十一、本章综合练习:

Home01:

```
package com.HomeWork;

public class Home01 {
    public static void main(String[] args) {
        System.out.println(Frock.getNextNum());//100100
        System.out.println(Frock.getNextNum());//100200
        Frock frock = new Frock();//100300
        Frock frock1 = new Frock();//100400
        Frock frock2 = new Frock();//100500
        //输出:
        System.out.println(frock.getSerialNumber());
        System.out.println(frock1.getSerialNumber());
        System.out.println(frock2.getSerialNumber());
    }
}
class Frock{
    //衣服出场的序列号起始值(由于static:所有new对象时值加载一次,下一次new便不再加载了)
    private static int currentNum = 100000;

    //提供对应的get()
    private int serialNumber;

    public Frock() {
        //为serialNumber获取唯一的序列号
        this.serialNumber = getNextNum();
    }

    public int getSerialNumber() {
        return serialNumber;
    }



    //生成上衣唯一序列号的方法,没调用一次currentNum增加100,并返回值
    public static int getNextNum(){
        return currentNum += 100;
    }
}
```

Home02:

```
package com.HomeWork;

public class Home02 {
    public static void main(String[] args) {
        Cat cat = new Cat();
        cat.shout();
        Dog dog = new Dog();
```

```
        dog.shout();
    }
}
//抽象类
abstract class  Animal{
    abstract public void shout();//抽象类必须公共,且没有static
}
//继承抽象类的类必须重写抽象类的方法 或者 自己本身也是抽象类,否则报错
class Cat extends Animal{
    @Override
    public void shout() {
        System.out.println("猫猫叫唤...");
    }
}
class Dog extends Animal{
    @Override
    public void shout() {
        System.out.println("狗会叫...");
    }
}
```

Home03:

```
package com.HomeWork;

import org.omg.PortableInterceptor.SYSTEM_EXCEPTION;

import javax.swing.*;

/*考察匿名内部类的使用:*/
public class Home03 {
    public static void main(String[] args) {

        Cellphone cellphone = new Cellphone();
        cellphone.testWork(new Calculate() {
            @Override
            public double work(double n1, double n2) {
                return n1 + n2;
            }
        },10,3);


    }
}

interface Calculate {
    //计算功能接口方法
    double work(double n1, double n2);//前缀默认是:public static final
}

class Cellphone {
    //测试类要调用接口方法要传入接口对象
    public static void testWork(Calculate calculate, double n1, double n2) {
        double result = calculate.work(n1, n2);
        System.out.println("计算结果:" + result);
```

```
    }
}
```

Home04:

```
package com.HomeWork;
/*内部类练习:*/
public class Home04 {
    public static void main(String[] args) {
        A a = new A();
        a.print("阿珍");
    }
}
class A{
    private String name = "啊强";
    public void print(String name){
        class B{
            private String name;
            public void show(String name){
                System.out.println(name + "\t爱上\t" + A.this.name);
            }
        }
        B b = new B();
        b.show(name);
    }

}
```

Home05:

```
/*
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
6.增加一个情况，如果唐僧过火焰山，使用 飞机 ==> 程序扩展性
使用代码实现上面的要求
编程 需求----->理解----->代码-->优化
 */
```

.

Home05:

```
package com.HomeWork.Home05;
/*实例化Person对象"唐僧"*/
public class Home05 {
    public static void main(String[] args) {
        Person tang = new Person("唐僧", new Boat());
        tang.common();//一般情况下用马车
        tang.passRiver();//过河用船只
        tang.passFireMountain();//过火焰山用飞机
    }
}
```

interface Vehicles:

```
package com.HomeWork.Home05;
/*交通工具接口类Vehicles,有work()实现规定*/
public interface Vehicles {
    void work();
}
```

Horse:

```
package com.HomeWork.Home05;
/*有Hours类实现Vehicles*/
public class Horse implements Vehicles{
    @Override
    public void work() {
        System.out.println("一般情况下,使用马车...");
    }
}
```

Boat:

```
package com.HomeWork.Home05;
/*有Boat类实现Vehicles*/
public class Boat implements Vehicles{
    @Override
    public void work() {
        System.out.println("过河时,使用船只...");
    }
}
```

