华东师范大学数据科学与工程学院实验报告

课程名称：计算机网络与编程	**年级：**21 级	**上机实践成绩：**
指导教师：张召	**姓名：**杨茜雅	**学号：**10215501435
上机实践名称：Lab04		**上机实践日期：**2023.3.24
上机实践编号：	**组号：**	**上机实践时间：**3.24-3.31

一、 实验目的

1、熟悉Java多线程编程

2、熟悉并掌握线程创建和线程间交互

二、 实验任务

1、熟悉创建线程方法，继承线程类，实现Runnable接口 (匿名类不涉及)

2、使用`sleep()` 、`join()` 、`yield()` 方法对线程进行控制

3、初步接触多线程编程

三、 使用环境

IntelliJ IDEA

JDK 版本: Java 19

四、 实验过程

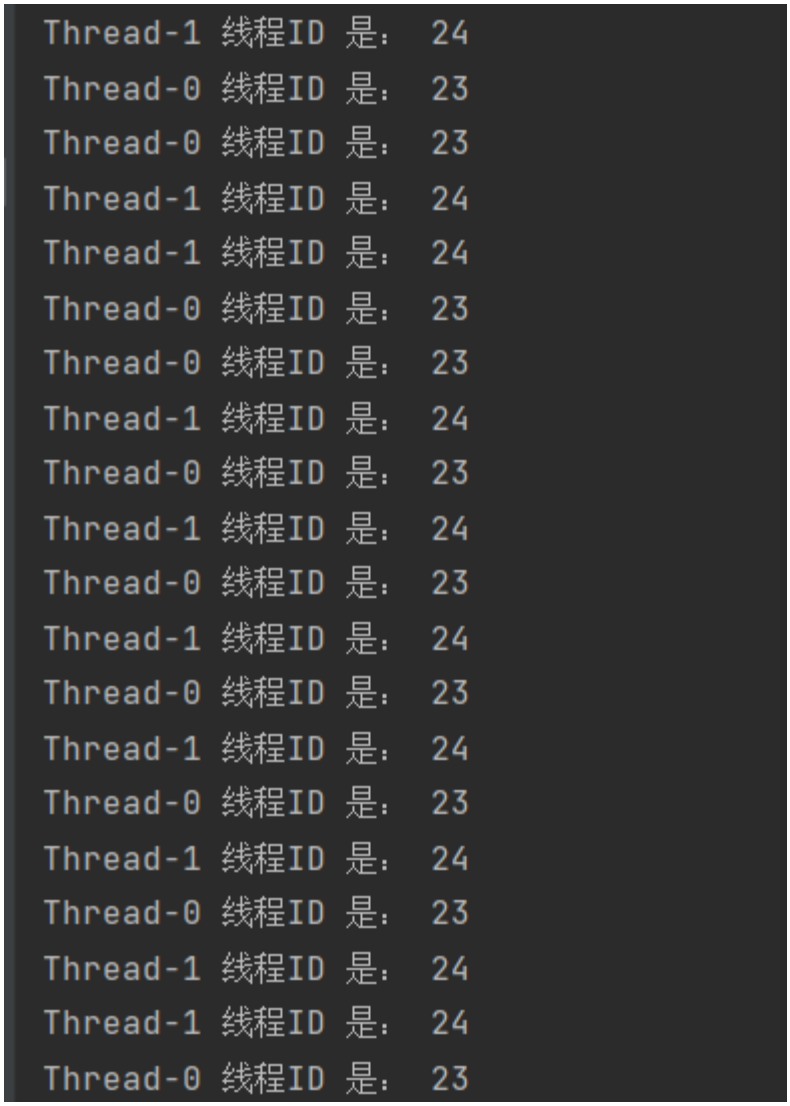
Task 1：使用继承Thread类的方式，编写ThreadTest类，改写run() 方法，方法逻辑为每隔1秒打印Thread.currentThread().getId() ，循环10次。实例化两个ThreadTest对象，并调用start() 方法，代码及运行结果附在实验报告中。

