

# SA第六次作业

## 1、用Java书写具有双向加锁功能的孤子模式（volatile , synchronized）。

### 源码

Main.java

```
package org.example;

public class Main {
    public static void main(String[] args) {
        System.out.println("Start.");
        Singleton obj1 = Singleton.getInstance();
        Singleton obj2 = Singleton.getInstance();
        if (obj1 == obj2) {
            System.out.println("obj1和obj2是同一对象实例");
        } else {
            System.out.println("obj1和obj2并非同一对象实例");
        }
        System.out.println("End.");
    }
}
```

Singleton.java

使用了 volatile 关键字来确保 instance 在多线程环境中的可见性。

使用两次if判空进行双重检查锁定，确保了只有在第一次创建实例时才会进行同步。

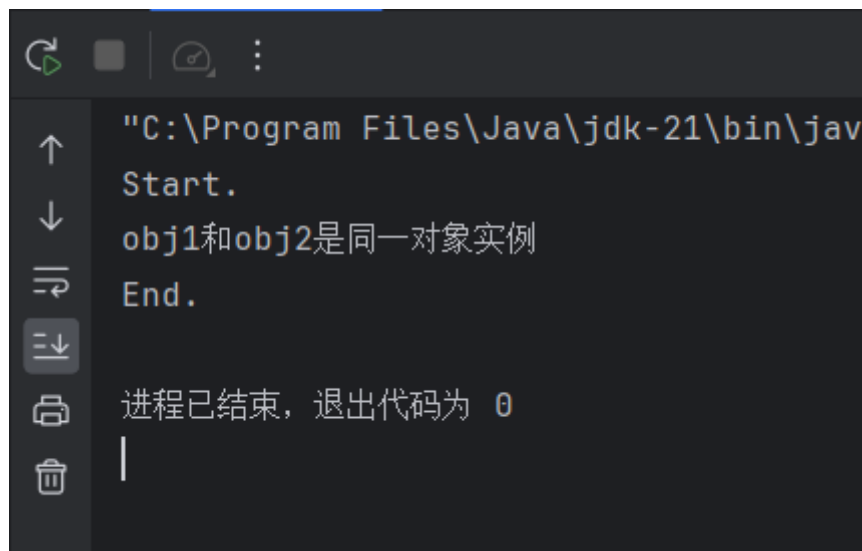
使用 synchronized 关键字获得该类对应的锁。如果该对象的锁没有被其他线程持有，那么该线程会获得锁并执行同步代码。如果该对象的锁已经被其他线程持有，那么该线程就会被阻塞，直到锁被释放。

```
package org.example;

public class Singleton {
    private static volatile Singleton instance;
    private Singleton() {}
    public static Singleton getInstance() {
        if (instance == null) {
            synchronized (Singleton.class) {
                if (instance == null) {
                    instance = new Singleton();
                }
            }
        }
    }
}
```

```
        return instance;
    }
}
```

## 运行截图



## 2、用Java书写具有可变用例数目的孤子模式。

### 源码

Main.java

```
package org.example;

public class Main {
    public static void main(String[] args) {
        System.out.println("Start.");
        VariableSingleton obj1 = VariableSingleton.getInstance("11");
        VariableSingleton obj2 = VariableSingleton.getInstance("11");
        if (obj1 == obj2) {
            System.out.println("obj1和obj2是同一对象实例");
        } else {
            System.out.println("obj1和obj2并非同一对象实例");
        }
        VariableSingleton obj3 = VariableSingleton.getInstance("11");
        VariableSingleton obj4 = VariableSingleton.getInstance("12");
        if (obj3 == obj4) {
            System.out.println("obj3和obj4是同一对象实例");
        } else {
            System.out.println("obj3和obj4并非同一对象实例");
        }
        System.out.println("End.");
    }
}
```

VariableSingleton.java

使用Map<String,VariableSingleton>来保存多个用例，每个用例用唯一的String作为key来指向。

因此获得用例时同样String获得的是同一个用例；不同String获得的是不同用例。

```
package org.example;

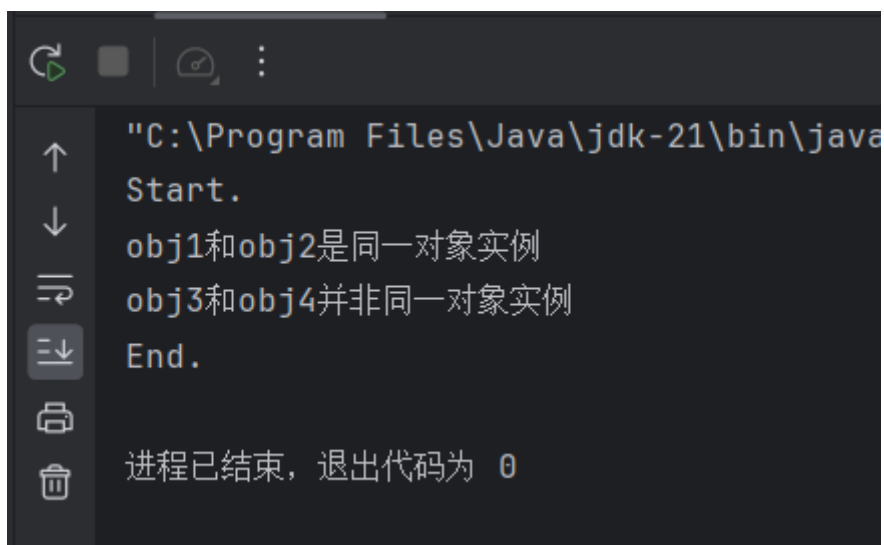
import java.util.HashMap;
import java.util.Map;

public class VariableSingleton {
    private static Map<String, VariableSingleton> instances = new HashMap<>();
    private String name;

    private VariableSingleton(String name) {
        this.name = name;
    }

    public static VariableSingleton getInstance(String name) {
        if (!instances.containsKey(name)) {
            synchronized (VariableSingleton.class) {
                if (!instances.containsKey(name)) {
                    instances.put(name, new VariableSingleton(name));
                }
            }
        }
        return instances.get(name);
    }
}
```

## 运行截图



```
"C:\Program Files\Java\jdk-21\bin\java
Start.
obj1和obj2是同一对象实例
obj3和obj4并非同一对象实例
End.
进程已结束，退出代码为 0
```