面向对象-day04

今日学习内容：

- this关键字
- super关键字
- final修饰符
- 代码块
- 内部类
- 枚举

今日学习目标：

- 重点掌握this关键字的含义和用法
- 重点掌握super关键字的含义和用法
- 掌握final修饰符的修饰类，方法，变量的含义
- 了解代码块有那些
- 掌握静态代码块的语法
- 了解内部类有哪些
- 掌握匿名内部类的语法
- 掌握枚举类的定义和使用

13. 面向对象查漏补缺

1.1. This关键字（重点掌握）

1.1.0 this 回顾

观察代码

```
public class Dog {
    String sn;
    String name;
    int age;

    public Dog(){}

    public Dog(String s,String n,int a){
        sn = s;
        name = n;
        age = a;
    }

    public void sayHi(){
        System.out.println("我的名字:"+ name);
        System.out.println("我的年龄:" + age);
    }
}
```

以上代码存在什么问题?

思考1：前面什么地方用到this？setter / constractor

思考2：this到底是什么呢？

回顾之前，this主要存在于两个位置：

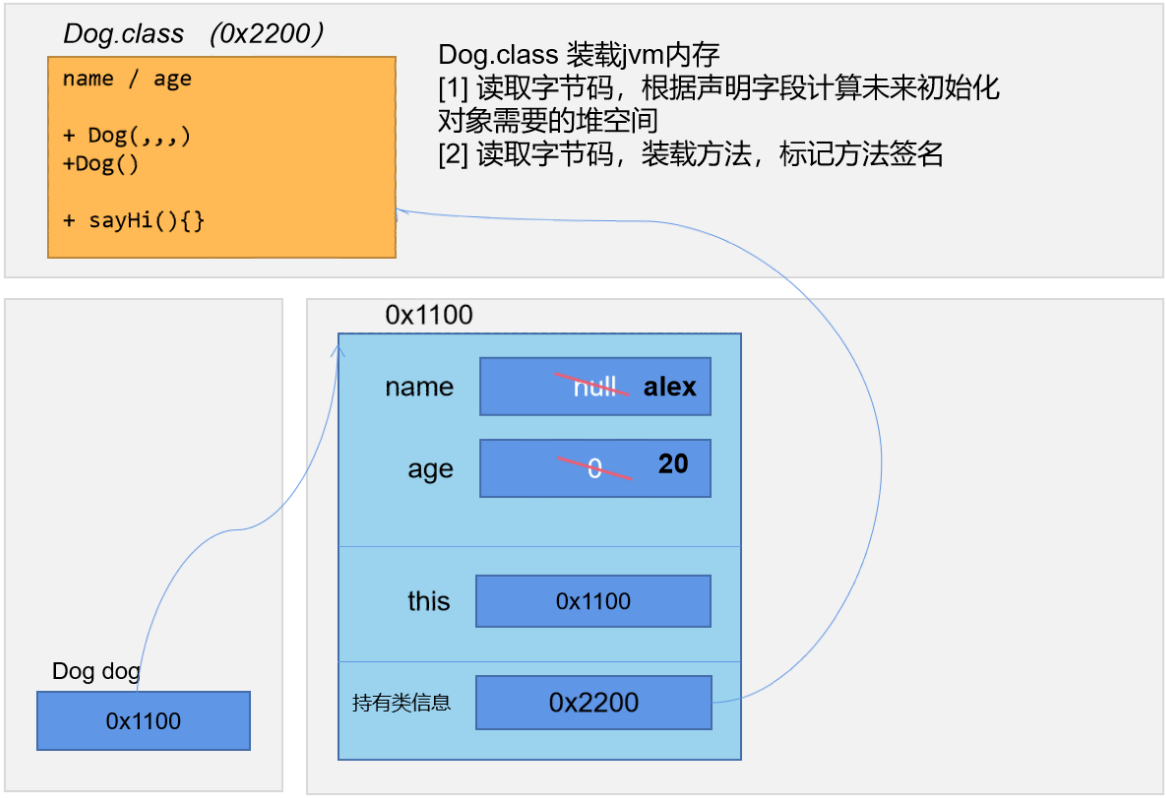
- 在构造器中：表示当前被创建的对象
- 在方法中：哪一个对象调用this所在的方法，此时this就表示哪一个对象

