

三个修饰符

Java Platform Standard Edition

郑春光

课程目标

CONTENTS

ITEMS **1** static

ITEMS **2** 静态成员、类加载

ITEMS **3** abstract

ITEMS **4** 抽象类、抽象方法

ITEMS **5** final

ITEMS **6** 最终类、方法，常量

static

Java Platform Standard Edition

实例属性

```
public class TestStaticField {  
    public static void main(String[] args) {  
  
        MyClass mc1 = new MyClass();  
        mc1.a = 10;  
  
        MyClass mc2 = new MyClass();  
        mc2.a = 20;  
  
        System.out.println(mc1.a + "\t" + mc2.a);  
    }  
}  
  
class MyClass {  
    int a; // 实例属性  
}
```

运行结果:
10 20

栈



mc1



mc2

堆



0x0000A001



0x0000B002

- 实例属性是每个对象各自持有的独立空间（多份），对象单方面修改，不会影响其他对象。

静态属性

```
public class TestStaticField {  
    public static void main(String[] args) {  
  
        MyClass mc1 = new MyClass();  
        mc1.b = 100;  
  
        MyClass mc2 = new MyClass();  
        mc2.b = 200;  
  
        System.out.println(mc1.b + "\t" + mc2.b);  
    }  
}  
  
class MyClass {  
    static int b; //静态属性  
}
```

运行结果:
200 200

栈

0x0000A001

mc1

0x0000B002

mc2

堆

省略上图

方法区

200

b

MyClass.class

MyClass对象的类信息

- 静态属性是整个类共同持有的共享空间（一份），任何对象修改，都会影响其他对象。

- 概念：
 - 静态 (static) 可以修饰属性和方法。
 - 称为静态属性 (类属性)、静态方法 (类方法)。
 - 静态成员是全类所有对象共享的成员。
 - 在全类中只有一份，不因创建多个对象而产生多份。
 - 不必创建对象，可直接通过类名访问。

- 练习：统计一个类的对象被创建过多少次？

静态方法

```
public class TestStaticMethod {  
    public static void main(String[] args) {  
        MyClass.method1();  
    }  
}  
  
class MyClass{  
    public static void method1(){  
        System.out.println("MyClass static method1()");  
        method2();  
    }  
  
    public static void method2(){  
        System.out.println("MyClass static method2()");  
    }  
}
```

可在其他类中，通过“类名.静态方法名”访问。

可在本类中，通过“静态方法名”访问。

由static修饰的静态方法。

- 已知静态方法：
- Arrays.copyOf();
- Arrays.sort();
- Math.random();
- Math.sqrt();
- 均使用类名直接调用。

- 静态方法允许直接访问静态成员。
- 静态方法不能直接访问非静态成员。
- 静态方法中不允许使用this或是super关键字。
- 静态方法可以继承，不能覆盖、没有多态。

动态代码块

```
public class TestDynamicBlock {  
    public static void main(String[] args) {  
        new MyClass();  
    }  
}  
  
class MyClass{  
    String field = "实例属性";  
  
    {  
        System.out.println(field);  
        System.out.println("动态代码块");  
    }  
  
    public MyClass(){  
        System.out.println("构造方法");  
    }  
}
```

创建对象时，触发动态代码块的执行。
执行地位：初始化属性之后、构造方法代码之前。
作用：可为实例属性赋值，或必要的初始行为。

运行结果：

实例属性
动态代码块
构造方法

- JVM首次使用某个类时，需通过CLASSPATH查找该类的.class文件。
- 将.class文件中对类的描述信息加载到内存中，进行保存。
 - 如：包名、类名、父类、属性、方法、构造方法...
- 加载时机：
 - 创建对象。
 - 创建子类对象。
 - 访问静态属性。
 - 调用静态方法。
 - `Class.forName(“全限定名”);`

静态代码块

```
public class TestStaticBlock {  
    public static void main(String[] args) {  
        MyClass.method();  
    }  
}  
  
class MyClass{  
    static String sField = "静态属性";  
  
    static{  
        System.out.println(sField);  
        System.out.println("静态代码块");  
    }  
  
    public static void method(){  
        /* 无代码  
        * 只为触发静态属性的初始化  
        * 和静态代码块的执行  
        */  
    }  
}
```

