

**Session 8****Multithreading & Concurrency****Phần I - Thực hiện trong 120 phút****1. Mục tiêu**

- Hiểu biết về đa luồng.
- Nắm vững vòng đời của thread.
- Hiểu và sử dụng các hàm thường dùng như `isAlive()`, `join()`, `wait()`, `notify()`.
- Hiểu biết về truyền thông giữa các thread.
- Hiểu biết về đồng bộ và bất đồng bộ.
- Hiểu và kiểm soát tình huống deadlock.

**2. Thực hiện**

**Bài thực hành 1:** Tạo một class tên là `PrintDemo` có hàm `printCount()` thực hiện vòng lặp từ 5->0 và in ra giá trị `i`. Tạo thread tên là `ThreadDemo` nhận vào đối tượng `PrintDemo` và trong `run()` gọi tới hàm `printCount()`. `MainClass` chứa hàm `main()` khởi tạo 2 thread trên và kích hoạt đồng thời. Chạy mã lệnh ở tình huống không `synchronized` và có để thấy sự khác biệt.

Bước 1: Viết class `PrintDemo`

```
package demo.jp2.lab05.baith1;

/**
 *
 * @authorminhvuvc
 */
public class PrintDemo {

    public void printCount() {
        try {
            for (int i = 5; i > 0; i--) {
                System.out.println("Counter --- " + i);
            }
        } catch (Exception e) {
```

```
        System.out.println("Thread interrupted.");  
    }  
}  
}
```

Bước 2: Viết class ThreadDemo

```
package demo.jp2.lab05.baith1;  
  
/**  
 *  
 * @authorminhvufc  
 */  
public class ThreadDemo extends Thread {  
  
    private Thread t;  
    private String threadName;  
    PrintDemo PD;  
  
    ThreadDemo(String name, PrintDemo pd) {  
        threadName = name;  
        PD = pd;  
    }  
  
    public void run() {  
        synchronized (PD) {  
            PD.printCount();  
        }  
        System.out.println("Thread " + threadName + " exiting.");  
    }  
  
    public void start() {
```

```
System.out.println("Starting " + threadName);
if (t == null) {
    t = new Thread(this, threadName);
    t.start();
}
}
}
```

**Chú giải:** synchronized có tác dụng **khóa** đối tượng trong lập trình đa luồng Java. Trong class TestThread viết sau đây, đối tượng PrintDemo sẽ được truyền "tham chiếu" do đó dù kích hoạt đồng thời nhưng trong tích tắc chỉ có 1 Thread được quyền truy cập. Các thread đến sau buộc phải chờ cho thread trước đó xong việc mới đến lượt.

Bước 3: Viết class TestThread

```
package demo.jp2.lab05.baith1;

/**
 *
 * @authorminhvuafc
 */
public class TestThread {

    public static void main(String args[]) {

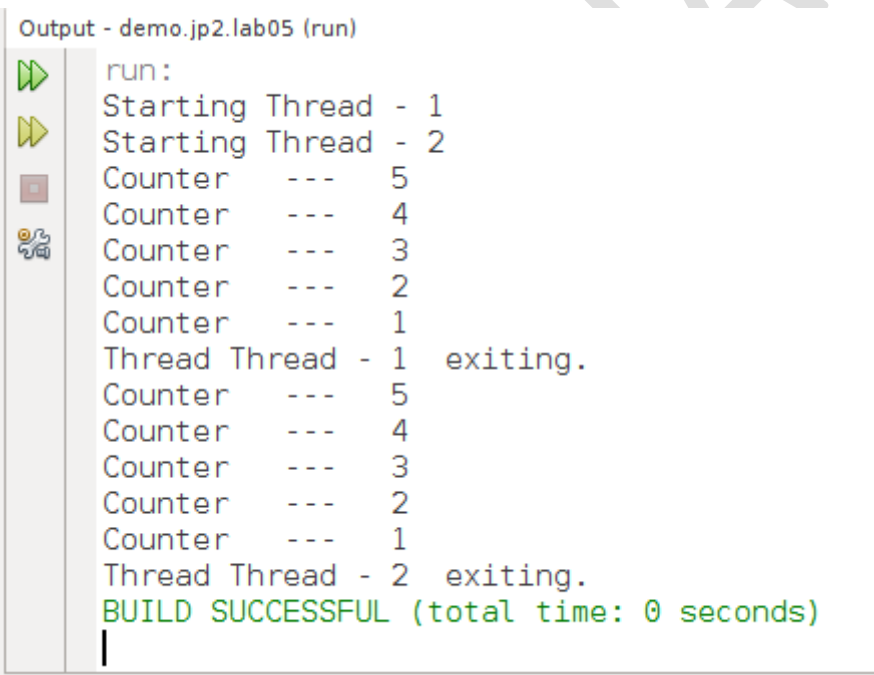
        PrintDemo PD = new PrintDemo();

        ThreadDemo T1 = new ThreadDemo("Thread - 1 ", PD);
        ThreadDemo T2 = new ThreadDemo("Thread - 2 ", PD);
```

```
T1.start();
T2.start();

// wait for threads to end
try {
    T1.join();
    T2.join();
} catch (Exception e) {
    System.out.println("Interrupted");
}
}
```