优化后的Dog

```
/**
 * 满足封装概念的Dog
 */
public class Dog {
    private String sn;
    private String name;
    private int age;

    public void setSn(String sn) {
        this.sn = sn;
    }

    public String getSn() {
        return sn;
    }

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

    public String getName() {
        return name;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public int getAge() {
        return age;
    }

    public Dog(String sn, String name, int age) {
        this.sn = sn;
        this.name = name;
        this.age = age;
    }

    public Dog() {

    }
}
```

问：this有什么用，什么时候用this？this怎么来的？

1.1.1 this 内存图



this关键字表示当前对象本身，一般用于类的内部，其内部存在一个地址，指向当前初始化的对象本身。

```
public class Test01 {
    public static void main(String[] args) {
        Dog dog = new Dog("旺财", 20);
        System.out.println("dog = " + dog);
    }
}
```

当new一个对象时，实际上产生了两个引用，一个是供类Dog内部调用其成员变量和成员方法的this关键字，一个是供外界程序调用实例成员的dog。

1.1.2 this 三种用法

什么时候需要使用this：

[1] 调用成员变量（掌握）

解决局部变量和成员变量之间的二义性，此时必须使用

[2] 调用其他成员方法（掌握）

同一个类中非static方法间互调（此时可以省略this，但是不建议省略）

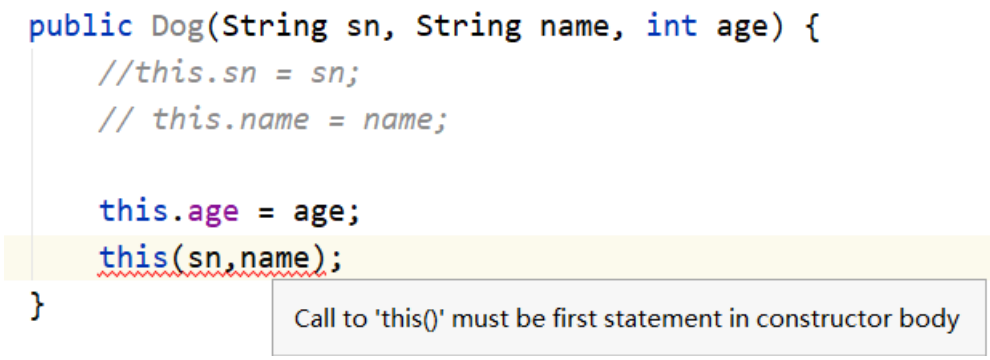
```
public void sayHi(){
    // System.out.println("大家好，我叫"+this.name+",我今年"+this.age+"岁");

    System.out.println("我的自白:");
    this.showInfo();
}
```

[3] 调用本类其他构造方法（掌握）

this可以调用本类其他构造方法，语法

```
this();
this(,,,);
```

注意：在构造方法中调用本类其他构造方法必须写到该构造方法第一句，否则出现编译错误。



一个结合this，满足封装的实战中的Dog类。

```
public class Dog {
    private String sn;
    private String name;
    private int age;

    public void setSn(String sn) {
        this.sn = sn;
    }

    public String getSn() {
        return sn;
    }

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

    public String getName() {
        return name;
    }

    public void setAge(int age) {
        this.age = age;
    }
```

```
    public int getAge() {
        return age;
    }

    public Dog(String sn, String name, int age) {
        //this.sn = sn;
        // this.name = name;

        this(sn,name);
        this.age = age;
    }

    public Dog(String sn,String name){
        this.sn = sn;
        this.name = name;
    }

    public Dog() {

    }

    public void sayHi(){
        // System.out.println("大家好，我叫"+this.name+",我今年"+this.age+"岁");

        this.showInfo();
    }

    public void showInfo(){
        System.out.println("我的自白:");
        System.out.println("我的名字:" + this.name);
        System.out.println("我的年龄:" + this.age);
    }
}
```

