厦門大學

信息学院软件工程系

《JAVA 程序设计》实验报告

实验 14

姓名：黄勖

学号：**22920212204392**

学院：信息学院

专业：软件工程

完成时间：**2023.5.29**

一、实验目的及要求

- 熟悉多线程

二、实验题目及实现过程

实验环境：Windows 10 21H2、jdk17、javafx scene builder、utf-8 编码

题目一

（一） 实验题目

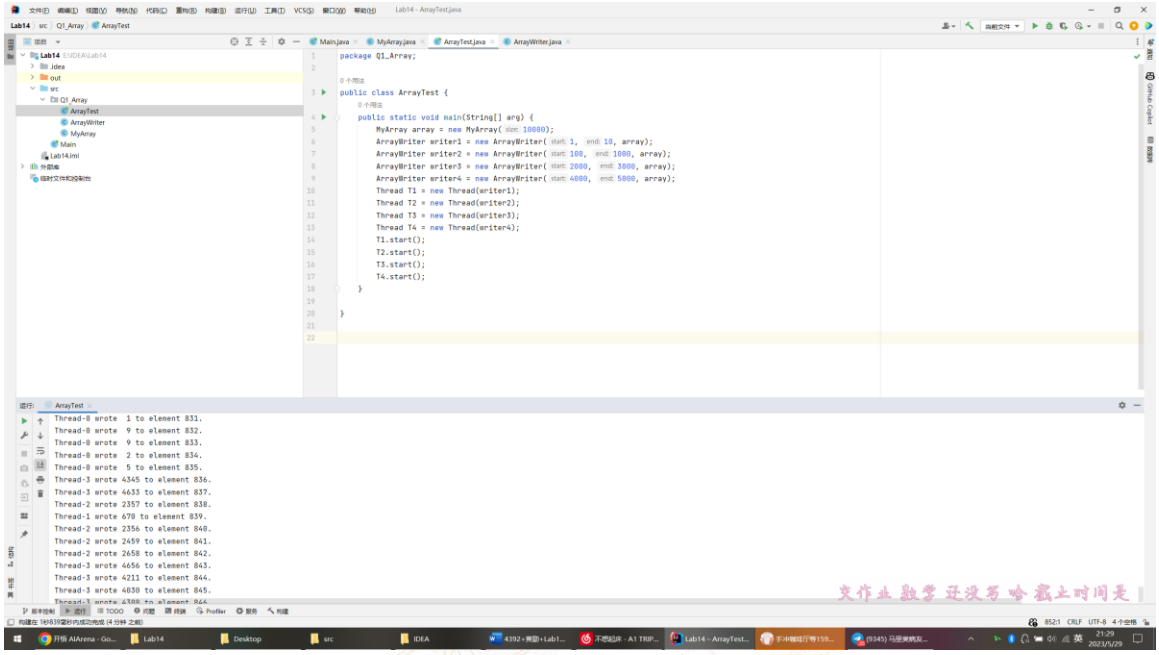
◆ 编程一个程序，4 个线程同时向一个 10000 位的数组中分别写入 1-10,100-1000,2000-3000,4000-5000 的数。

