三个修饰符

> Author：zhangzhang
>
> Version：1.0.0

一、引言

1.1 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。

二、abstract

2.1 生活中的抽象

2.2 不该被创建的对象

```java
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?

三、抽象类、抽象方法【 重点 】

3.1 抽象类

应用：abstract修饰类，此类不能new对象。

```java
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。

- 抽象类意为不够完整的类、不够具体的类，
- 抽象类对象无法独立存在，即不能new对象。

3.2 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：
  - 可被子类继承，提供共性属性和方法。
  - 可声明为引用，更自然的使用多态。

- 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。

3.3 不该被实现的方法

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”。

```
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{}
```

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。

3.4 抽象方法

```java
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");
    }
}
```

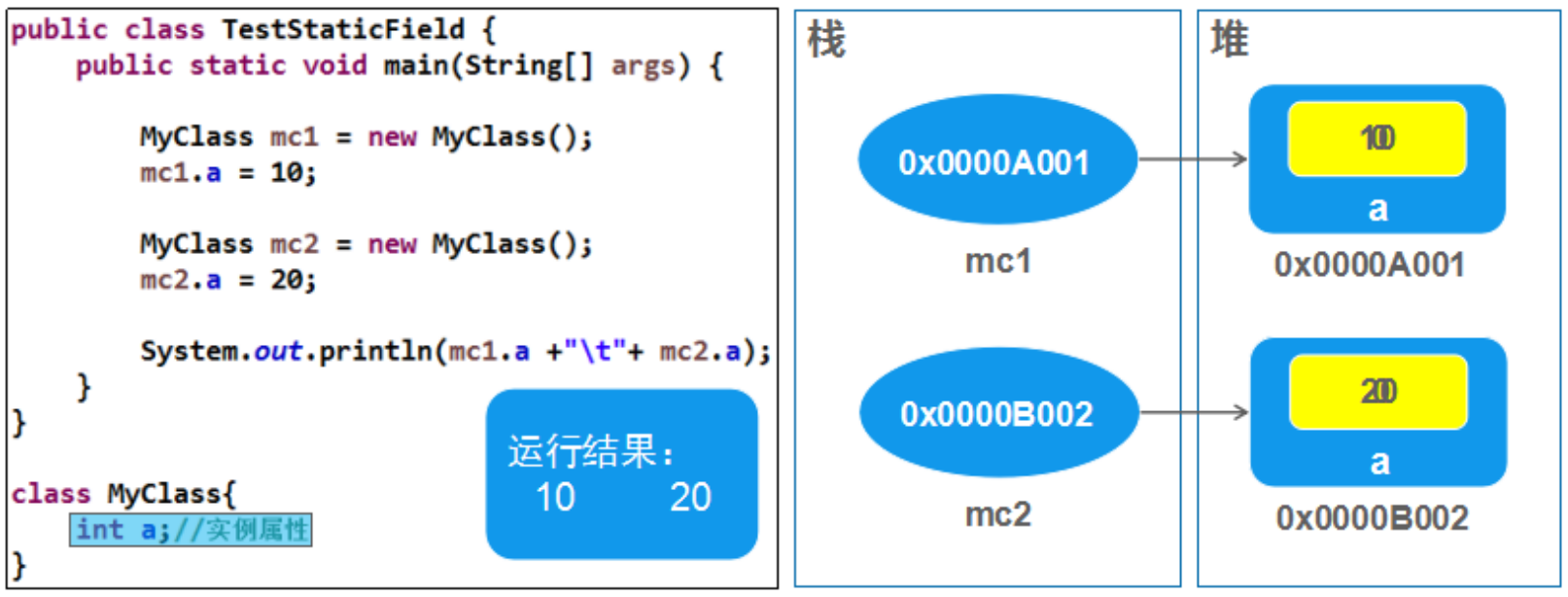
被abstract修改的方法，称为抽象方法，**只有方法声明，没有方法实现**（{}的部分）。意为不完整的方法，**必须包含在抽象类中**。

