姓名：张煜 学号：2208300002

1. **static**

题目要求：

请解释什么是static关键字，并列举至少3种用法。

**答：**

1. 静态变量

静态变量也称为类变量，是在类中定义的变量，前面加上了 static 关键字，在类加载时创建。这些变量由该类的所有实例共享。也就是说，无论创建了多少个类的实例，静态变量只有一个副本存在于Java虚拟机（JVM）中，所有对象共用同一份数据，仅占用一份内存空间。

2. 静态方法

静态方法是属于类的方法，而不是某个对象的方法。可以在没有创建类的实例的情况下调用静态方法。静态方法只能直接访问静态成员变量和其他静态方法。静态方法内不能使用this关键字。

3. 静态初始化块

静态初始化块用来初始化类，而不是类的实例，在类加载时给静态成员属性赋值，在类加载时才会执行，它在类首次被加载到JVM中时执行一次。不能访问非静态成员变量。静态代码块有限执行于构造代码块。静态代码块和静态成员属性，谁在代码前，谁先执行。

4. 静态内部类

静态内部类是一种特殊类型的内部类，它不需要依赖于外部类的实例即可创建。静态内部类不能访问外部类的非静态成员。

1. **静态成员变量**

**题目要求： 请从以下角度思考静态成员变量和非静态成员变量的区别：**

**1. 存储位置：**存储在Java堆中的静态区，所有的实例共享这一块区域的数据。

**2. 生命周期：**随着类的加载而存在，直到JVM退出才销毁。也就是说，静态成员变量的生命周期与类相同。

**3. 出现顺序：** 静态成员变量先于任何对象实例化之前就已经存在，因为类加载时静态成员变量就已经被初始化**。**

**4. 调用方式：**可以通过类名直接访问，而不需要创建类的实例。

**5. 初始化时机：**静态成员变量在类加载时就会被初始化，这通常发生在程序启动或首次引用该类的静态成员时。

**6. 内存占用：** 不管创建多少个对象实例，静态成员变量只会占用一块内存空间。

**7. 共享：**被所有实例共享，意味着如果一个对象修改了静态成员变量的值，那么所有对象看到的都是同一个更新后的值。

1. **静态方法**

**题目要求： 请简述static方法和非static方法有什么区别？什么情况下应该使用static方法？**

**答：**

1. **区别**
2. 所属关系

静态方法属于类级别，非静态方法数据对象级别，即使没有实例化对象也可以直接访问，与类的所有实例共享。

2）访问限制

只能直接访问类的静态成员属性和其他静态方法，由于静态方法不是与特定对象关联，故不能直接访问非静态成员变量或调用非静态方法。

3）对象依赖

不依赖对象的方法行为，故不需要实例化对象就可以调用。

4）内存使用

静态方法在内存中仅占有一份，不随对象的增加而增加对内存空间的占用

2、**使用静态方法的情况：**

1）工具方法

当一个方法操作的对象是与对象无关的通用操作可以考虑静态方法，例如：对数学计算，对于链表、数组、图、堆、栈、等数据结构进行操作处理。

1. 常量容器

当类主要用于提供一些常量或者配置信息时，可以将这些常量定义为静态变量，并提供静态方法来获取这些常量的值。

1. 单例模式

在实现单例模式时，通常会有一个静态方法用来返回单例对象的唯一实例。

1. 类级别操作

当方法的操作直接与类有关，而不涉及具体的对象状态时，适合使用静态方法。

1. **代码块**

**答：**

匿名代码块 (Instance Initialization Block)

* 定义：没有名称的代码块，用于初始化类的新实例。
* 执行时机：每当创建一个新的类实例时，匿名代码块就会被执行。
* 作用：确保每个对象在创建时都能得到一致的初始化状态。
* 示例：在类定义中，不在任何方法内，也不带有任何修饰符的代码块 {}。

静态代码块 (Static Initialization Block)

* 定义：带有 static 关键字的代码块，用于类级别的初始化。
* 执行时机：在类初次加载到 JVM 时执行，并且只执行一次。
* 作用：初始化静态变量或执行一次性的类级别的初始化操作。
* 示例：在类定义中，不在任何方法内，带有 static 关键字的代码块 static {}。