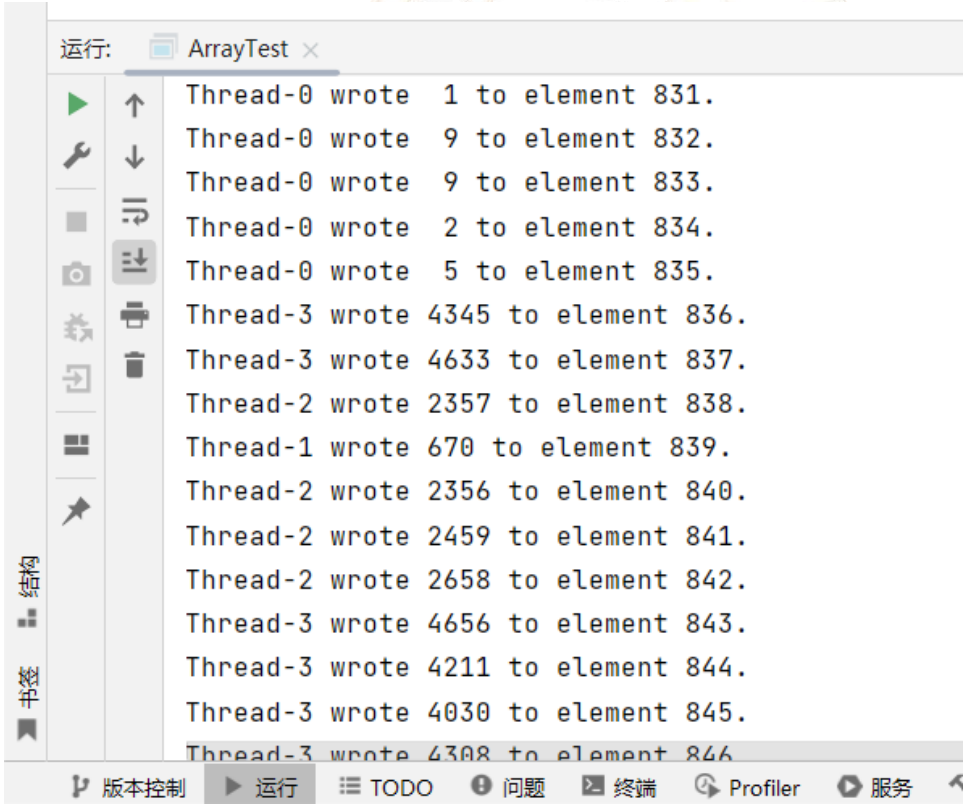
（二） 实现过程

思路：设计了三个类。ArrayWriter 实现了 Runnable 接口，重写 run 方法，实现每个线程向数组写入数字的功能，其构造函数需要初始化输入数字的范围以及数组。Myarray 类是负责实现数组相关的功能，构造函数创建数组，add 函数由 synchronized 修饰实现线程同步，其他线程调用这个方法就会被阻塞，排队等待 CPU 资源。ArrayTest 类用创建 MyArray 对象，再用 ArrayWriter 的对象创建四个线程，实现四个线程同时向数组写入数字。

（三） 过程截图

最终结果（全屏截图）


package Q1_Array;
public class ArrayTest {
public static void main(String[] arg) {
MyArray array = new MyArray(size: 10000);
ArrayWriter writer1 = new ArrayWriter(start: 1, end: 10, array);
ArrayWriter writer2 = new ArrayWriter(start: 100, end: 1000, array);
ArrayWriter writer3 = new ArrayWriter(start: 2000, end: 3000, array);
ArrayWriter writer4 = new ArrayWriter(start: 4000, end: 5000, array);
Thread T1 = new Thread(writer1);
Thread T2 = new Thread(writer2);
Thread T3 = new Thread(writer3);
Thread T4 = new Thread(writer4);
T1.start();
T2.start();
T3.start();
T4.start();
}
}
Thread-0 wrote 1 to element 831.
Thread-0 wrote 9 to element 832.
Thread-0 wrote 9 to element 833.
Thread-0 wrote 2 to element 834.
Thread-0 wrote 5 to element 835.
Thread-3 wrote 4345 to element 836.
Thread-3 wrote 4633 to element 837.
Thread-2 wrote 2357 to element 838.
Thread-1 wrote 670 to element 839.
Thread-2 wrote 2356 to element 840.
Thread-2 wrote 2459 to element 841.
Thread-2 wrote 2658 to element 842.
Thread-3 wrote 4656 to element 843.
Thread-3 wrote 4211 to element 844.
Thread-3 wrote 4030 to element 845.


运行:
ArrayTest
Thread-0 wrote 1 to element 831.
Thread-0 wrote 9 to element 832.
Thread-0 wrote 9 to element 833.
Thread-0 wrote 2 to element 834.
Thread-0 wrote 5 to element 835.
Thread-3 wrote 4345 to element 836.
Thread-3 wrote 4633 to element 837.
Thread-2 wrote 2357 to element 838.
Thread-1 wrote 670 to element 839.
Thread-2 wrote 2356 to element 840.
Thread-2 wrote 2459 to element 841.
Thread-2 wrote 2658 to element 842.
Thread-3 wrote 4656 to element 843.
Thread-3 wrote 4211 to element 844.
Thread-3 wrote 4030 to element 845.
版本控制
运行
TODO
问题
终端
Profiler
服务