产生继承关系后，子类必须重写父类中所有的抽象方法，否则子类还是抽象类。

```java
public class TestAnimal {
  public static void main(String[] args) {
    //Animal animal=new Animal();
    Animal dog=new Dog();
    Animal cat=new Cat();

    dog.eat();
    cat.eat();

  }
}

//动物类
abstract class Animal {

  String breed;//品种
  int age;//年龄
  String sex;//性别

  //吃(抽象方法)
  public abstract void eat();
  //睡
  public void sleep() {
    System.out.println("动物睡...");
  }
}

//狗狗类
class Dog extends Animal{
  @Override
  public void eat() {
    System.out.println("狗在吃骨头");
  }
}

//猫咪类
class Cat extends Animal{
  @Override
  public void eat() {
    System.out.println("猫在吃鱼");
  }
}
```

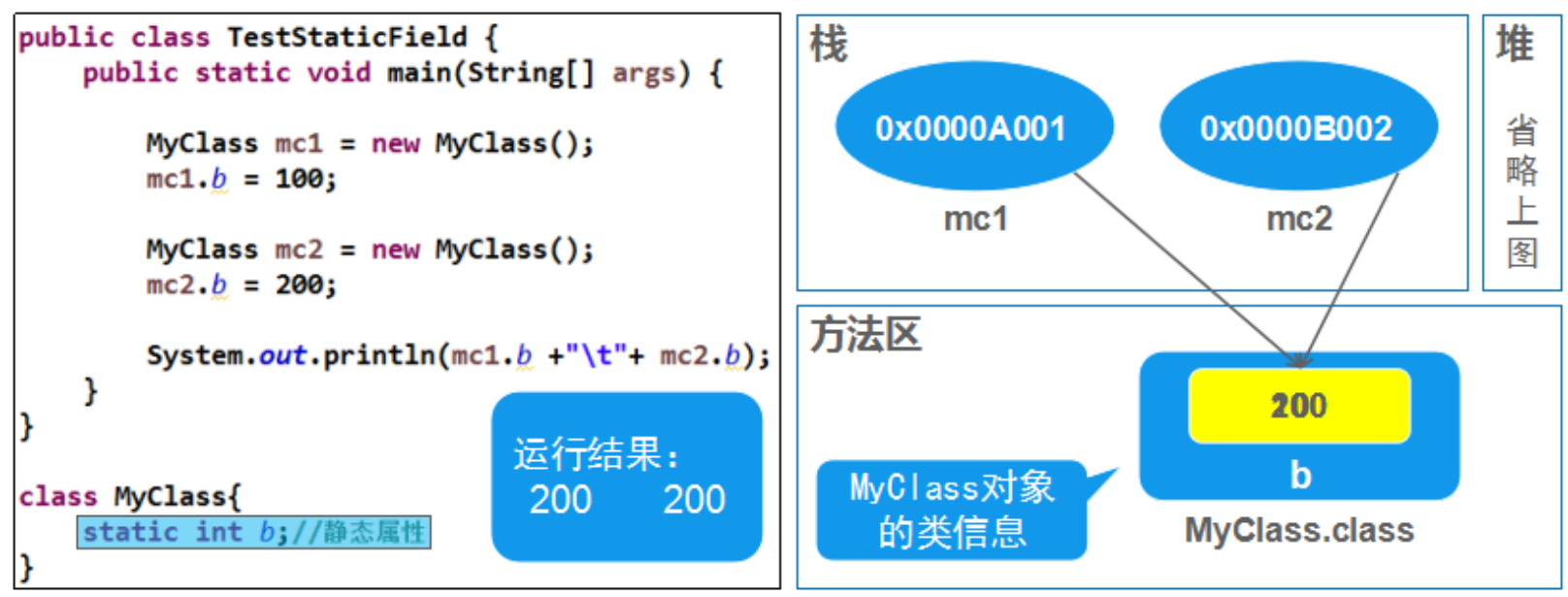
四、静态属性

4.1 实例属性



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。

