代码块 （重点）

一、基本介绍

- **代码块(初始化块)：简单理解成 只有方法体的方法.**
- **语法：**

```
/*
    [修饰符]{
        代码;
    };
*/
```

- **注意**

1. **修饰符 可选写 也只能写 static.**
2. **有static为静态代码块 ， 没有则为普通代码块.**
3. **逻辑语句可以为任何逻辑语句。**
4. **; 号可写可不写.**

二、代码块的好处

1. **相当于另外一种形态的构造器(对构造器的一种补充机制),可以做初始化的操作.**
2. **场景：如果多个构造器中都有多个重复的语句，可以抽取到初始化块中，提高代码的复用性.**

- **代码例子**

```
package com.codeBlock;
/*1.三个构造器构成了重载
 * 2.三个构造器都有相同的语句,代码复用性低
 * 3.我们可以把相同的语句放入到一个代码块中
 * 4.这样无论调用哪个构造器,在创建对象之后,都会先调用代码块的内容*/
public class Block01 {
    public static void main(String[] args) {
       //创建完这个对象后就已经先调用了代码块了
        Movie the_liangzai_lin = new Movie("The liangzai Lin");

    }
}
class Movie{
    private String name;
    private double price;
    private  String director;

    //普通代码块
    {
```

```
        System.out.println("语句1：");
        System.out.println("语句2：");
        System.out.println("语句3：");
    }
    //构造器
    public Movie(String name) {
        this.name = name;
    }
    //构造器
    public Movie(double price, String director) {
        this.price = price;
        this.director = director;
    }
    //构造器
    public Movie(String name, double price, String director) {
        this.name = name;
        this.price = price;
        this.director = director;
    }

}
```

三、代码块细节

Detail 01：

- **静态代码块：就是对类的初始化，而且它随着类的加载而执行，并且只会执行一次，若是 普通代码块，每创建一个对象就执行。**

Detail 02:

- **类什么时候被加载.（重要）**

1. **创建对象实例时.(new).**
2. **创建子类对象时,父类也会被加载.**
3. **使用类的静态成员时.**

```
package com.codeBlock;

public class Block02 {
    public static void main(String[] args) {
        /*类被加载的情况：*/
        //1.创建对象实例时会加载类
        //new AA();
        //2.创建子类对象实例时，父类也会被加载
        //new BB();//两个静态代码块都执行,并且先执行父类的,因为先加载父类再加载子类
        //使用类的静态成员时,类也会被加载，静态代码块也会被执行
        System.out.println(Cat.n1);//由于Cat类被加载了,所以静态代码块会被执行且只执行一次

    }
}
class Cat{
```

```
    public static int n1 = 9;
    static {
        System.out.println("Cat 的静态代码1被执行..");
    }
}
class AA{
    //静态代码块
    static {
        System.out.println("AA 的静态代码块1被执行");
    }
}
class BB extends  AA{
    static {
        System.out.println("BB的静态代码块2被执行...");
    }
}
```

Detail 03:

- **普通代码块在创建实例时(new),会被隐式调用,被创建一次，就会被调用一次.**
- **若是使用类的 静态成员 时,普通代码块 并不会执行.**

```
public class Block02 {
    public static void main(String[] args) {
        //调用对象的静态属性
        /*会调用h属性,静态代码块会被调用一次并且是先调用静态代码块,与普通代码块无关*/
        System.out.println(DD.h);
    }
}
class DD {
    public static  int h = 10;
    //静态代码块：类被加载时会被调用,而且只会被调用一次
    static {
        System.out.println("DD的静态代码块1.。");
    }
    //普通代码块：只有在new一个对象时会被调用,而且是每创建一个对象就会被调用一次
    {
        System.out.println("DD的普通代码块");
    }
}
```

Detail 04：(重点、难点)

