第十一章 static与final

1. Static

1.1 static概述

1. staitc是一个修饰符，表示静态的
2. static可以修饰：
   a. 类：静态内部类（后面章节讲）
   b. 属性（字段|状态|成员变量）
      i. static 修饰的字段，不属于对象，属于类
      ii. 在类加载的时候分配的内存，加载完毕即可使用
      iii. 通过类名称直接访问

代码块

```
package com.powernode.static05;

class Student {
    public String name = "zs";
    public static double classFee = 8000;
}

public  class Test {
    public static void main(String[] args) {
        //静态变量：类名称直接可以访问
        System.out.println(Student.classFee);
        //实例变量：也叫对象变量，只有对象才可以访问
        System.out.println(new Student().name);
    }
}
```

   c. 什么情况下使用static修饰属性呢？
3. 方法

1.2 static修饰属性

1. 什么情况下使用static修饰属性呢？

- 对象共享的属性使用staitc修饰，比如：班级费用，是所有学员共享的属性

2. 对象共享的属性，任何对象对其修改，都会对其他对象有影响
3. static修饰的变量不属于对象，属于类，也叫类变量或者静态变量

代码块

```
package com.powernode.static06;

class Teacher{
    String name ;
    int age;
    //部门经费
    static double deptExpenditure = 8000;
}
public class Test {
    public static void main(String[] args) {
        //静态变量：类名称直接访问
        System.out.println(Teacher.deptExpenditure);
        System.out.println(new Teacher().age);
    }
}
```

1.3 基于内存的角度理解staitc

- static修饰的变量类加载分配空间
- 分配在元空间中（元空间主要用于存放类的元数据，像类的结构、静态变量、方法字节码等都包含在内）

代码块

```
package com.powernode.static07;

class Teacher {
    String name;
    int age;
    //部门经费
    static double deptExpenditure = 8000;
}

public class Test {
    public static void main(String[] args) {
        System.out.println(Teacher.deptExpenditure);//8000
        Teacher t1 = new Teacher();
```

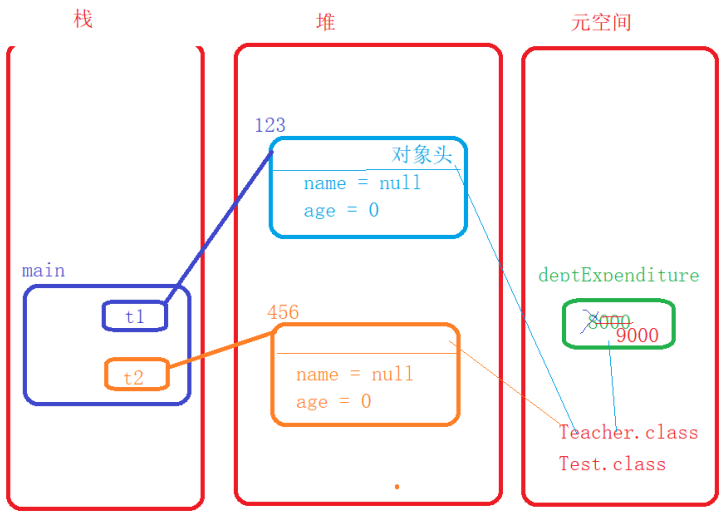
```
14            System.out.println(t1.name);
15            System.out.println(t1.age);
16            t1.deptExpenditure += 1000;
17            System.out.println(t1.deptExpenditure);//9000
18            System.out.println(Teacher.deptExpenditure);//9000
19
20            Teacher t2 = new Teacher();
21            System.out.println(t2.deptExpenditure);//9000
22        }
23    }
```

```
package com.powernode.static07;

class Teacher {
    String name;
    int age;
    //部门经费
    static double deptExpenditure = 8000;
}

public class Test {
    public static void main(String[] args) {
        System.out.println(Teacher.deptExpenditure);//8000
        Teacher t1 = new Teacher();
        System.out.println(t1.name);
        System.out.println(t1.age);
        t1.deptExpenditure += 1000;
        System.out.println(t1.deptExpenditure);//9000
        System.out.println(Teacher.deptExpenditure);//9000

        Teacher t2 = new Teacher();
        System.out.println(t2.deptExpenditure);//9000
    }
}
```