4.2 静态属性



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。

4.3 什么是静态

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。

4.4 课堂案例

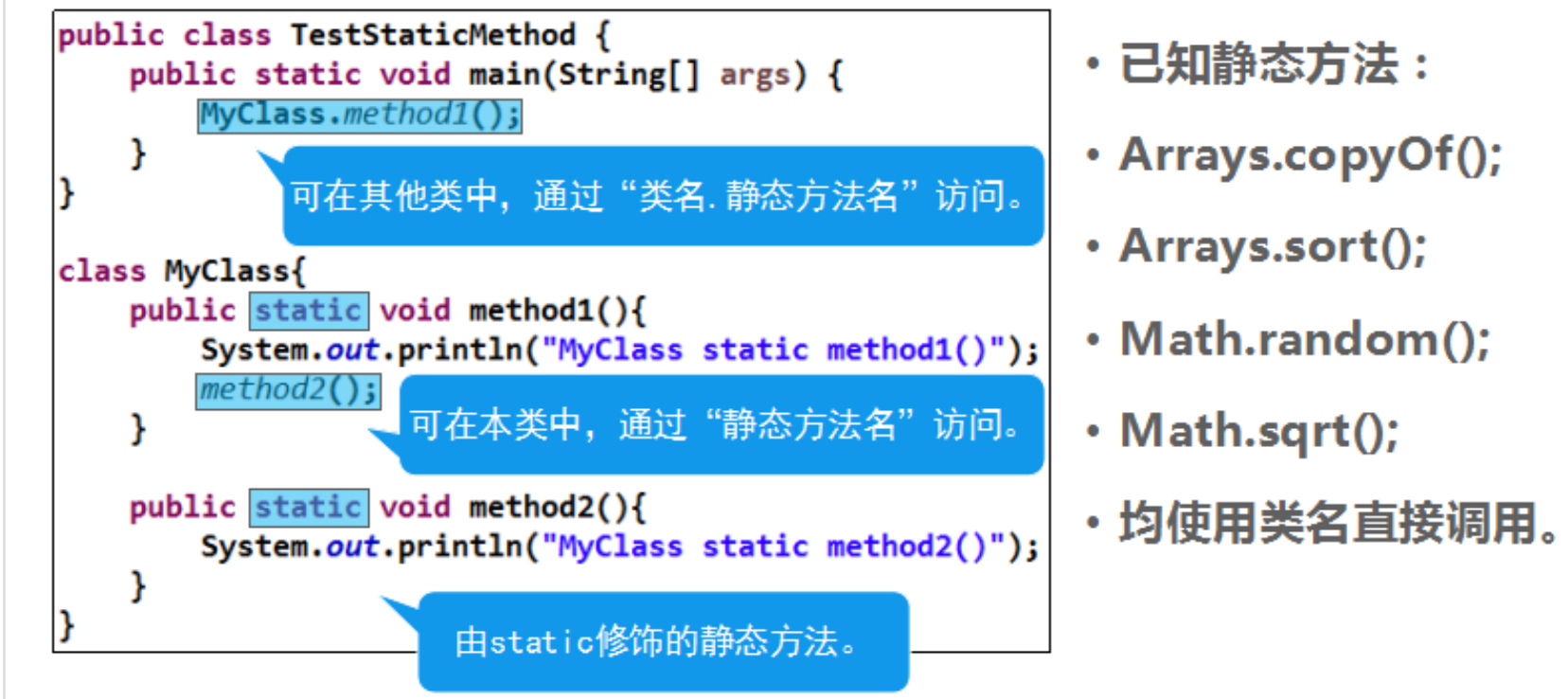
练习：统计一个类的对象被创建过多少次?

```
public class TestTeacher {
  public static void main(String[] args) {
    System.out.println("对象创建之前次数:"+Teacher.count);
    Teacher t1=new Teacher();
    Teacher t2=new Teacher();
    Teacher t3=new Teacher();
    System.out.println("对象创建之后次数:"+Teacher.count);
  }
}
//老师类
class Teacher {
  //姓名
  String name;
  //年龄
  int age;
  //工资
  double salary;
  //保存对象创建的次数
  static int count=0;

  public Teacher() {
    //count
    count++;
  }
```

```
    public void show() {
      System.out.println(name+"---"+age+"---"+salary);
    }
}
```