Bước 4: Xem kết quả



```
Output - demo.jp2.lab05 (run)
run:
Starting Thread - 1
Starting Thread - 2
Counter --- 5
Counter --- 4
Counter --- 3
Counter --- 2
Counter --- 1
Thread Thread - 1 exiting.
Counter --- 5
Counter --- 4
Counter --- 3
Counter --- 2
Counter --- 1
Thread Thread - 2 exiting.
BUILD SUCCESSFUL (total time: 0 seconds)
```

**Chú giải:** Thử bỏ synchronized để xem sự khác nhau

**Bài thực hành 2:** Tạo một class tên là Chat có hàm question() và hàm answer(). Tạo 2 Thread lần lượt tên là Anna và Michael, trong 2 class có biến đối tượng Chat và mảng

câu hỏi (trong Anna) và mảng câu trả lời (trong Michael). Vận dụng kiến thức về đồng bộ hóa với hàm wait(), notify() để thực hiện đoạn chat theo thứ tự.

Bước 1: Viết class Chat

```
package demo.jp2.lab05.btth2;

/**
 *
 * @authorminhvuvc
 */
class Chat {

    boolean flag = false;

    public synchronized void Question(String msg) {
        if (flag) {
            try {
                wait();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        System.out.println(msg);
        flag = true;
        notify();
    }

    public synchronized void Answer(String msg) {
        if (!flag) {
            try {
                wait();
            } catch (InterruptedException e) {
```

```
        e.printStackTrace();
    }
}

    System.out.println(msg);
    flag = false;
    notify();
}
}
```

Bước 2: Viết class Anna

```
package demo.jp2.lab05.btth2;

/**
 *
 * @authorminhvufc
 */
public class Anna implements Runnable {

    Chat m;
    String[] s1 = {"Hi", "How are you ?", "I am also doing fine!"};

    public Anna(Chat m1) {
        this.m = m1;
        new Thread(this, "Question").start();
    }

    public void run() {
        for (int i = 0; i < s1.length; i++) {
            m.Question(s1[i]);
        }
    }
}
```

```
}  
}
```

Bước 3: Viết class Michael

```
package demo.jp2.lab05.btth2;  
  
/**  
 *  
 * @authorminhvufc  
 */  
public class Michael implements Runnable {  
  
    Chat m;  
    String[] s2 = {"Hi", "I am good, what about you?", "Great!"};  
  
    public Michael(Chat m2) {  
        this.m = m2;  
        new Thread(this, "Answer").start();  
    }  
  
    public void run() {  
        for (int i = 0; i < s2.length; i++) {  
            m.Answer(s2[i]);  
        }  
    }  
}
```

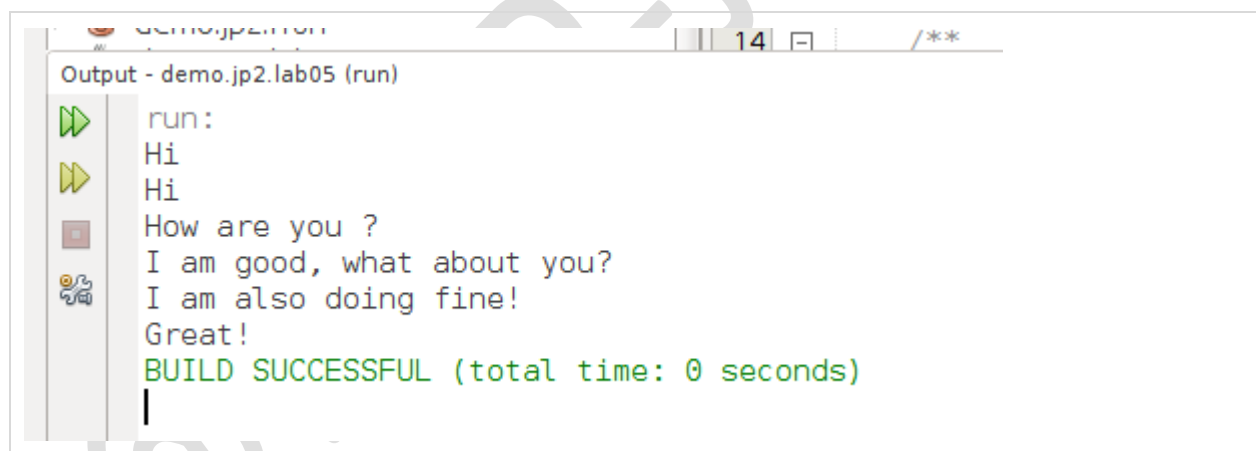
Bước 4: Viết class TestThread

```
package demo.jp2.lab05.btth2;  
  
/**  
 *
```

```
* @authorminhvuvc
*/
public class TestThread {

    /**
     * @param args thecommandlinearguments
     */
    public static void main(String[] args) {
        Chat m = new Chat();
        new Anna(m);
        new Michael(m);
    }
}
```