1. **继承static**

**答：**

1. 访问而非继承：虽然 static 成员不是被继承的，但是子类仍然可以访问父类中的 static 成员，只要满足访问权限的要求。
2. 重命名：如果子类想要拥有一个与父类同名的 static 成员，它可以声明自己的 static 成员，但这不是继承的结果，而是重新声明了一个同名的 static 成员。
3. 隐藏：当子类声明了一个与父类同名的 static 成员时，实际上是在子类中“隐藏”了父类的 static 成员。这意味着当从子类中访问该成员时，将优先访问子类的 static 成员。
4. 静态方法的重写：不能重写静态方法。静态方法只能被隐藏，即子类可以定义一个与父类静态方法同名的方法，但这不是一个真正的重写过程，因为静态方法是通过类名而不是通过实例来调用的
5. **程序分析**

**案例一、**

1. class Test {

2.     public Test() {

3.         System.out.println("构造器");

4.     }

5.     public void info() {

6.         System.out.println("info");

7.     }

8.     static {

9.         System.out.println("test static 1");

10.     }

11.     public static void main(String[] args) {

12.         new Test().info();

13.     }

14.     {

15.         System.out.println("代码块");

16.     }

17.     static {

18.         System.out.println("test static 2");

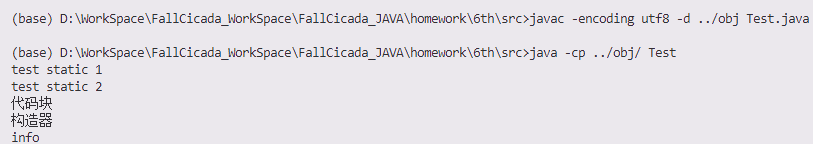
19.     }

20. }

21.

**分析及结果：**

1. **静态初始化块的执行：**
   * **在Java中，当类被加载时，所有的静态初始化块（static块）将按照它们在代码中的出现顺序优先被执行。**
   * **在这个例子中有两个静态初始化块：**
     + **System.out.println("test static 1");**
     + **System.out.println("test static 2");**
   * **因此，在程序开始执行之前，控制台会先输出这两条信息。**
2. **非静态初始化块的执行：**
   * **非静态初始化块（不带static关键字的代码块）会在每次创建对象时被执行，并且是在构造器调用之前执行。**
   * **这里有一个非静态初始化块：**
     + **System.out.println("代码块");**
   * **当new Test()被执行时，这个块会被触发。**
3. **构造器的执行：**
   * **构造器在对象创建时被调用。**
   * **在这个例子中，构造器会输出：**
     + **System.out.println("构造器");**
4. **方法调用：**
   * **最后，在main方法中，new Test().info();创建了一个Test实例并调用了info方法，输出了info。**
5. **运行结果**

****

**案例二、**

1. public class Test06 {

2.     public static void main(String[] args) {

3.         Child child = new Child();

4.     }

5. }

6.     class Parent {

7.         static {

8.             System.out.println("静态代码块Parent");

9.         }

10.         {

11.             System.out.println("构造代码块Parent");

12.         }

13.         public Parent() {

14.             System.out.println("构造方法Parent");

15.         }

16.         }

17.         class Child extends Parent {

18.         static {

19.             System.out.println("静态代码块Child");

20.         }

21.         {

22.             System.out.println("构造代码块Child");

23.         }

24.         public Child() {

25.             System.out.println("构造方法Child");

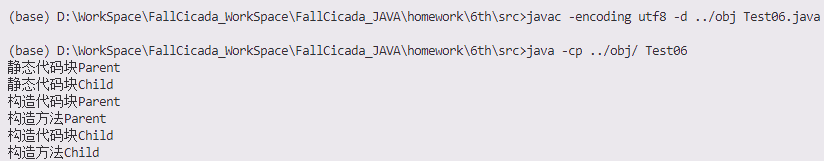
26.         }

27. }

28.

**运行及分析：**

1. **静态初始化块：**
   * **静态代码块在类加载时执行，并且只执行一次。首先，父类Parent的静态代码块会被执行，接着是子类Child的静态代码块。因此，输出将是：**
     + **"静态代码块Parent"**
     + **"静态代码块Child"**
2. **实例初始化块和构造方法：**
   * **当创建Child类的实例时，首先调用父类Parent的构造链。这意味着Parent类的构造代码块和构造方法将被执行，之后才是Child类自身的构造代码块和构造方法。**
   * **执行顺序为：**
     + **"构造代码块Parent"**
     + **"构造方法Parent"**
     + **"构造代码块Child"**
     + **"构造方法Child"**
3. **运行结果：**

****

**案例三、**

1. class B {

2.     public static B b = new B();

3.     public static B b2 = new B();

4.     {

5.         System.out.println("构造块");

6.     }

7.     static {

8.         System.out.println("静态块");

9.     }

10. }

11.     public class Test06 {

12.     public static void main(String[] args) {

13.         B b = new B();

14.     }

15. }

16.