五、静态方法、类加载

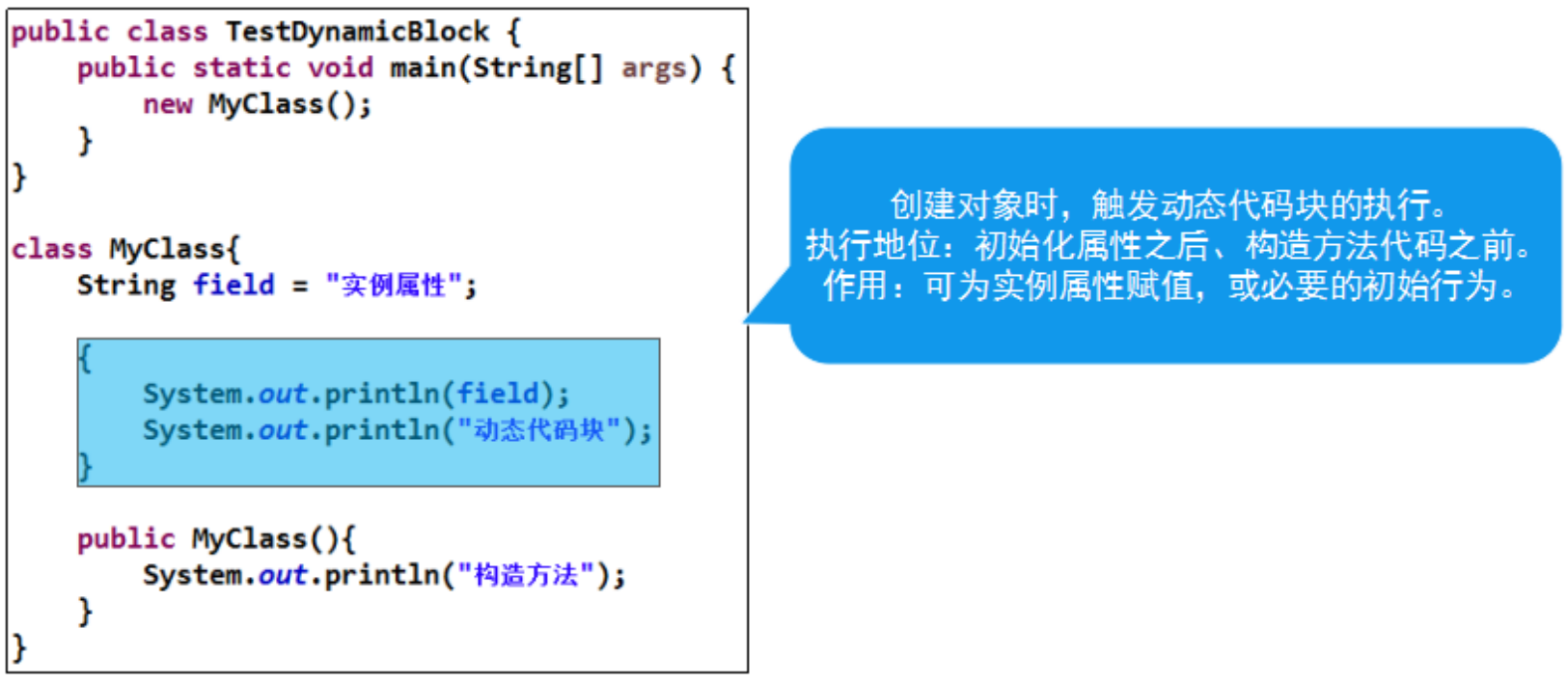
5.1 静态方法



5.2 静态的特点

> - 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，不能重写、没有多态。

5.3 动态代码块



> 运行结果：
> 实例属性
> 动态代码块
> 构造方法

5.4 类加载

> 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。

- 主动加载：Class.forName(“全限定名”);

5.4 静态代码块

```
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：
静态属性

注：方法只有被调用才会执行。

六、final

6.1 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）

6.2 final类

- 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。

6.3 final变量

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。

6.4 实例常量

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

class Student{
    final String name;// = "Tom"

    {
        //name = "tom";
    }

    public Student(){
        //name = "tom";
    }
}
```

错误：可能尚未初始化变量name

- 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。

6.5 静态常量

```java
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。

6.6 对象常量

```java
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