代码：

```java
package Muti;
public class ThreadTest extends Thread {
    public void run() {
        int i = 0;
        while (i < 10) {
            System.out.println(Thread.currentThread().getName() + " 线程ID 是： " + Thread.currentThread().getId());
            i++;
            try {
                Thread.sleep( millis: 1000);
            } catch (InterruptedException e) {
                throw new RuntimeException(e);
            }

        }
    }
    public static void main (String[] args){
        ThreadTest mthread1 = new ThreadTest();
        ThreadTest mthread2 = new ThreadTest();
        mthread1.start();
        mthread2.start();
    }
}
```

运行结果：



Task 2:

给出以下BattleObject 、Battle 、TestBattle 类，请改写Battle 类，实现Runnable接口， run()方法逻辑为让bo1 调用attackHero(bo2) ，直到bo2 的状态为isDestoryed() ，请完成代码后使用TestBattle 进行测试，将实现代码段及运行结果附在实验报告中。

代码：

```java
package Muti;

public class Battle implements Runnable {
    private BattleObject bo1;
    private BattleObject bo2;

    public Battle(BattleObject bo1, BattleObject bo2) {
        this.bo1 = bo1;
        this.bo2 = bo2;
    }

    @Override
    public void run() {
        while (!bo2.isDestoryed()) {
            bo1.attackHero(bo2);
        }
    }
}
```

```java
package Muti;

public class TestBattle {
    public static void main(String[] args) {
        BattleObject bo1 = new BattleObject();
        bo1.name = "Object1";
        bo1.hp = 600;
        bo1.attack = 50;
        BattleObject bo2 = new BattleObject();
        bo2.name = "Object2";
        bo2.hp = 500;
        bo2.attack = 40;
        Battle battle = new Battle(bo1, bo2);
        Thread thread = new Thread(battle);
        thread.start();
        try {
            thread.join();
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }
    }
}
```

运行结果：

```
C:\Users\86138\.jdks\openjdk-19.0.2\bin\java.exe "-javaagent:E:\Prog
Object1 正在攻击 Object2, Object2 的耐久还剩 450.00
Object1 正在攻击 Object2, Object2 的耐久还剩 400.00
Object1 正在攻击 Object2, Object2 的耐久还剩 350.00
Object1 正在攻击 Object2, Object2 的耐久还剩 300.00
Object1 正在攻击 Object2, Object2 的耐久还剩 250.00
Object1 正在攻击 Object2, Object2 的耐久还剩 200.00
Object1 正在攻击 Object2, Object2 的耐久还剩 150.00
Object1 正在攻击 Object2, Object2 的耐久还剩 100.00
Object1 正在攻击 Object2, Object2 的耐久还剩 50.00
Object1 正在攻击 Object2, Object2 的耐久还剩 0.00
Object2被消灭。

Process finished with exit code 0
```

Task 3：完善代码，用join方法实现正常的逻辑，并将关键代码和结果写到实验报告中。

代码：

```
package Muti;
public class ThreadTest03 implements Runnable{
    @Override
    public void run(){
        System.out.println(Thread.currentThread().getName());
    }
    public static void main(String[] args) throws InterruptedException {
        ThreadTest03 join = new ThreadTest03();
        Thread thread1 = new Thread(join, name: "上课铃响");
        Thread thread2 = new Thread(join, name: "老师上课");
        Thread thread3 = new Thread(join, name: "下课铃响");
        Thread thread4 = new Thread(join, name: "老师下课");
        thread1.start();
        thread1.join();
        thread2.start();
        thread2.join();
        thread3.start();
        thread3.join();
        thread4.start();
        thread4.join();
    }
}
```

运行结果：

```
C:\Users\86138\.jdks\openjdk-19.0.2\bin\java.exe "-javaagent:E:\Program Files\JetBrains\Inte
上课铃响
老师上课
下课铃响
老师下课

Process finished with exit code 0
```