需求1：请用面向对象知识构建以下类信息。

类型	字段			行为
狗狗	昵称	健康值	品种	输出信息
猫猫	昵称	健康值	性别	输出信息

```
public class Pet {
    private String name;
    private int health;

    public Pet(String name, int health) {
        this.name = name;
        this.health = health;
    }

    public Pet(){}

    public void print(){
```

```
        System.out.println("我的名称:" + this.name);
        System.out.println("我的健康值:" + this.health);
    }
}
```

Dog.java

```
public class Dog extends Pet{

    private String strain; // 品种

    public void setStrain(String strain) {
        this.strain = strain;
    }

    public String getStrain() {
        return strain;
    }

    public Dog(){}
    public Dog(String name,int health,String strain){
        setName()
        setHealth()
        this.strain = strain;
    }

    // 问题1:Dog中有几个成员变量?
    // 写Dog类的构造方法，会遇到什么问题?
}
```

1.2. super关键字（重点掌握）

回顾之前什么时候使用super:

- 在子类方法中，调用父类被覆盖的方法，此时必须使用super (回顾课堂案例)

1.2.1 super 关键字

super 关键字表示父类对象，子类要访问父类成员时可以使用super。
super只是一个关键字，内部没有引用（地址）。

[1] super 访问父类非私有字段（了解）

```
System.out.println("我的名字" + super.nick);
System.out.println("我的健康值" + super.health);
System.out.println("我是一只" + this.strain);
```

[2] super 访问父类非私有方法(重要)

```
super.print();
System.out.println("我是一只" + this.strain);
```

[3] super 访问父类构造方法(重要)

```
语法:
super();
super(,,,);
```

```
public class Dog extends Pet {
    String strain;

    public Dog(String nick,int health,String strain){
        super(nick,health);
        this.strain = strain;
    }
    ` ` ` `
}
```

总结:
[1]super 调用构造方法必须写在子类构造方法的第一句。
[2]如果子类构造方法没有显式调用父类构造方法时，那么jvm会默认调用父类的无参构造super()

```
public class Dog extends Pet {
String strain;

public Dog(){
    // super();
}

public Dog(String nick, int health, String strain) {
    // super(nick, health, love);
    super();
    this.strain = strain;
}
```

实战开发中，结合this，super，封装、继承下的类

```
public class Pet {
    private String nick;
    private int health;
    private int love;

    public String getNick() {
        return nick;
    }

    public void setNick(String nick) {
        this.nick = nick;
    }

    public int getHealth() {
        return health;
    }
```

```
    public void setHealth(int health) {
        if (health < 0) {
            System.out.println("健康值不合法");
            this.health = 100;
        } else {
            this.health = health;
        }
    }

    public int getLove() {
        return love;
    }

    public void setLove(int love) {
        this.love = love;
    }

    public Pet() {
        super();
    }

    public Pet(String nick, int health, int love) {
        super();
        this.nick = nick;
        this.setHealth(health);
        this.love = love;
    }

    public void print() {
        System.out.println("我的名字:" + this.nick);
        System.out.println("我的健康值:" + this.health);
        System.out.println("我的亲密度:" + this.love);
    }
}
```

```
public class Dog extends Pet {
    private String strain;

    public String getStrain() {
        return strain;
    }

    public void setStrain(String strain) {
        this.strain = strain;
    }

    public Dog() {
        super();
    }

    public Dog(String nick, int health, int love, String strain) {
        super(nick, health, love);
        this.strain = strain;
    }
```

```
    @Override
    public void showInfo() {
        super.showInfo();
        System.out.println("strain:" + this.strain);
    }
}
```