- **先static ----> 再普通 ---->最后 构造方法.（注意是在一个类中）**

1. **调用一个 静态代码块 和 静态属性 初始化 (先静态).**

```
package com.main_;

public class Example02 {
    public static void main(String[] args) {
        A a = new A();
        /*分析:
```

```
         * 1.由于 (静态代码块) 和 (静态属性初始化)调用的优先级是一样的,
         * 若有 (多个代码块) 和 (多个静态变量) 初始化,
         * 则按他们定义的顺序调用
         * 2.所以此顺序是先执行n1属性, 调用getN1(),在执行代码块, 调用输出代码块内容*/
    }
}

class A{
    private static int n1 = getN1();

    {
        System.out.println("代码块输出的代码1...");
    }

    public static int getN1(){
        System.out.println("getN1()的执行...");
        return 100;
    }
}
```

2. **调用 代码块 和 普通属性 的初始化 (再普通).**
3. **最后 调用 构造方法.**

Detail 05：(代码块构造器注意事项)

```
class A{
    public A(){//构造器
        //这个位置隐藏的执行要求
        super();//1.有一个默认的的super, 要返回到父类执行构造器
                //2.调用本类的普通代码块 和
        System.out.println("ok");
    }
}
```

Detail 06： (继承代码块关系)

1. **父类的 静态代码块 和 静态属性 先执行(优先级一样, 按顺序执行)**
2. **子类的 静态代码块 和 静态属性 先执行(优先级一样, 按顺序执行)**
3. **父类的 普通代码块 和 普通属性 先执行(优先级一样, 按顺序执行)**
4. **父类的 构造方法.**
5. **子类的 普通代码块 和 普通属性 先执行(优先级一样, 按顺序执行)**
6. **子类的 构造方法.**

- **创建对象的执行过程：**

1. **进行类的加载(先加载父类【所以是 先执行 static】，再加载子类【所以其次执行 子类的 static】).**
2. **创建对象(先 按顺序 执行 普通代码,再执行 构造器【先 执行父类的 构造器内容 再执行子类的构造器内容】).**

```
public class CodeBlockDetail04 {
public static void main(String[] args) {
//老师说明
//(1) 进行类的加载
//1.1 先加载 父类 A02 1.2 再加载 B02
//(2) 创建对象
//2.1 从子类的构造器开始
//new B02();//对象
new C02();
}
}
class A02 { //父类
private static int n1 = getVal01();
static {
System.out.println("A02 的一个静态代码块..");//(2)
}
{
System.out.println("A02 的第一个普通代码块..");//(5)
}
public int n3 = getVal02();//普通属性的初始化
public static int getVal01() {
System.out.println("getVal01");//(1)
return 10;
}

public int getVal02() {
System.out.println("getVal02");//(6)
return 10;
}
public A02() {//构造器
//隐藏
//super()
//普通代码和普通属性的初始化...... System.out.println("A02 的构造器");//(7)
}
}
```

Detail 07：

1. **静态代码块 只能 直接调用 静态成员.(与静态方法一样)**
2. **普通代码 可以调用任意成员.（与普通方法一样）**

3. 题1：下面的代码输出什么？1min

```
class Person {
    public static int total;//静态变量
    static {//静态代码块
        total = 100;
        System.out.println("in static block!");//(1)
    }
}

public class Test {
    public static void main(String[] args) {
        System.out.println("total = "+ Person.total); //100
        System.out.println("total = "+ Person.total); //100
    }
}
```

。

- **结果: in ststic block! 100 100**
- **粗略分析:**

1. **total = 100 只是复制 然后输出Sys，最后Person.total 是 100**

练习2：

题2：下面的代码输出什么？ CodeBlockExercise02.java

```
class Sample
{
    Sample(String s)
    {
        System.out.println(s);
    }
    Sample()
    {
    System.out.println( “Sample默认构造函数被调用");
    }
}

class Test{
    Sample sam1=new Sample("sam1成员初始化");//
    static Sample sam=new Sample("静态成员sam初始化 ");//
    static{
        System.out.println("static块执行");//
        if(sam==null)System.out.println("sam is null");
    }
    Test()//构造器
    {
        System.out.println("Test默认构造函数被调用");//
    }
}
//主方法
public static void  main(String  str[])
{
    Test a=new Test();//无参构造器
}

//运行结果， 输出什么内容，并写出. 2min看看
1. 静态成员sam初始化
2. static块执行
3. sam1成员初始化
4.
```

。