Task 4：完善代码，将助教线程设置为守护线程，当同学们下课时，助教线程自动结束。并将关键代码和结果写到实验报告中。

代码：

```
package Muti;
public class ThreadTest04 implements Runnable {
    @Override
    public void run() {
        int worktime = 0;
        while (true) {
            System.out.println("助教在教室的第" + worktime + "秒");
            try {
                Thread.currentThread().sleep( millis: 1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            worktime++;
        }
    }

    public static void main(String[] args) throws InterruptedException {
        ThreadTest04 inClassroom = new ThreadTest04();
        Thread thread = new Thread(inClassroom,  name: "助教");
        thread.setDaemon(true);
        thread.start();
        for (int i = 0; i < 10; i++) {
            thread.sleep( millis: 1000);
            System.out.println("同学们正在上课");
            if (i == 9) {
                System.out.println("同学们下课了");
            }
        }
    }
}
```

运行结果：

```
C:\Users\86138\.jdks\openjdk-19.0.2\bin\java.exe "-javaagent:E:\Program
助教在教室的第0秒
同学们正在上课
助教在教室的第1秒
同学们正在上课
助教在教室的第2秒
同学们正在上课
助教在教室的第3秒
同学们正在上课
助教在教室的第4秒
同学们正在上课
助教在教室的第5秒
同学们正在上课
助教在教室的第6秒
同学们正在上课
助教在教室的第7秒
同学们正在上课
助教在教室的第8秒
同学们正在上课
助教在教室的第9秒
同学们正在上课
同学们下课了
```

Task 5：给出TestVolatile 类，测试main 方法，观察运行结果，并尝试分析结果。

代码：

```java
package Muti;

// if variable is not volatile, this example may not be terminated
// but this behaviour may differ on some machines
class TestVolatile extends Thread{
    //volatile
// volatile boolean sayHello = true;
    boolean sayHello = true;
    public void run() {
        long count=0;
        while (sayHello) {
            count++;
        }
        System.out.println("Thread terminated." + count);
    }
    public static void main(String[] args) throws InterruptedException {
        TestVolatile t = new TestVolatile();
        t.start();
        Thread.sleep( millis: 1000);
        System.out.println("after main func sleeping...");
        t.sayHello = false;
        t.join();
        System.out.println("sayHello set to " + t.sayHello);
    }
}
```

运行截图：

```
C:\Users\86138\.jdks\openjdk-19.0.2\bin\java.exe "-javaagent:E:\Program Files\JetBrain
after main func sleeping...
```

分析结果：

Volatile是java虚拟机提供的轻量级的同步机制，有三个特性：

（1） 保证可见性 （2）不保证原子性 （3）禁止指令重排

理解可见性：

可见性是指当多个线程访问同一个变量时，一个线程修改了这个变量的值，其他线程能够立即看得到修改的值。

JMM（Java 内存模型，Java Memory Model）本身是一种抽象的概念，并不真实存在。它描述的是一组规则或规范，通过这组规范，定了程序中各个变量的访问方法。JMM 关于同步的规定：

1）线程解锁前，必须把共享变量的值刷新回主内存；

2）线程加锁前，必须读取主内存的最新值到自己的工作内存；

3）加锁解锁是同一把锁；

由于 JVM 运行程序的实体是线程，创建每个线程时，JMM 会为其创建一个工作内存（有些地方称为栈空间），工作内存是每个线程的私有数据区域。

而在这个程序中，代码一直卡在 System.out.println(“after main func sleeping….”)阶段，主要原因就是因为 volatile 的特性（1），变量 sayHello 不使用 volatile 就没有可见性，主线程一直在循环中，sayHello 的值一直没办法被更新为 false。