Bước 5: Xem kết quả



```
Output - demo.jp2.lab05 (run)
run:
Hi
Hi
How are you ?
I am good, what about you?
I am also doing fine!
Great!
BUILD SUCCESSFUL (total time: 0 seconds)
```

**Chú giải:** synchronized được đặt ở hàm question() và answer() và vì trong hàm có gọi tới wait() và notify(). Nếu không, khi chạy chương trình sẽ quăng ngoại lệ **IllegalMonitorStateException**.

**Bài thực hành 3:** Viết chương trình tính tổng các số từ 0 đến 6 (lấy  $0+1+2+3+..+6$ ). Việc tính toán yêu cầu thực hiện đồng thời trên 2 luồng, một luồng sẽ tính tổng từ 0 đến 3 và một luồng từ 4 đến 6. Do đó cần viết thread nhận vào 2 số nguyên a và b và



tính tổng cộng dồn bắt đầu từ số a đến số b. Không cần biết luồng nào thực hiện xong trước nhưng khi kết thúc cả 2 thread thì cộng kết quả của 2 luồng và hiển thị ra màn hình.

Bước 1: Viết class SumTotal

```
public class SumTotal extends Thread {  
  
    int startNumber, endNumber;  
    int total = 0;  
  
    public SumTotal(String name, int startNum, int endNum) {  
        super(name);  
        this.startNumber = startNum;  
        this.endNumber = endNum;  
    }  
  
    @Override  
    public void run() {  
        for (int i = startNumber; i <= endNumber; i++) {  
            total += i;  
            System.out.println(getName() + " is alive: " + isAlive());  
            try {  
                Thread.sleep(110);  
            } catch (InterruptedException ex) {  
                Logger.getLogger(TestNotify.class.getName()).log(Level.SEVERE, null,  
ex);  
            }  
        }  
    }  
}
```

Bước 2: Viết class TestJoin

```
package demo.jp2.lab05.btth3;
```

```
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 *
 * @author minhvuvc
 */
public class TestJoin {
    //Bước 3: Viết phương thức start()
    public void start() {
        try {
            SumTotal sum1 = new SumTotal("Thread 1", 0, 3);
            SumTotal sum2 = new SumTotal("Thread 2", 4, 16);

            sum1.start();
            sum2.start();

            sum1.join();
            System.out.println(sum1.getName() + " join");
            sum2.join();
            System.out.println(sum2.getName() + " join");

            System.out.println(sum1.getName() + " is alive: " + sum1.isAlive());
            System.out.println(sum2.getName() + " is alive: " + sum2.isAlive());
            int total = sum1.total + sum2.total;
            System.out.println("Kết quả tổng = " + total);
        } catch (InterruptedException ex) {
            Logger.getLogger(TestJoin.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    public class SumTotal extends Thread {

        int startNumber, endNumber;
        int total = 0;

        public SumTotal(String name, int startNum, int endNum) {
            super(name);
            this.startNumber = startNum;
            this.endNumber = endNum;
        }

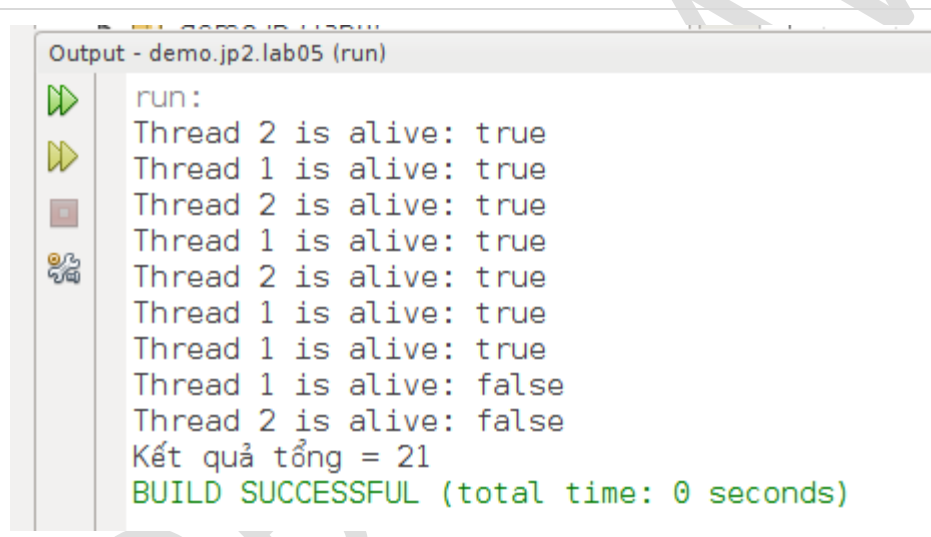
        @Override
        public void run() {
            for (int i = startNumber; i <= endNumber; i++) {
```

```

        total += i;
        System.out.println(getName() + " is alive: " + isAlive());
    }
    try {
        Thread.sleep(110);
    } catch (InterruptedException ex) {
        Logger.getLogger(TestJoin.class.getName()).log(Level.SEVERE, null,
ex);
    }
}
}
}
//Bước 4: Viết hàm main()
public static void main(String[] args) {
    TestJoin test = new TestJoin();
    test.start();
}
}