类加载时，触发静态代码块的执行（仅一次）。
执行地位：静态属性初始化之后。
作用：可为静态属性赋值，或必要的初始行为。

运行结果：

静态属性
静态代码块

注：方法只有被调用才会执行。

对象创建过程

静态

```
class Super{  
    static String SUPER_FIELD = "父类静态属性";  
  
    static{  
        System.out.println(SUPER_FIELD);  
        System.out.println("父类静态代码块");  
    }  
}
```

实例

```
String superField = "父类实例属性";  
  
{  
    System.out.println(superField);  
    System.out.println("父类动态代码块");  
}  
  
public Super(){  
    System.out.println("父类构造方法");  
}  
}
```

```
public class TestCreateObject {  
    public static void main(String[] args) {  
        new Super();  
        System.out.println("-----");  
        new Super();  
    }  
}
```

运行结果：

父类静态属性
父类静态代码块
父类实例属性
父类动态代码块
父类构造方法

父类实例属性
父类动态代码块
父类构造方法

带有继承的对象创建过程

静态

```
class Sub extends Super{  
    static String SUB_FIELD = "子类静态属性";  
  
    static{  
        System.out.println(SUB_FIELD);  
        System.out.println("子类静态代码块");  
    }  
}
```

实例

```
String subField = "子类实例属性";  
  
{  
    System.out.println(subField);  
    System.out.println("子类动态代码块");  
}  
  
public Sub(){  
    System.out.println("子类构造方法");  
}
```

```
public class TestCreateObject {  
    public static void main(String[] args) {  
        new Sub();  
    }  
}
```

运行结果：

父类静态属性
父类静态代码块
子类静态属性
子类静态代码块
父类实例属性
父类动态代码块
父类构造方法
子类实例属性
子类动态代码块
子类构造方法

- `static`修饰的成员为静态成员，无需创建对象，可直接通过类名访问。
- 静态方法不能直接访问非静态成员。
- 静态方法中不能使用`this`或`super`。
- 静态方法可以继承、不能重写、没有多态。
- 静态代码块在类加载时被执行，且只执行一次。

abstract

Java Platform Standard Edition

什么是抽象

- 似是而非的，像却又不是；具备某种对象的特征，但不完整。



生活中的抽象



无该被创建的对象

```
public class TestAbstract {  
    public static void main(String[] args) {  
        Animal a = new Animal();  
    }  
}  
  
class Animal{  
    String breed;  
    int age;  
    String sex;  
  
    public Animal(){}  
  
    public void eat(){  
        System.out.println("动物在吃...");  
    }  
  
    public void sleep(){  
        System.out.println("动物在睡...");  
    }  
}
```

Animal 仅是一种会吃会睡的对象，
再无其他行为，不够具体、不够完整。

程序是用来模拟现实世界、解决现实问题的，
现实世界中存在的都是“动物”具体的子类对象，
并不存在“动物”对象，
所以，Animal **不应该被独立创建成对象**。

如何限制这种对象的创建？

- 应用：abstract修饰类，此类不能new对象。

```
public class TestAbstract {  
    public static void main(String[] args) {  
        Animal a = new Animal();  
    }  
}  
  
abstract class Animal{  
    String breed;  
    int age;  
    String sex;  
  
    public Animal(){}  
  
    public void eat(){  
        System.out.println("动物在吃...");  
    }  
  
    public void sleep(){  
        System.out.println("动物在睡...");  
    }  
}
```

Animal是抽象的，无法实例化。

被abstract修饰的类，称为**抽象类**。
抽象类意为不完整的类、不够具体的类，
抽象类对象无法独立存在，即**不能new对象**。

抽象类的作用

```
public class TestAbstract {  
    public static void main(String[] args) {  
        Animal a1 = new Dog();  
        Animal a2 = new Cat();  
    }  
}  
  
abstract class Animal{  
    public Animal(){}  
  
    public void eat(){  
        System.out.println("动物在吃...");  
    }  
  
    public void sleep(){  
        System.out.println("动物在睡...");  
    }  
}  
  
class Dog extends Animal{  
  
}  
  
class Cat extends Animal{  
  
}
```