1.3. static修饰符（掌握）

需求 + 问题：

问一个生产车的工厂，一共生产了多少量车？
=> 构成一个车(Car) 的类，统计Car一共创建了多少对象？
=> 紧接着继续思考：统计Car创建了多少对象是不是需要一个变量totalCount？
=> 紧接着继续思考：在哪里声明这个变量totalCount呢？

1.3.1 static

static 关键字表示静态，可以修饰变量构成静态变量，修饰方法构成静态方法。

静态变量和静态方法都归类所有，称为类的静态成员，用static关键字修饰。

1.3.2 静态变量

在类中，用static关键字修饰的成员变量称为静态变量，归类所有，也称为类变量，类的所有实例/对象都可以访问，被类的所有实例或对象所共享。

语法：

```
static 数据类型 成员变量 [=初始值];
```

静态变量的访问

```
类名.静态变量(推荐写法)
对象/实例.静态变量
```

需求:Car创建了多少量车？

```
public class Car{

    String brand;
    String type;
    float price;

    // 静态变量，归类所有
    static int count = 0;

    public Car(){
```

```
        //Car.count++;
        this.count++;
    }

    public Car(String brand,String type,float price){
        this.brand = brand;
        this.type = type;
        this.price = price;

        // Car.count++;
        this.count++;
    }
    ````
}
```

思考:为什么通过实例可以访问静态成员？ => 类变量/静态变量被类的所有实例或对象所共享!

## 1.3.2 静态方法

static 也可以修饰方法称为静态方法，归类所有，也称类方法。形式

```
[修饰符] static 返回值类型 方法名(形参列表){

}
```

静态方法访问方式

```
类名.静态方法() (推荐)
对象.静态方法()
```

静态方法特性

[1] 静态方法中可以访问静态变量和类的其他静态方法

[2] 实例方法中可以访问静态成员(静态变量和静态方法);静态方法不能访问实例成员

```
public class Car {

    static int num = 0;

    public Car(){
        Car.num++;

        //num++;  <==> this.num++;
        //this.num++;
    }

    public static void test(){
        System.out.println("test");
    }

    public static int getNum(){

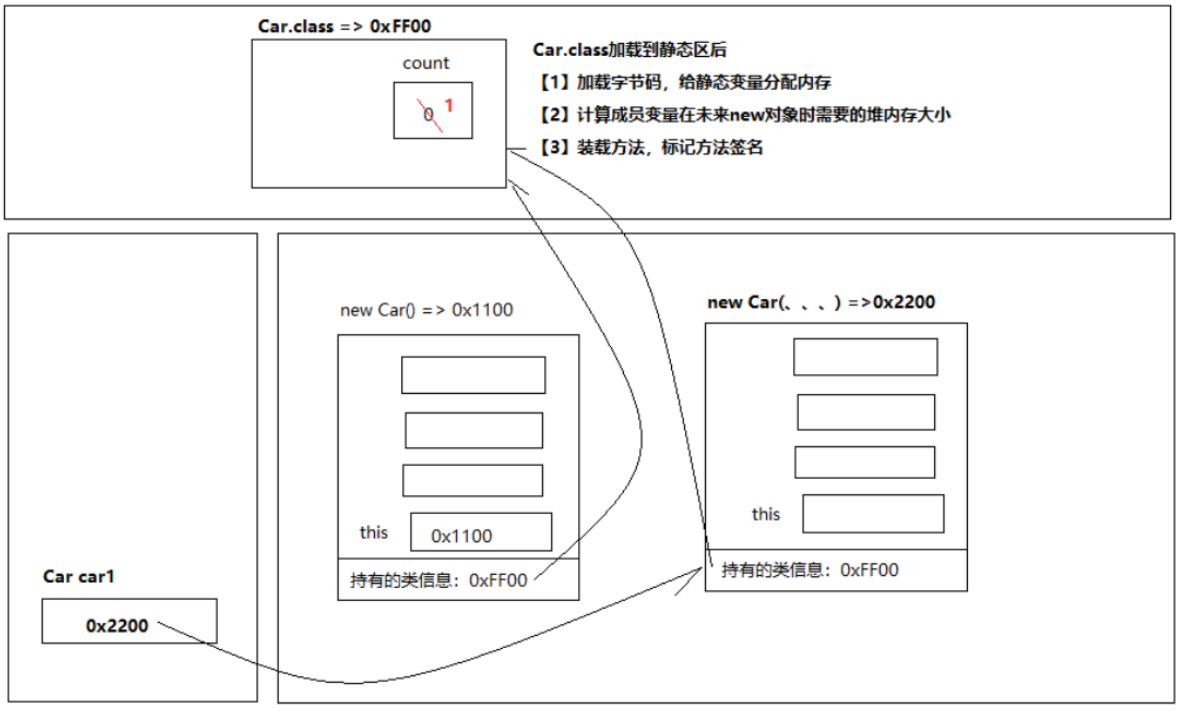
        // System.out.println(this.count);
```
```

