static-答案

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。

答案及解析：

static是Java中的一个关键字，用于表示一个类、变量或方法是静态的。静态成员属于类本身，而不是类的实例对象，因此可以通过类名直接访问，不需要创建类的实例。

以下是static关键字的三种常见用法：

1. 静态变量：使用static关键字声明的变量是静态变量，也称为类变量。静态变量属于类本身，所有类的实例共享同一个静态变量，可以通过类名直接访问。静态变量通常用于存储所有实例共享的常量或配置信息。
2. 静态方法：使用static关键字声明的方法是静态方法，也称为类方法。静态方法不依赖于任何实例对象，只能访问静态变量或调用其他静态方法。静态方法通常用于提供公共的工具方法或实现全局访问点。
3. 静态代码块：使用static关键字声明的代码块是静态代码块，用于初始化静态变量或执行其他一次性的初始化操作。静态代码块在类加载时执行，只执行一次。

其它使用static关键字的情况包括静态内部类、静态导入等，但上述三种用法是最常见的。

2.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：
2. 生命周期：
3. 出现顺序：
4. 调用方式：
5. 初始化时机：
6. 内存占用：
7. 共享：

答案及解析：

Java中的静态成员变量和非静态成员变量有以下几个区别：

1. 存储位置：静态成员变量存储在方法区中，而非静态成员变量存储在堆内存中。
2. 生命周期：静态成员变量的生命周期与类的生命周期相同，而非静态成员变量的生命周期与对象的生命周期相同。
3. 出现顺序：静态成员变量优先于非静态成员变量在内存中出现
4. 调用方式：可以通过类名直接调用静态成员变量，也可以通过对象引用调用（不推荐），而非静态成员变量必须通过对象引用调用。
5. 初始化时机：静态成员变量在类加载时就会被初始化，而非静态成员变量在对象创建时才会被初始化。
6. 内存占用：静态成员变量只会被保存一份，而非静态成员变量每次创建对象都会占用一份内存空间。
7. 共享：静态成员变量可以被类的所有实例共享，而非静态成员变量只属于该实例。

3.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？

答案及解析：

static方法即静态方法，静态方法不需要实例化即可直接通过类名访问，通常用来提供类级别的功能或工具方法。

非static方法只能通过该类实例调用。静态方法可以用类名调用（在同一个类中的静态方法可以省略类名直接调用），也可以用对象调用。

如果当某个方法被该类任何一个对象调用的结果都相同时，就可以把该方法定义为静态方法。这种情况通常方法内部没有访问类中的实例变量和实例方法。

4.代码块

题目要求：

请简述匿名代码块和静态代码块的作用。

答案及解析：

匿名代码块：没有名字和修饰符的代码块，也叫构造块。在每次进行实例化时会被依次调用执行一次，执行顺序在构造器之前。

静态代码块：使用static关键字声明的代码块。会在类被加载时固定执行一次，通常用于做某些固定的准备工作。

5.继承static

题目要求：

请回答在Java中，能否继承static成员变量或方法?

答案及解析：

在Java中，可以继承静态成员变量和方法，但是继承后的子类不能重写静态方法。

继承是子类获取父类属性和方法的过程，而静态成员变量和方法属于类本身，所以子类可以通过继承获取父类中的静态成员变量和方法。子类也可以通过类名直接访问父类中的静态成员变量和方法。

但是，当子类中定义了与父类中的静态方法相同的方法名和参数列表时，子类中的方法会隐藏父类中的静态方法，而不是覆盖它。此时，如果使用子类调用该方法，将会调用子类中定义的方法而不是父类中的静态方法。

因此，在Java中继承静态成员变量和方法是合法的，但是需要注意子类中不能重写父类的静态方法，而是可以在子类中定义与父类相同方法名的方法，但是这不会覆盖父类中的静态方法。

```
class Parent {
    static int a = 1;
    static void test1(){
        System.out.println("Parent test1");
    }

    void test2(){
        System.out.println("Parent test1");
    }
}

class Child extends Parent {
    static int a = 3;
    static void test1(){
        System.out.println("Child test1");
    }
    void test2(){
        System.out.println("Child test2");
```

```
    }
}

class Test{
    public static void main(String[] args) {
        Parent parent = new Child();
        parent.test1(); //Parent test1
        parent.test2(); //Child test2
    }
}
```

6.程序分析

题目要求：

请阅读并分析以下三个案例中，程序启动运行的结果。

案例一：

```
class Test {
    public Test() {
        System.out.println("构造器");
    }

    public void info() {
        System.out.println("info");
    }

    static {
        System.out.println("test static 1");
    }

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

```
17
18      {
19          System.out.println("代码块");
20      }
21
22      static {
23          System.out.println("test static 2");
24      }
25  }
```

答案及解析：