1.4 static修饰方法

1. static修饰的方法称为静态方法
2. 类加载后即可使用，不需要创建对象
3. 什么情况下使用static修饰一个方法?
   a. 不需要创建对象，即可访问的方法可以使用static修饰
      - 比如：数据运算的方法，求最大值，最小值，一个数的平台等
   b. 如果static修饰的属性是private，通常情况下，会提供static 的set和get方法

代码块

```
1   package com.powernode.static08;
2
3   class MyMath{
4       public static int add(int x ,int y){
5           return x + y;
6       }
```

```
    public static int max(int x,int y){
        return x > y ? x:y;
    }
}
class Teacher {
    private String name;
    private int age;
    //部门经费
    private static double deptExpenditure = 8000;

    public static double getDeptExpenditure() {
        return deptExpenditure;
    }
}
public class Test{
    public static void main(String[] args) {
        System.out.println(MyMath.add(2, 3));
        System.out.println(MyMath.max(2, 3));
        System.out.println(Teacher.getDeptExpenditure());
    }
}
```

1.5 静态成员和非静态成员互相访问

代码块

```
package com.powernode.static09;

class Teacher{
    private String name = "zs";
    private static double deptExpenditure = 8000;

    /**
     * 1.静态方法
     *    1.静态方法不可以访问实例变量
     *    2.静态方法不可以访问实例方法
     */
    public static void method01(){
        System.out.println(deptExpenditure);
        //Non-static field 'name' cannot be referenced from a static context
        //System.out.println(name);静态方法不可以访问实例变量
        method02();
       // method03();
    }
    public static void method02(){}

```

```
21      /**
22       * 2.实例方法
23       *   1.实例变量
24       *   2.实例方法
25       *   3.静态变量
26       *   4.静态方法
27       * 问? 实例方法为什么可以访问静态成员呢?
28       *   1.因为实例方法可以访问的时候，说明对象已经创建
29       *   2.创建对象时，会先把类加载进来
30       *   3.类加载完毕，静态变量在元空间中分配了内存，静态方法即可使用
31       *   4.所以实例方法可以访问静态成员
32       *   5.静态方法不可以访问实例成员
33       */
34      public  void method03(){
35          System.out.println(name);
36          System.out.println(deptExpenditure);
37          method02();
38      }
39
40  }
41  public class Test {
42
43  }
```

1.6 单例设计模式（饿汉式一）

代码块

```
1   package com.powernode.static10;
2
3   class Chairman{
4       //1.把构造器封装起来（创建对象需要用到构造器，构造器不能让其他类访问）
5       private Chairman(){
6           System.out.println("Chairman.Chairman");
7       }
8       //2.类加载的时候创建对象(类只加载一次，所以在类加载的时候创建比较合适)
9       private static Chairman chairman = new Chairman();
10
11      //3.提供get方法获得对象
12      public static Chairman getInstance() {
13          return chairman;
14      }
15  }
16  public class Test {
17      public static void main(String[] args) {
18          /**
```

```
19            * 需求:
20            *   1.一个公司只有一个董事长
21            *   2.董事长类只能创建一个对象
22            * 一个类只能创建一个对象，这个类就是单例（只能有一个实例对象）
23            */
24           Chairman instance = Chairman.getInstance();
25           System.out.println(instance);
26       }
27   }
```