```
        // this.showInfo();

        Car.test();
        return Car.num;
    }

    public void showInfo(){
        // 访问静态变量
        // System.out.println(Car.num);

        // 访问静态方法
        Car.getNum();
    }
}
```

1.3.4 jvm加载 static 成员的过程（理解）

当加载一个类到jvm的方法区时，首先jvm会扫描xx.class中的静态成员并分配空间且初始化。

当通过xx.class new一个对象时，可以在该对象的实例方法中访问静态成员；反之静态方法中不能访问实例成员。



1.4. final修饰符（掌握）

final 表示最终的意思，可以修饰类、方法、局部变量、成员变量。

1.4.1 最终类(掌握)

final 修饰类表示最终类。

```
public final class Car extends MotoVehicle {
}
```

最终类不能被继承。

1.4.2 最终方法(掌握)

如果一个方法被final修饰，称为最终方法。

```
public final void test() {
    System.out.println("test");
}
```

最终方法不能被重写。

1.4.3 常量(掌握)

final修饰的局部变量称为**常量**，常量只能赋值一次，不能再重新赋值。

- 基本数据类型：表示的值不能改变
- 引用数据类型：所引用的地址值不能改变

[1] final修饰基本数据类型

```
final int a = 10;
// error
// a = 20
```

[2] final修饰引用数据类型

```
// 常引用
// final 修饰引用数据类型
final Car car = new Car();
System.out.println(car);
car.setBrand("Benz");
car.setType("X5");

car.setBrand("Audi");
car.setBrand("A4");

// car 被final修饰，不能再用于指向其他堆空间
//car = new Car();
//System.out.println(car);
```

1.5. 代码块（了解）

{} 标记的代码称为代码块，根据其位置的不同可以分为普通代码块、构造代码块、静态代码块、同步代码块(后续讲解)。

1.5.1普通代码块（已学过）

普通代码块{}，也成局部代码块，一般存在于方法中，形成作用域。

作用域特性:
[1].作用域可以嵌套，内层作用作用域可以访问外层作用域的变量
[2].当访问一个变量时，首先在变量所在的作用域查找，如果能找到，停止查找并输出变量内容；当本作用域没找到时，尝试去上一层作用域查找，依次类推。这个过程形成的查找链称为作用域链。

```
public class Dog {

    int count0 = 0;

    public void showInfo(){

        int count1 = 10;

        // 普通代码块
        {
            // int count1 = 100;
            int count2 = 20;
            System.out.println(count2);
            System.out.println(count1);
            // System.out.println(count0);

            System.out.println(this.count0);
        }
    }

}
```

1.5.2构造代码块（了解）

构造代码块在类中(类的内部)、方法外
构造代码块构造一个对象执行一次，在构造方法前执行。

```
public class Car{

    private String brand;
    private String type;