部分代码展示

Main.java MyArray.java ArrayTest.java ArrayWriter.java

```java
package Q1_Array;

0 个用法
public class ArrayTest {
    0 个用法
    public static void main(String[] arg) {
        MyArray array = new MyArray( size: 10000);
        ArrayWriter writer1 = new ArrayWriter( start: 1,  end: 10, array);
        ArrayWriter writer2 = new ArrayWriter( start: 100,  end: 1000, array);
        ArrayWriter writer3 = new ArrayWriter( start: 2000,  end: 3000, array);
        ArrayWriter writer4 = new ArrayWriter( start: 4000,  end: 5000, array);
        Thread T1 = new Thread(writer1);
        Thread T2 = new Thread(writer2);
        Thread T3 = new Thread(writer3);
        Thread T4 = new Thread(writer4);
        T1.start();
        T2.start();
        T3.start();
        T4.start();
    }

}
```

Main.java MyArray.java ArrayTest.java ArrayWriter.java

```java
public class ArrayWriter implements Runnable {
    2 个用法
    private MyArray array;
    3 个用法
    private int startValue;
    2 个用法
    private int endValue;
    1 个用法
    Random r = new Random();

    4 个用法
    ArrayWriter(int start, int end, MyArray array) {
        this.startValue = start;
        this.endValue = end;
        this.array = array;
    }

    @Override
    public void run() {
        for (int j = 0; j <= 2500; j++) {
            int i = r.nextInt( bound: endValue - startValue) + startValue;
            array.add(i);
        }

    }

}
```

Main.java MyArray.java ArrayTest.java ArrayWriter.java

```java
public class MyArray {
    2 个用法
    private int[] array; // the shared integer array
    2 个用法
    private int writeIndex = 0;
    1 个用法
    private static final SecureRandom generator = new SecureRandom();

    1 个用法
    public MyArray(int size) { array = new int[size]; }

    1 个用法
    public synchronized void add(int value) {
        int position = writeIndex;
        try {
            Thread.sleep(generator.nextInt( bound: 500));
        } catch (InterruptedException ex) {
            Thread.currentThread().interrupt(); // re-interrupt the thread
        }
        array[position] = value;
        System.out.printf("%s wrote %2d to element %d.%n", Thread.currentThread().getName(), value, position);
        ++writeIndex;
    }
}
```

题目二

（一） 实验题目

- 用两种方法编程一个程序：一个线程向一个 4 位循环缓冲区中循环写入 1-10 的随机数，另一个线程将循环缓冲区中数据取出打印出来。

（二） 实现过程

思路：

第一种方法： 总共设计了四个类

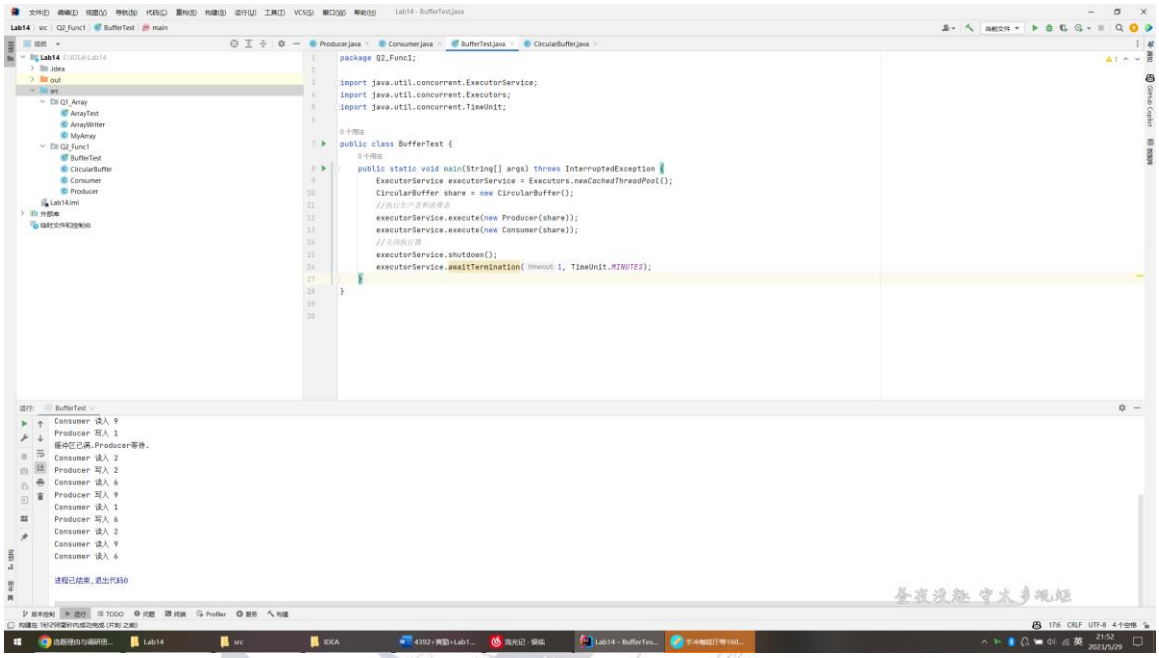
CircularBuffer 类实现循环缓冲区，该类初始化了一个长度为 4 的数组，synchronized 修饰的 blockingPut 函数负责向缓冲区写入随机数，然后重置写入的下标为原下标加一再除以数组长度的余数，最后输出写入结果，唤醒所有在该对象上等待的线程。blockingGet 函数负责从缓冲区获取随机数并打印出来，同样重置读下标为原下标加一再除以数组长度的余数，输出读取结果，唤醒唤醒所有在该对象上等待的线程。