```

Bước 5: Chạy chương trình và xem kết quả



```

Output - demo.jp2.lab05 (run)
run:
Thread 2 is alive: true
Thread 1 is alive: true
Thread 2 is alive: true
Thread 1 is alive: true
Thread 2 is alive: true
Thread 1 is alive: true
Thread 1 is alive: true
Thread 1 is alive: false
Thread 2 is alive: false
Kết quả tổng = 21
BUILD SUCCESSFUL (total time: 0 seconds)

```

**Chú giải:** join() có tính chất buộc tất cả mã lệnh tiếp theo sẽ phải chờ cho đến khi luồng thực thi xong hoặc sau một khoảng thời gian xác định. Nhiều luồng cùng join thì các mã lệnh tiếp theo phải chờ cho đến khi hoàn thành tất cả luồng thì mới tiếp tục.

**Bài thực hành 4:** Tạo một class tên là ChaoHoi, trong đó khởi tạo 2 đối tượng Object là ThayDo (thầy đồ) và XaTruong (xã trưởng). Tạo 2 luồng ThayDoChao và XaTruongChao trong đó có thực hiện đồng bộ 2 đối tượng trên với yêu cầu Thầy đồ

chào thì Xã trưởng chào và ngược lại. Trong tình huống này cả 2 cùng chào nên cuộc chào hỏi không thể kết thúc.

Bước 1: Viết class ChaoHoi

```
package demo.jp2.lab05.btth4;

/**
 *
 * @author minhvuvc
 */
public class ChaoHoi {

    //Bước 2: Khai báo object
    public static Object ThayDo = new Object();
    public static Object XaTruong = new Object();

    //Bước 3: Viết class thread ThayDoChao
    private static class ThayDoChao extends Thread {

        public void run() {
            synchronized (ThayDo) {
                System.out.println("Thầy đồ: Cúi người chào...");
                try {
                    Thread.sleep(10);
                } catch (InterruptedException e) {
                }
                System.out.println("Thầy đồ: đợi xã trưởng chào lại...");
                synchronized (XaTruong) { // Deadlock
                    System.out.println("Thầy đồ: kết thúc chào hỏi...");
                }
            }
        }
    }

    //Bước 4: Viết thread XaTruongChao
    private static class XaTruongChao extends Thread {

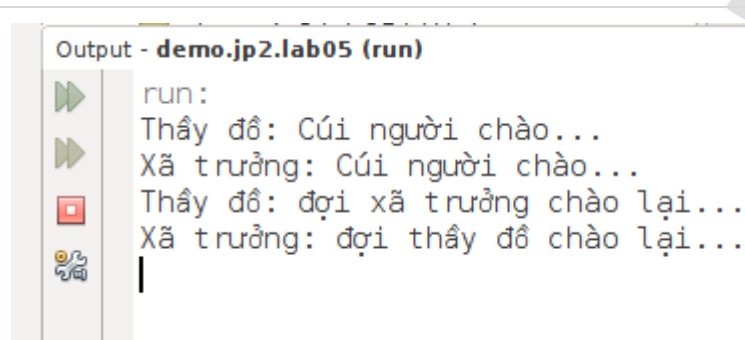
        public void run() {
            synchronized (XaTruong) {
                System.out.println("Xã trưởng: Cúi người chào...");
                try {
                    Thread.sleep(10);
                } catch (InterruptedException e) {
                }
                System.out.println("Xã trưởng: đợi thầy đồ chào lại...");
                synchronized (ThayDo) { // Deadlock
                }
            }
        }
    }
}
```

```
        System.out.println("Xã trưởng: kết thúc chào hỏi...");
    }
}

//Bước 5: Viết hàm main()
public static void main(String args[]) {

    ThayDoChao thayDo = new ThayDoChao();
    XaTruongChao xaTruong = new XaTruongChao();
    thayDo.start();
    xaTruong.start();
}
}
```