接下来我们修改代码看一下情况是够符合我们的猜想：

加了volatile之后的代码：

```
package Muti;

// if variable is not volatile, this example may not be terminated
// but this behaviour may differ on some machines
class TestVolatile extends Thread{
    //volatile
 volatile boolean sayHello = true;
//    boolean sayHello = true;
    public void run() {
        long count=0;
        while (sayHello) {
            count++;
        }
        System.out.println("Thread terminated." + count);
    }
    public static void main(String[] args) throws InterruptedException {
        TestVolatile t = new TestVolatile();
        t.start();
        Thread.sleep( millis: 1000);
        System.out.println("after main func sleeping...");
        t.sayHello = false;
        t.join();
        System.out.println("sayHello set to " + t.sayHello);
    }
}
```

运行结果：

```
C:\Users\86138\.jdks\openjdk-19.0.2\bin\java.exe "-javaagent:E:\Program Files
after main func sleeping...
Thread terminated.1099740111
sayHello set to false

Process finished with exit code 0
```

从这里可以看出，sayHello加了volatile之后主线程成功结束，因为JMM主动通知主线程更新sayHello的值，sayHello现在是false了。

Task 6:

给出PlusMinus 、TestPlus 、Plus 三个类，描述TestPlus 的main 方法的运行逻辑，并多次运行，观察输出结果，并尝试分析结果。

代码：

```
public class PlusMinus {
    public int num;
    public void plusOne(){
        num = num + 1;
    }
    public void minusOne(){
        num = num - 1;
    }
    public int printNum(){
        return num;
    }
}
public class TestPlus {
    public static void main(String[] args) throws InterruptedException {
        PlusMinus plusMinus = new PlusMinus();
        plusMinus.num = 0;

        int threadNum = 10;
        Thread[] plusThreads = new Thread[threadNum];

        for(int i=0;i<threadNum;i++){
            plusThreads[i] = new Plus(plusMinus);
        }

        for(int i=0;i<threadNum;i++){
            plusThreads[i].start();
        }

        for(int i=0;i<threadNum;i++){
            plusThreads[i].join();
        }

        System.out.println(plusMinus.printNum());
    }
}

class Plus extends Thread{
    Plus(PlusMinus pm){
        this.plusMinus = pm;
    }
    @Override
    public void run(){
```

```
        for(int i=0;i<10000;i++){
            plusMinus.plusOne();
        }
    }
    PlusMinus plusMinus;
}
```

运行结果（多次运行）：

```
C:\Users\86138\.jdks\openjdk-19.0.2\bin\java.exe "-javaagent
52451

Process finished with exit code 0
C:\Users\86138\.jdks\openjdk-19.0.2\bin\java.exe "-javaagent:E:\Program Files\JetBrains\Intel
42796

Process finished with exit code 0
C:\Users\86138\.jdks\openjdk-19.0.2\bin\java.exe "-javaagent:E:\Program Files\JetBra
38820

Process finished with exit code 0
C:\Users\86138\.jdks\openjdk-19.0.2\bin\java.exe "-javaagent:E:\Program Files\Je
47819

Process finished with exit code 0
```

结果分析：

程序new了十个线程，然后依次开启十个线程，主线程依次等待新线程执行完毕。最后输出plusMinus的返回值。为什么每次运行的结果都不一样，可能是因为程序里面的变量都是普通的共享变量而不是使用volatile来保证可见性，所以当变量被修改之后，什么时候写入主存是不确定的，当其他线程去读取的时候，这个内存中可能还是原来的旧值，因此无法保证可见性，所以每次输出的结果都有可能不一样。并且PlusMinus中定义的加和减都不是原子性操作，也没有用synchronized和Lock来保证原子性，所以不能保证任意时刻只有一个线程执行该代码块，所以存在原子性问题，因此每次输出的结果都不一致。

五、总结

通过本次实验初步尝试并且熟悉了创建线程的方法，继承线程类，实现Runnable接口，了解使用sleep() 、join() 、yield() 方法对线程进行控制的方法，初步接触多线程编程，了解多线程编程的内核，为之后的java实践打下基础。