
        Car car1 = new Car(-20);
        car1.getAir().flow();

    }
}
class Car{
    private double temperature;

    public Car(double temperature) {
```

```
        this.temperature = temperature;
    }

    //成员内部类
    class Air {
        public void flow(){
            if(temperature > 40){
                System.out.println("吹冷气！！");
            } else if(temperature < 0) {
                System.out.println("吹暖气！！！");
            } else {
                System.out.println("关闭空调！！");
            }
        }
    }

    public Air getAir(){
        return new Air();
    }
}
```

## 7.枚举类

1. 创建一个Color枚举类
2. 有RED,BLUE,BLACK, YELLOW,GREEN这个五个枚举值/对象
3. Color有三个属性redValue,greenValue,blueValue
4. 创建构造方法，参数包括这三个属性
5. 每个枚举值都要给这三个属性赋值，三个属性对应的值分别是
6. red: 255,0,0 ； blue:0,0,255； black:0,0,0； yellow:255,255,0； green:0,255,0；
7. 定义接口，里面有方法show，要求Color实现该接口
8. show方法中显示三属性的值
9. 将枚举对象在switch语句中匹配使用

```
package com.yt.homework;

/**
 * @auther yt
 * @address https://www.cnblogs.com/y-tao/
```

```
 */  public class Homework08 {
    public static void main(String[] args) {
        //演示枚举值的switch的使用
        Color green = Color.GREEN;
        green.show();

        //switch()中放入枚举对象
        //在每个case后面。直接写上在枚举类中定义的枚举对象即可
        switch(green){
            case READ:
                System.out.println("匹配到红色");
                break;
            case GREEN:
                System.out.println("匹配到绿色");
                break;
            case BLUE:
                System.out.println("匹配到蓝色");
                break;
            case BLACK:
                System.out.println("匹配到黑色");
                break;
            case YELLOW:
                System.out.println("匹配到黄色");
                break;
            default:
                System.out.println("没有匹配到。。。");
        }

    }
}

//枚举类
enum Color implements IColor{
    //枚举
    READ(255,0,0),BLUE(0,0,255),
    BLACK(0,0,0),YELLOW(255,255,0),
    GREEN(0,255,0);
    //属性
    private int redValue;
    private int greenValue;
    private int blueValue;
    //构造器
    Color(int redValue, int greenValue, int blueValue) {
        this.redValue = redValue;
        this.greenValue = greenValue;
        this.blueValue = blueValue;
    }
```

```
        //实现接口中的方法
    @Override
    public void show() {
        System.out.println("属性值为" + redValue + "," + greenValue + "," + bl
    }
}
//定义接口
interface IColor{
    public void show();
}
```