1.7 单例设计模式（饿汉式二）

- 在类加载的时候创建对象，同时还需要处理其他的业务
- 我们就可以使用static块

代码块

```
1   package com.powernode.static11;
2
3   class Chairman{
4       //1.把构造器封装起来（创建对象需要用到构造器，构造器不能让其他类访问）
5       private Chairman(){
6           System.out.println("Chairman.Chairman");
7       }
8       //2.类加载的时候创建对象(类只加载一次，所以在类加载的时候创建比较合适)
9       private static Chairman chairman;
10      static{
11          chairman = new Chairman();
12          //处理其他的业务
13          System.out.println("------只能处理去一次-------");
14      }
15
16      //3.提供get方法获得对象
17      public static Chairman getInstance() {
18          return chairman;
19      }
20  }
21  public class Test {
22      public static void main(String[] args) {
23          /**
24           * 需求:
25           *   1.一个公司只有一个董事长
26           *   2.董事长类只能创建一个对象
27           * 一个类只能创建一个对象，这个类就是单例（只能有一个实例对象）
28           */
29          Chairman instance = Chairman.getInstance();
```

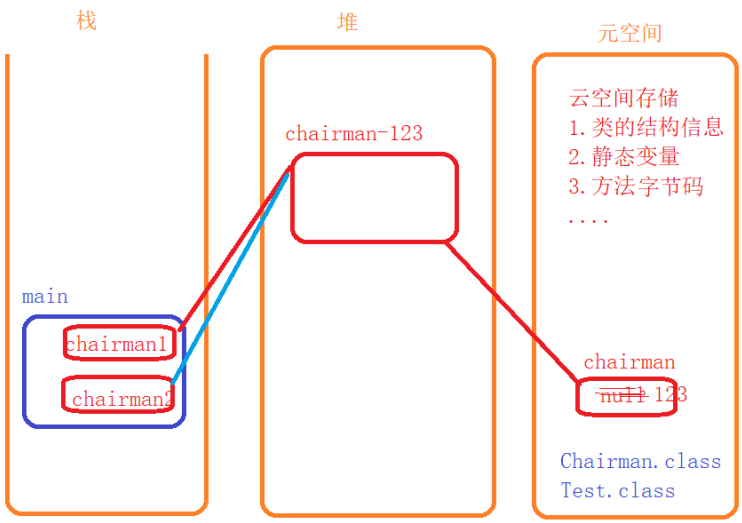
```
30          System.out.println(instance);
31      }
32  }
```

1.8 单例设计模式（懒汉式）

代码块

```
1   package com.powernode.static12;
2
3   class Chairman{
4       //1.把构造器封装
5       private Chairman(){
6           System.out.println("Chairman.Chairman");
7       }
8
9       //2.声明一个Chairman类型的静态变量
10      private static Chairman chairman;
11      //3.获得对象
12      public static Chairman getInstance(){
13          if (chairman == null){
14              chairman = new Chairman();
15          }
16          return chairman;
17      }
18  }
19  public class Test {
20      public static void main(String[] args) {
21          Chairman chairman1 = Chairman.getInstance();
22          Chairman chairman2 = Chairman.getInstance();
23          System.out.println(chairman1 == chairman2);
24      }
25  }
```

```
class Chairman{
    //1.把构造器封装
    private Chairman(){
        System.out.println("Chairman.Chairman");
    }
    //2.声明一个Chairman类型的静态变量
    private static Chairman chairman;
    //3.获得对象
    public static Chairman getInstance(){
        if (chairman == null){
            chairman = new Chairman();
        }
        return chairman;
    }
}
public class Test {
    public static void main(String[] args) {
        Chairman chairman1 = Chairman.getInstance();
        Chairman chairman2 = Chairman.getInstance();
        System.out.println(chairman1 == chairman2);
    }
}
```