```
1  test static 1
2  test static 2
3  代码块
4  构造器
5  info
```

程序执行顺序说明：

- 当程序启动时，首先Test类被类加载器加载。加载后立即执行静态代码块中的语句。如有多个静态代码块，则按照其定义的先后顺序依次执行。所以首先输出“test static 1”、“test static 2”字样。
- 接下来JVM调用main()方法，执行其中的语句。main()函数中是一条连续调用语句，首先使用new关键字创建一个Test类实例，然后令该实例调用类中的info()方法。
- 当Test类实例被创建时，按照执行顺序，先执行所有的匿名代码块，最后调用构造函数。所以接下来输出“代码块”、“构造器”字样。
- 对象创建完毕后，调用info()方法，输出“info”字样。

案例二：

```
1  public class Test06 {
2      public static void main(String[] args) {
```

```
        Child child = new Child();
    }
}

class Parent {
    static {
        System.out.println("静态代码块Parent");
    }

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

    public Parent() {
        System.out.println("构造方法Parent");
    }
}

class Child extends Parent {
    static {
        System.out.println("静态代码块Child");
    }

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

    public Child() {
        System.out.println("构造方法Child");
    }
}
```

答案及解析：

```
静态代码块Parent
静态代码块Child
构造代码块Parent
构造方法Parent
构造代码块Child
构造方法Child
```

程序执行顺序说明：

- Test类作为程序的入口，先执行其main()方法。main()方法中创建了一个Child类型对象，赋值给一个Child类型引用变量。Java中创建某个子类对象时，需要先创建其父类的对象，所以Parent类需要先被进行加载。
- Parent类被加载之后，执行Parent类中的静态代码块，输出“静态代码块Parent”字样。
- 注意，接下来并未直接进入Parent实例的构造流程，而是先将子类Child进行加载，输出“静态代码块Child”字样。

  > 为什么不完成Parent实例的构造之后，再加载Child子类？
  >
  > - 从一般性的执行原则来讲，静态性质相关步骤的执行优先级总是高于非静态性质步骤。
  > - 从实际运行角度来说，在Parent类完成构造的过程中，有可能需要调用被子类重写过后的方法。如果子类此时未被加载，则方法无法调用。

- 接下来进入父类实例Parent的创建流程。先调用其匿名代码块，输出“构造代码块Parent”字样，再调用其构造器，输出“构造方法Parent”字样。
- 父类实例完成创建后，开始创建子类实例。顺序同上，先调用其匿名代码块，输出“构造代码块Child”字样，再调用其构造器，输出“构造方法Child”字样。

案例三：

```java
class B {
    public static B b = new B();
    public static B b2 = new B();

    {
        System.out.println("构造块");
    }

    static {
        System.out.println("静态块");
    }
}
public class Test06 {
    public static void main(String[] args) {
        B b = new B();
    }
}
```

答案及解析：

```
构造块
构造块
静态块
构造块
```

此题的难点在于静态代码块的执行时机问题。程序执行顺序说明：

- 程序启动，首先B类完成类加载。
- 当类加载结束后，所有静态成员按照定义顺序进行加载，所以首先被加载的是t1和t2两个静态变量（注意，并非直接无条件调用静态代码块）。
- t1变量都在声明时进行了显式赋值，所以先执行了等号右边的new B()步骤，接下来进入一次B对象构造流程，先调用匿名代码块，输出“构造块”字样。

- 当这个B对象构造完毕后，其引用被赋值给t1变量，至此t1变量初始化完毕，接下来初始化t2变量，与t1同理，再次调用B类匿名代码块，输出“构造块”字样。
- 接下来，t1和t2两个静态变量全部加载完毕，按照定义顺序，执行B类中静态代码块的内容，输出“静态块”字样。
- 最后，所有静态成员加载完毕，调用main()方法开始执行程序，再次创建B类实例，所以再次输出“构造块字样”。

7.单例模式

题目要求：

请解释什么是单例模式，并提供一个使用静态变量实现单例模式的例子。

答案及解析：

单例模式是一种设计模式，它保证一个类仅有一个实例，并提供一个全局访问方式。这意味着该类只有一个对象可以被创建并且在程序的所有部分都可以访问到这个对象。

```
public class Test07_Singleton {
    public static void main(String[] args) {
        Student s = Student.getInstance();
        Teacher t = Teacher.getInstance();
    }
}
//饿汉式：
class Student {
    private static final Student instance = new Student();

    private Student() {
        // 私有构造函数，避免类被实例化
    }

```

```
    public static Student getInstance() {
        return instance;
    }
}
//懒汉式：
class Teacher {
    private static Teacher instance;

    private Teacher(){
    }

    public static Teacher getInstance(){
        if (instance == null){
            instance = new Teacher();
        }
        return instance;
    }
}
```

8.统计次数

题目要求：

编写一个Java类，实现如下功能：

- 该类能够自动记录被实例化的次数（即创建过该类多少对象）
- 能够随时通过调用某个方法，获取到这个数值

答案及解析：

```
class ObjectCounter{
    private static int count;

```

```
4       public ObjectCounter(){
5           count++;
6       }
7       public static int getObjectCount(){
8           return count;
9       }
10  }
11  public class Test08_ObjectCounter {
12      public static void main(String[] args) {
13          new ObjectCounter();
14          new ObjectCounter();
15          System.out.println("实例化次
    数："+ObjectCounter.getObjectCount());
16      }
17  }
```