Plane:

```
package com.HomeWork.Home05;

public class Plane implements Vehicles{
    @Override
    public void work() {
        System.out.println("过火焰山时,使用飞机...");
    }
}
```

VehiclesFactory:

```
package com.HomeWork.Home05;
/*创建交通工具工厂,可以获得Horse,Boat*/
public class VehiclesFactory {
    //马是同一匹马(如果这里先用static声明对象,下一次实例化时就不用再次加载一个新的马匹了)
    private static Horse horse = new Horse();//饿汉式
    private static Plane plane = new Plane();
```

```
    public static Horse getHorse(){
       //return new Horse();
        return horse;
    }
    public static Boat getBoat(){
        return new Boat();
    }
    public static  Plane getPlane(){
        return plane;
    }
}
```

Person:

```
package com.HomeWork.Home05;

public class Person {
    private String name;
    //声明一下交通工具的接口
    private Vehicles vehicles;

    public Person(String name, Vehicles vehicles) {
        this.name = name;
        this.vehicles = vehicles;
    }
    /*要求一般情况下用Horse作为交通工具,大河时使用Boat作为交通工具*/
    //编程思想:可以把具体要求封装成具体方法
    public void common(){
        //到工厂里拿一辆马车(由于方法已经static所以给可以直接用)
        //若本来准备的(传的接口对象)不是马车,则进入循环,到工厂了拿
        if (!(vehicles instanceof Horse)){
            /*若没有交通工具 或 交通工具不是马车
                     ---->
            vehicles == null or  vehicles != Horse
            则进入循环将交通工具(vehicles)赋值成马车(Horse)*/
            vehicles = VehiclesFactory.getHorse();
        }
        vehicles.work();
    }
    public void passRiver(){
        //到工厂里拿一艘船(由于方法已经static所以给可以直接用)
        if (!(vehicles instanceof Boat)){
            //同理:
            vehicles = VehiclesFactory.getBoat();
        }
        vehicles.work();
    }

    public void passFireMountain(){
        if (!(vehicles instanceof Plane)){
            vehicles = VehiclesFactory.getPlane();
        }
        vehicles.work();
    }
}
```

Home06:

Home06:

```
package com.HomeWork.Home06;

public class Home06 {
    public static void main(String[] args) {
        Car car = new Car(60);
        car.getAir().flow();
    }
}
```

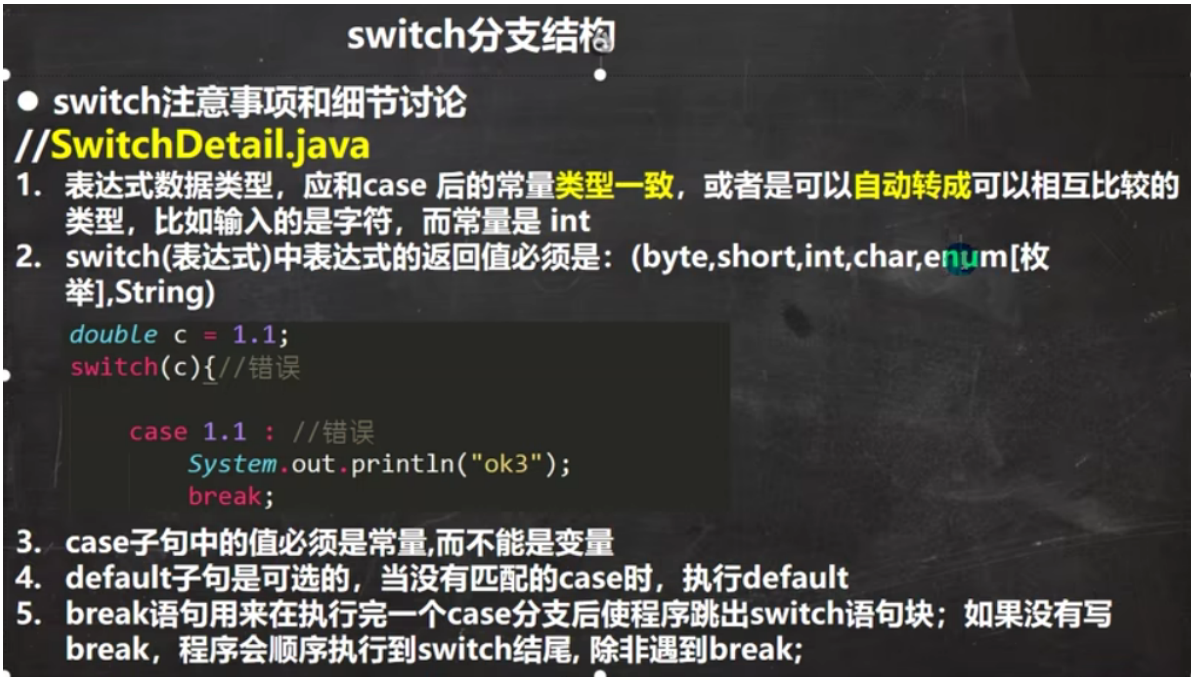
Car :

```
package com.HomeWork.Home06;
/*1.属性:temperature
* 2.这内有空调Air类:
*   有吹风功能flow
*   Air有监视车内温度的功能(若超过40度则吹冷气,若低于0度则吹暖气)
*   若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flow方法,测试空调吹的风是否正确*/
public class Car {
    private double temperature;//车室内温度

    public Car(double temperature) {
        this.temperature = temperature;
    }

    class Air{
        public void flow(){//吹风功能
            if (temperature > 40){
                System.out.println("空调吹冷气");
            }else if(temperature < 0){
                System.out.println("空调吹暖气");
            }else{
                System.out.println("空调关闭");
            }
        }
    }
    public Air getAir(){
        return new Air();
    }
}
```