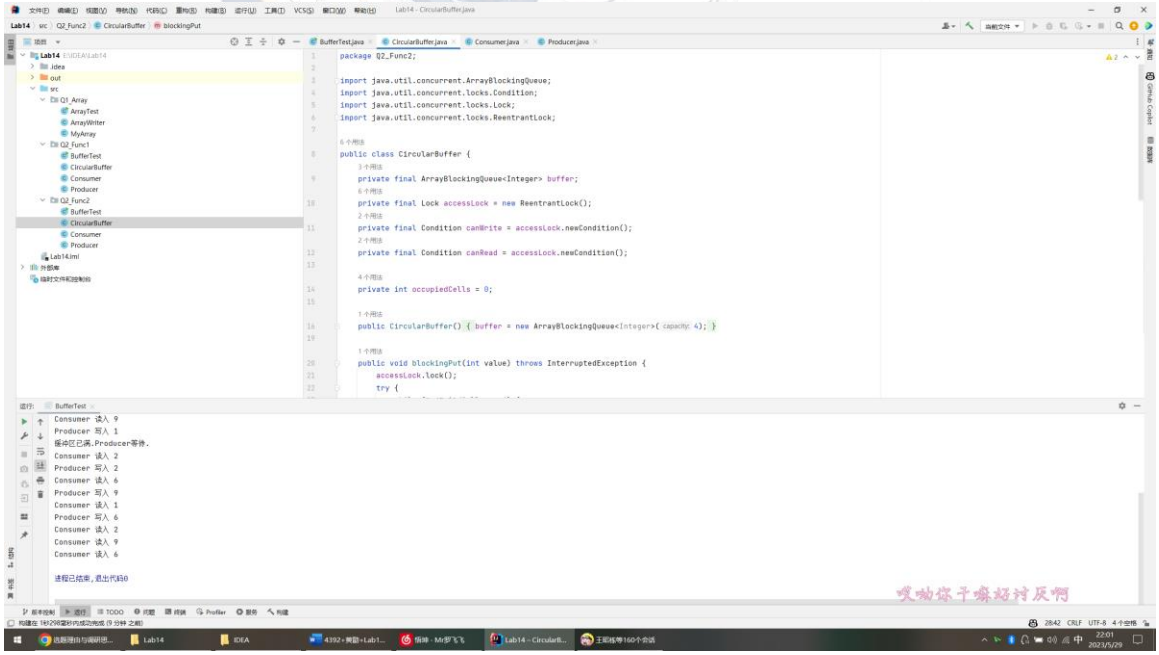
Producer 负责输入，Consumer 负责读出，两个类实现 Runnable 接口，重写 run 方法，分别调用 CircularBuff 的 blockingPut 方法和 blockingGet 方法。BufferTest 类创建线程池、CircularBuffer 的对象，Producer 对象、Consumer 对象，运行两个线程实现读取和写循环缓冲区。