**运行及分析：**

1. **静态初始化块的执行：**
   * **静态初始化块会在类被加载时执行。**
   * **B类的静态初始化块会首先执行，输出：**

**静态块**

1. **静态变量的初始化：**
   * **在类被加载时，静态变量也会被初始化。**
   * **B类有两个静态变量 b 和 b2：**
     + **public static B b = new B();**
     + **public static B b2 = new B();**
   * **初始化这两个静态变量时，会分别创建两个 B 对象，并执行各自的构造过程。**
2. **非静态初始化块的执行：**
   * **每次创建对象时，非静态初始化块会在构造器调用之前执行。**
   * **创建 B 对象时，会执行非静态初始化块，输出：**

**构造块**

* **然后会调用默认构造器（默认构造器没有输出）。**

**具体步骤如下：**

1. **加载 B 类时，首先执行静态初始化块，输出：**

**静态块**

1. **初始化静态变量 b：**
   * **创建 B 对象，执行非静态初始化块，输出：**

**构造块**

* **调用默认构造器（没有输出）。**

1. **初始化静态变量 b2：**
   * **再次创建 B 对象，执行非静态初始化块，输出：**

**构造块**

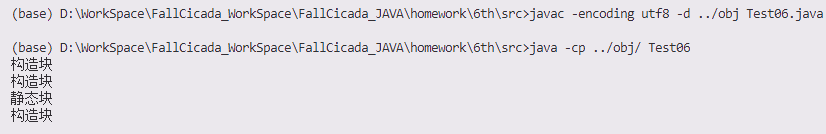
* **调用默认构造器（没有输出）。**

1. **在 main 方法中创建新的 B 对象：**
   * **执行非静态初始化块，输出：**

**构造块**

* **调用默认构造器（没有输出）。**

**5．运行**

****

1. **单例模式**

**单例模式是一种常用的软件设计模式，它保证一个类只有一个实例，并提供一个全局访问点。这种模式通常用于那些需要频繁实例化然后销毁的对象，或者创建对象需要消耗大量资源的情况，通过单例模式可以避免对系统资源的浪费，同时也简化了对这些实例的访问。**

**在单例模式中，我们通常会采取以下步骤来实现：**

1. **将类的构造函数设为私有，防止其他对象实例化该类。**
2. **定义一个静态私有变量来保存该类的唯一实例。**
3. **提供一个公共的静态方法来返回这个唯一实例。**

**例子：**

1. public class Singleton {

2. // 创建 Singleton 的一个对象

3. private static Singleton instance = new Singleton();

4.

5. // 私有构造方法，防止外部实例化

6. private Singleton() {}

7.

8. // 静态的获取实例方法

9. public static Singleton getInstance() {

10. return instance;

11. }

12.

13. // 假设的一个方法

14. public void showMessage() {

15. System.out.println("Hello, I am a singleton!");

16. }

17. }

18.

1. **统计次数**

**代码**

1. public class Test05 {

2.     // 静态变量用于记录所有实例的数量

3.     private static int Count = 0;

4.

5.     // 私有构造函数，防止外部直接new

6.     private Test05() {

7.         // 每当创建一个新实例时，instanceCount加1

8.         Count++;

9.     }

10.

11.     // 提供一个静态方法，允许外部获取实例数量

12.     public static int getCount() {

13.         return Count;

14.     }

15.

16.     // 测试代码

17.     public static void main(String[] args) {

18.         Test05 obj1 = new Test05();

19.         Test05 obj2 = new Test05();

20.         Test05 obj3 = new Test05();

21.

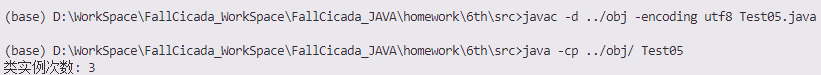
22.         System.out.println("类实例次数: " + Test05.getCount());

23.     }

24. }

25.

**运行结果：**

****