Home07:



。

Home07：

```
package com.HomeWork.Home07;

public class Home07 {
    public static void main(String[] args) {
        //枚举值的switch使用
        Color black = Color.BLACK;
        black.show();//显示black有哪些参数
        /*switch()参数必须是:
        (byte,short,int,char,enum,String)*/
        switch (black){
            case BLACK:
                System.out.println("匹配到黑色");
                break;
            case RED:
                System.out.println("匹配到红色");
                break;
            case BLUE:
                System.out.println("匹配到蓝色");
                break;
            case GREEN:
                System.out.println("匹配到绿色");
                break;
            case YELLOW:
                System.out.println("匹配到黄色");
                break;
            default:
                System.out.println("匹配不到任何颜色");
                break;
        }
    }
}
```

interface print:

```
package com.HomeWork.Home07;
/*4.定义接口，里面有show()，要求Color实现接口
 * 5.show()显示三个属性值
 * 9.将枚举对象在switch语句中匹配使用*/
public interface print {
    void show();
}
```

enum Color:

```
package com.HomeWork.Home07;

import jdk.internal.org.objectweb.asm.tree.analysis.Value;

/*1.有三个属性：redValue，greenValue，blueValue
 * 2.创建构造方法，参数包括这三个属性
 * 3.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：
 * red：255,0,0 blue：0,0,255 black: 0,0,0 yellow: 255,255,0 green:0,255,0
 * 4.定义接口，里面有show()，要求Color实现接口
 * 5.show()显示三个属性值
 * 9.将枚举对象在switch语句中匹配使用*/
public enum Color implements print {
    /*写法等价于public static final Color RED = new Color(255, 0, 0);
    * 此写法是: 类为非enum时使用 */
    RED(255, 0, 0),
    BLUE(0, 0, 255),
    BLACK(0, 0, 0),
    YELLOW(255, 255, 0),
     GREEN(0, 255, 0);
    private int redValue;
    private int greenValue;
    private int blueValue;

    private Color(int redValue, int greenValue, int blueValue) {
        this.redValue = redValue;
        this.greenValue = greenValue;
        this.blueValue = blueValue;
    }


    public int getRedValue() {
        return redValue;
    }

    public int getGreenValue() {
        return greenValue;
    }

    public int getBlueValue() {
        return blueValue;
    }

    @Override
```

```
    public void show() {
        System.out.println("属性值: " +
                getRedValue() + "," +
                getGreenValue() + "," +
                getBlueValue());
    }
}
```