第二种方法：引入了循环数组 ArrayBlokingQueue，代码量减少，更简单；使用了互斥锁 ReentrantLock，在写入和读取之前上锁，操作完成后解锁，从而达到线程同步的效果。

（三） 过程截图

最终结果（全屏截图）





```
运行: BufferTest
Consumer 读入 9
Producer 写入 1
缓冲区已满.Producer等待.
Consumer 读入 2
Producer 写入 2
Consumer 读入 6
Producer 写入 9
Consumer 读入 1
Producer 写入 6
Consumer 读入 2
Consumer 读入 9
Consumer 读入 6
```

部分代码展示

BufferTest.java × CircularBuffer.java × Consumer.java × Producer.java ×

```java
package Q2_Func2;

import java.security.SecureRandom;

1 个用法
public class Producer implements Runnable {
    2 个用法
    private static final SecureRandom generator = new SecureRandom();
    2 个用法
    private final CircularBuffer share; //共享的缓冲区

    // constructor
    1 个用法
    public Producer(CircularBuffer share) { this.share = share; }

    @Override
    public void run() {
        //写入10个值
        for (int count = 1; count <= 10; count++) {
            try {
                Thread.sleep(generator.nextInt( bound: 3000)); //
                share.blockingPut( value: generator.nextInt( bound: 10) + 1);
            } catch (InterruptedException exception) {
                Thread.currentThread().interrupt();
            }
        }
    }

```

BufferTest.java × CircularBuffer.java × Consumer.java × Producer.java ×

```java
package Q2_Func2;


import java.security.SecureRandom;

1 个用法
public class Consumer implements Runnable {
    2 个用法
    private CircularBuffer share;//共享的缓冲区
    1 个用法
    private static final SecureRandom generator = new SecureRandom();

    // constructor
    1 个用法
    Consumer(CircularBuffer share) { this.share = share; }

    @Override
    public void run() {
        //读取10个值
        for (int count = 1; count <= 10; count++) {
            try {
                Thread.sleep(generator.nextInt( bound: 3000));
                share.blockingGet();
            } catch (InterruptedException exception) {
                Thread.currentThread().interrupt();
            }
        }
    }
```

BufferTest.java CircularBuffer.java Consumer.java Producer.java

```java
import java.util.concurrent.locks.ReentrantLock;

6 个用法
public class CircularBuffer {
    3 个用法
    private final ArrayBlockingQueue<Integer> buffer;
    6 个用法
    private final Lock accessLock = new ReentrantLock();
    2 个用法
    private final Condition canWrite = accessLock.newCondition();
    2 个用法
    private final Condition canRead = accessLock.newCondition();

    4 个用法
    private int occupiedCells = 0;

    1 个用法
    public CircularBuffer() { buffer = new ArrayBlockingQueue<Integer>( capacity: 4); }

    1 个用法
    public void blockingPut(int value) throws InterruptedException {
        accessLock.lock();
        try {
            while (occupiedCells == 4) {
                System.out.println("缓冲区已满.Producer等待.");
                canWrite.await();
            }
            buffer.put(value);
```

BufferTest.java CircularBuffer.java Consumer.java Producer.java

```java
package Q2_Func2;


import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;

0 个用法
public class BufferTest {
    0 个用法
    public static void main(String[] args) throws InterruptedException {
        ExecutorService executorService = Executors.newCachedThreadPool();
        CircularBuffer share = new CircularBuffer();

        //执行生产者和消费者
        executorService.execute(new Producer(share));
        executorService.execute(new Consumer(share));
        //关闭执行
        executorService.shutdown();
        executorService.awaitTermination( timeout: 1, TimeUnit.MINUTES);
    }
}
```

三、实验总结与心得记录

通过本次实验，我对 Java 使用多线程编程有了更深入的了解。我学会了如何构建一个线程等操作，同时，我也学会了对线程的相关操作，实现了多次输出。在实验的过程中我为线程编写了对应的方法，实现了需要的功能，这对我来说收获颇丰。

```
java.lang.IllegalMonitorStateException: current thread is not owner
```

直接对 Java 对象进行 wait、notifyall、notify 方法时，会出现上图所示的错误，用 synchronized 修饰该对象的相关操作后正常运行