1.9 总结

- 饿汉式
  - 类加载的时候创建对象，无论是否使用都会创建
  - 不存在线程安全问题
- 懒汉式
  - 需要用到才会创建
  - 存在线程安全问题

1.10 静态块，实例块和构造器的执行顺序

代码块

```
package com.powernode.static13;

public class Test {
    static {
        System.out.println("静态块");
    }
    {
        System.out.println("实例块");
    }
    Test(){
        System.out.println("构造器");
    }

    public static void main(String[] args) {
        new Test();
```

```
        //new Test();
        /**
         * 输出结果:
         *    静态块     // 类加载时执行静态代码块
         *    实例块     // 创建对象时先执行实例代码块
         *    构造器     // 最后执行构造方法
         */
    }
}


class Test01{
    static {
        System.out.println("静态块");
    }
    {
        System.out.println("实例块");
    }
    Test01(){
        this(10);//调用重载构造器
    System.out.println("构造器1");
    }
    public Test01(int i){
        System.out.println("构造器2");
    }

    public static void main(String[] args) {
        new Test01();
        /**
         * 1.执行顺序: 静态块->Test01()->Test01(int i) ->实例块
         * 2.输出结果: 静态块->实例块->构造器2->构造器1
         */
    }
}
```

1.11 JVM的懒加载机制

代码块

```
package com.powernode.static14;

class Cat{
    static {
        System.out.println("Cat.static initializer");
    }
}
```

```java
class Dog{
    static {
        System.out.println("Dog.static initializer");
    }
}
public class Test {
    public static void main(String[] args) {
        /**
         * 1.只加载了Cat.class，执行静态块且只执行一次
         * 2.Dog类不加载，静态块不执行
         * 3.需要Dog的时候才会加载，这就是懒加载
         * 4.JVM就是懒加载的机制
         * 5.new了两个Cat对象，也只输出一次静态块的内容
         */
        new Cat();
        new Cat();
    }
}
```

2. final

2.1 final的概述（会背）

1. final：最终的
2. 可以修饰：
   a. 类：不可以被继承
   b. 变量
      i. 一旦赋值，不可修改
      ii. **final修饰的变量称为常量**
      iii. 常量都大写
   c. 方法：不可以被覆盖

2.2 final修饰类

代码块

```java
package com.powernode.final15;

/**
 * 使用final类的核心原则是：
```

```
 *     当一个类不应该或不需要有子类时，将其声明为final。
 *     这样做可以保证类的安全性、不可变性，或者实现性能优化。
 *     不过，过度使用final会让代码变得僵化，所以要根据具体的设计需求来合理使用。
 */
/*final class Person{}*/
class Person{}
class Teacher extends Person{

}

public class Test {
}
```

2.3 final修饰变量

- 现实生活中
  - 比如PI在整个项目中赋值3.14，PI的值不能改变，这样PI可以使用final修饰
  - 公务员的国籍（中国）：不能改变

代码块

```
package com.powernode.final16;
class IMath{
    //静态常量（static final）
    public static final double PI = 3.14;
}
class CivilServant{
    public String name;
    public final String COUNTRY = "中国";

    /*public void setCOUNTRY(String COUNTRY) {
        this.COUNTRY = COUNTRY;
    }*/
}
public class Test {
    public static void main(String[] args) {
        //IMath.PI = 3.1415;Cannot assign a value to final variable 'PI'
        CivilServant civilServant = new CivilServant();
        civilServant.name = "zs";
        //civilServant.COUNTRY = "日本";java: 无法为 final 变量 COUNTRY 分配值
    }
}
```

2.4 final变量赋值

2.4.1 实例变量

代码块

```
package com.powernode.final17;


class CivilServant{
    public String name;
    public final String COUNTRY;
    //构造器中赋值
    /*public CivilServant(String COUNTRY, String name) {
        this.COUNTRY = COUNTRY;
        this.name = name;
    }*/
    //实例块中赋值
    {
        COUNTRY = "中国";
    }
}
public class Test {
}
```

2.4.2 静态变量

代码块

```
package com.powernode.final18;

class IMath{
    public static final double PI;
    static {
        PI = 3.14;
    }
}
public class Test {
}
```

2.5 final修饰方法

代码块

```java
package com.powernode.final19;

class Person{
    public final void eat(){
        System.out.println("Person.eat");
    }
}
class Teacher extends Person{
    //'eat()' cannot override 'eat()' in 'com.powernode.final19.Person'; overridden method is final
   /* @Override
    public void eat() {
        super.eat();
    }*/
}
public class Test {
}
```

- 什么情况下，使用final修饰一个方法呢?