Bước 6: Chạy chương trình và xem kết quả



```
Output - demo.jp2.lab05 (run)
run:
Thấy đồ: Cúi người chào...
Xã trưởng: Cúi người chào...
Thấy đồ: đợi xã trưởng chào lại...
Xã trưởng: đợi thấy đồ chào lại...
```

**Chú giải:** tình huống này gọi là DeadLock khi các luồng đều chờ đợi khóa của nhau nhả ra để thực hiện tiếp. Đoạn comment // Deadlock chính là chỗ luồng bị dừng xử lý và rơi vào trạng thái nghỉ.

## Phần II - Bài tập tự làm

1. Yêu cầu người dùng nhập một số nguyên từ bàn phím (ví dụ là 10), sau đó yêu cầu nhập số luồng muốn tạo (ví dụ 2). Yêu cầu dùng Thread để tính tổng đồng thời bằng số luồng sau đó hiển thị kết quả. Ví dụ trong tình huống này là 1 luồng sẽ tính tổng ( $0 + 1 + 2 + \dots + 5$ ) và luồng thứ 2 tính tổng ( $6 + 7 + \dots + 10$ ).
2. Làm quen với đồng bộ hai hoặc nhiều thread:  
Tạo 2 thread:
  - Thread thứ nhất sinh ra một số ngẫu nhiên từ 0 đến 50

- Thread thứ hai nhận số ngẫu nhiên này và hiển thị đó là số chẵn hay số lẻ  
Đồng bộ 2 thread này để đảm bảo thread thứ hai luôn luôn chạy sau thread thứ nhất.

**3.** Cho mảng tên chưa được chuẩn hóa như sau:

```
String[] listName = {"nGuYeN hOa bInH", "lE tHaNh hAi", "tRan mAnH tIeN", "lE qUaNg qUaN"};
```

Tạo thread thứ nhất, cứ sau mỗi giây hiển thị một tên ngẫu nhiên trong mảng trên.

Tạo thread thứ hai sẽ nhận về tên tương ứng do thread1 vừa trả về, chuẩn hóa tên và hiển thị tên đã chuẩn hóa ra màn hình.

Đồng bộ 2 thread này.

Thực hiện xử lý để mỗi thread sẽ thực hiện 5 lần rồi kết thúc.

**Gợi ý:** Tạo một đối tượng chứa mảng, và 1 hàm sẽ trả về ngẫu nhiên 1 tên trong mảng. Truyền đối tượng này vào 2 Thread và synchronized.

**4.** Tạo thread thứ nhất cứ sau mỗi giây hiển thị một số ngẫu nhiên từ 0 đến 50.

Trong thread 1 kiểm tra nếu số này là:

Số chẵn thì sẽ chuyển đến thread 2 để thực thi

Số lẻ thì sẽ chuyển đến thread 3 để thực thi.

Thread 2 sau khi nhận được điều khiển từ thread 1 sẽ in ra số đó và tất cả các ước số của nó.

Thread 3 khi nhận được điều khiển từ thread 1 sẽ in ra số và bình phương của số đó.

Đồng bộ 3 thread này.

**5.** Tạo thread thứ nhất sau mỗi giây sẽ sinh ra một số ngẫu nhiên từ 1 đến 100

Thread thứ hai sẽ lấy số ngẫu nhiên này và in ra số đó là số chẵn hay số lẻ

Đồng bộ 2 thread này

**6.** Tạo thread thứ nhất sau mỗi giây sẽ sinh ra một số ngẫu nhiên từ 1 đến 100

Thread thứ hai lấy số ngẫu nhiên này và in ra các ước số của nó

Đồng bộ 2 thread