    // 构造代码块
    {
        System.out.println("构造代码块...");
    }

    public Car(){

    }

    public Car(String brand,String type,float price){
        System.out.println("Car(String,String,float)");
        this.brand = brand;
        this.type = type;
    }
}
```

开发中不使用初始化代码块，即使要做初始化操作，可以直接在构造器中完成即可。

1.5.3静态代码块(掌握)

被static关键字修饰的代码块称为静态代码块。
静态代码块位于类的内部、方法的外部。
静态代码块只执行一次（jvm加载xx.class时执行），在构造代码块、构造方法前执行。

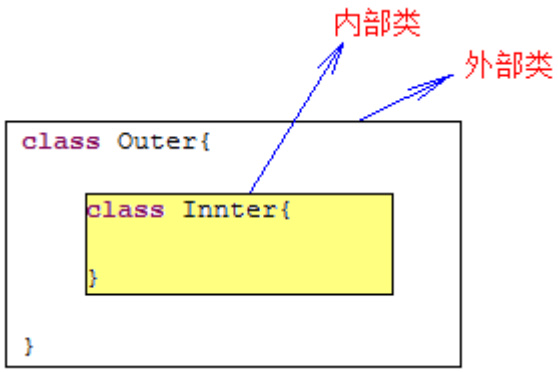
```
public class Car{
    String brand;
    String type;
    float price;

    static int count;

    // 静态代码块
    static{
        System.out.println("静态代码块...");
        count = 0;
    }

    public Car(){

    }

    public Car(String brand,String type,float price){
        System.out.println("Car(String,String,float)");
        this.brand = brand;
        this.type = type;
        this.price = price;
    }
}
```

当类的字节码被加载到内存时，此时程序需要加载一些资源(读取资源文件、读取配置文件等)，可以使用静态代码块，此时被加载进来的资源一般都可以被多个实例所共享。

1.6. 内部类

1.6.2 内部类概述（了解）

什么是内部类，把一个类定义在另一个类的内部，把里面的类称之为内部类，把外面的类称之为外部类。（能认识内部类即可）



内部类可以看作和字段、方法一样，是外部类的成员，而成员可以有static修饰。

- 静态内部类：使用static修饰的内部类，那么访问内部类直接使用外部类名来访问

- 实例(成员)内部类：没有使用static修饰的内部类，访问内部类使用外部类的对象来访问
- 局部(方法)内部类：定义在方法中的内部类，一般不用
- 匿名内部类：特殊的局部内部类，适合于仅使用一次使用的类

对于每个内部类来说，Java编译器会生成独立.class文件。

- 静态和实例内部类：外部类名$内部类名字
- 局部内部类：外部类名$数字内部类名称
- 匿名内部类：外部类名$数字

1.6.6 匿名内部类(掌握)

当一个类只使用一次，可以声明成匿名内部类。匿名内部类 必须有 **实现** 存在。

匿名内部类，可以使用父类构造器和接口名来完成。

针对类，定义匿名内部类来继承父类（使用较少）：

```
new 父类构造器([实参列表]){
    //匿名内部类的类体部分
}
```

针对接口，定义匿名内部类来实现接口（使用较多）：

```
new 接口名称(){
    //匿名内部类的类体部分
}
```

注意：这里不是根据 父类/接口 创建对象，而是一种语法而已

```
public interface IHouseFindable {
    public void findHouse();
}
-----------------------------------------------
    public class Student {
    private String name;
    private IHouseFindable findable;

    public String getName() {
        return name;
    }

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

    public IHouseFindable getFindable() {
        return findable;
    }

    public void setFindable(IHouseFindable findable) {
        this.findable = findable;
    }
```