作用：

1. 可被子类继承，提供共性属性和方法。
2. 可声明为引用，强制使用多态。

经验：

抽象父类，可作为子类的组成部分，
依附于子类对象存在，
由**父类共性+子类独有**组成完整的子类对象。

不该被实现的方法

```
abstract class Animal{  
    public void eat(){  
        System.out.println("动物在吃...");  
    }  
  
    public void sleep(){  
        System.out.println("动物在睡...");  
    }  
}  
  
class Dog extends Animal{}  
  
class Cat extends Animal{}
```

需求：

Dog中的eat()应输出“狗在吃骨头”

Cat中的eat()应输出“猫在吃鱼”

父类提供的方法很难满足子类不同需求，
如不定义，则表示所有动物都不会吃、睡。
如定义，略显多余，多数会被子类覆盖。

方法声明必要，方法实现多余。

抽象方法

```
abstract class Animal{  
    public abstract void eat();  
  
    public void sleep(){  
        System.out.println("动物在睡");  
    }  
}  
  
class Dog extends Animal{  
    public void eat() {  
        System.out.println("狗在吃骨头");  
    }  
}  
  
class Cat extends Animal{  
    public void eat() {  
        System.out.println("猫在吃鱼");  
    }  
}
```

被abstract修改的方法，称为抽象方法，
只有方法声明，没有方法实现（{}的部分）。
意为不完整的方法，必须包含在抽象类中。

产生继承关系后，子类必须覆盖父类中所有的
抽象方法，否则子类还是抽象类。

- `abstract`修饰类：不能`new`对象，但可以声明引用。
- `abstract`修饰方法：只有方法声明，没有方法实现。（需包含在抽象类中）
- 抽象类中不一定有抽象方法，但有抽象方法的类一定是抽象类。
- 子类继承抽象类后，必须覆盖父类所有的抽象方法，否则子类还是抽象类。

final

Java Platform Standard Edition

什么是最终

- 概念：最后的，不可更改的。
- final可修饰的内容：
 - 类（最终类）
 - 方法（最终方法）
 - 变量（最终变量）

- final修饰类：此类不能被继承。
 - String、Math、System均为final修饰的类，不能被继承。
- final修饰方法：此方法不能被覆盖。
 - 意为最终方法，不支持子类以覆盖的形式修改。

- final修饰变量：此变量值不能被改变（常量）。

```
public class TestFinal {  
    public static void main(String[] args) {  
        final int num = 10;  
        num = 20;  
    }  
}
```

错误：无法为最终变量num分配值

所有final修饰的变量只能赋值一次，值不允许改变。

```
public class TestFinal {  
    public static void main(String[] args) {  
        new Student();  
    }  
}
```

错误：可能尚未初始化变量name

```
class Student{  
    final String name;// = "Tom"  
  
    {  
        //name = "tom";  
    }  
  
    public Student(){  
        //name = "tom";  
    }  
}
```

实例常量不再提供默认值，必须手动赋予初始值。

赋值时机：显示初始化、动态代码块、构造方法。

注意：如果在构造方法中为实例常量赋值，
必须保证所有的构造方法都能对其正确赋值。

静态常量

```
public class TestFinal {  
    public static void main(String[] args) {  
        System.out.println(Student.SCHOOL_NAME);  
    }  
}  
  
class Student{  
    static final String SCHOOL_NAME; // = "北京市第一中学"  
  
    static{  
        //SCHOOL_NAME = "北京市第一中学";  
    }  
}
```

错误：可能尚未初始化变量SCHOOL_NAME

静态常量不再提供默认值，必须手动赋予初始值。

赋值时机：显示初始化、静态代码块。

```
public class TestFinal {  
    public static void main(String[] args) {  
  
        final int num = 100;  
        num += 20;  
  
        final int[] nums = new int[]{11,22,33};  
        nums = new int[5];  
  
        final Student s = new Student();  
        s = new Student();  
    }  
}  
  
class Student{  
    String name;  
}
```

final修饰基本类型：值不可变

final修饰引用类型：地址不可变

- final修饰类：此类不能被继承。
- final修饰方法：此方法不能被覆盖。
- final修饰变量：此变量值不能被改变。（无初始值、只允许赋值一次）
 - 局部常量：显示初始化。
 - 实例常量：显示初始化、动态代码块、构造方法。
 - 静态常量：显示初始化、静态代码块。
 - 基本数据类型常量：值不可变。
 - 引用数据类型常量：地址不可变。