```
    public void learn(){
        System.out.println("java");
    }

    public void findHouse(){
        findable.findHouse();
    }
}
-----------------------------------------------------------------
 public class Test01 {
    public static void main(String[] args) {
        /*
        class Inner implements IHouseFindable {

            @Override
            public void findHouse() {
                System.out.println("老师帮助找房子");
            }
        }
        IHouseFindable findable = new Inner();
        */

        IHouseFindable findable = new IHouseFindable() {
            @Override
            public void findHouse() {
                System.out.println("找到房子");
            }
        };


        Student s1 = new Student();
        s1.setName("二狗");
        s1.setFindable(findable);
        s1.findHouse();
    }
}
```

1.7. 枚举类（掌握）

1.7.1. 枚举的诞生史（了解）

在服装行业，衣服的分类根据性别可以表示为三种情况：男装、女装、中性服装。

```
private  ?  type;
public void setType(? type){
    this.type = type;
}
```

需求：定义一个变量来表示服装的分类？请问该变量的类型使用什么？

使用int和String类型，且先假设使用int类型，因为分类情况是固定的，为了防止调用者乱创建类型，可以把三种情况使用**常量**来表示。

```
public class ClothType {
    public static final int MEN = 0;
    public static final int WOMEN = 1;
    public static final int NEUTRAL = 2;
}
```

注意：常量使用final修饰，并且使用大写字母组成，如果是多个单词组成，使用下划线分割。

此时调用setType方法传递的值应该是ClothType类中三个常量之一。但是此时依然存在一个问题——依然可以乱传入参数比如100，此时就不合理了。

同理如果使用String类型，还是可以乱设置数据。那么说明使用int或String是类型不安全的。那么如果使用对象来表示三种情况呢?

```
public class ClothType {
    public static final ClothType MEN =  new ClothType();
    public static final ClothType WOMEN =  new ClothType();
    public static final ClothType NEUTRAL =  new ClothType();
}
```

此时调用setType确实只能传入ClothType类型的对象，但是依然不安全，为什么？因为调用者可以自行创建一个ClothType对象，如：setType(new ClothType())。

此时为了防止调用者私自创建出新的对象，我们把CLothType的构造器私有化起来，外界就访问不了了，此时调用setType方法只能传入ClothType类中的三个常量。此时代码变成：

```
public class ClothType {
    public static final ClothType MEN =  new ClothType();
    public static final ClothType WOMEN =  new ClothType();
    public static final ClothType NEUTRAL =  new ClothType();
    private ClothType() {}
}
```

高，实在是高！就是代码复杂了点，如果存在定义这种类型安全的且对象数量固定的类的语法，再简单点就更好了——有枚举类。

1.7.2. 枚举类的定义和使用（掌握）

枚举是一种特殊的类，专门用于声明可罗列的常量值，定义格式：

```
public enum  枚举类名{
    常量对象A,
    常量对象B,
    常量对象C ;
}
```

我们自定义的枚举类在底层都是直接继承了java.lang.Enum类的。

```
public enum ClothType {
    MEN, WOMEN, NEUTRAL;
}
```

枚举中都是全局公共的静态常量，可以直接使用枚举类名调用。

```
ClothType type = ClothType.MEN;
```

因为java.lang.Enum类是所有枚举类的父类,所以所有的枚举对象可以调用Enum类中的方法.

```
String  name = 枚举对象.name();          //  返回枚举对象的常量名称
int ordinal  = 枚举对象.ordinal();  //  返回枚举对象的序号,从0开始
```

int ordinal = 枚举对象.ordinal(); // 返回枚举对象的序号,从0开始

注意：枚举类不能使用创建对象

```
public class EnumDemo {
    public static void main(String[] args) {
        int ordinal = ClothType.MEN.ordinal();
        String name = ClothType.MEN.name();
        System.out.println(ordinal);
        System.out.println(name);

        new ClothType();    //语法报错
    }
}
```

目前，会定义枚举类和基本使用就可以了，后面还会讲更高级的使用方式。