

**BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC MỎ - ĐỊA CHẤT**



BÁO CÁO HỌC THUẬT

ĐỀ TÀI: LẬP TRÌNH ĐA TIẾN TRÌNH VÀ ỨNG DỤNG

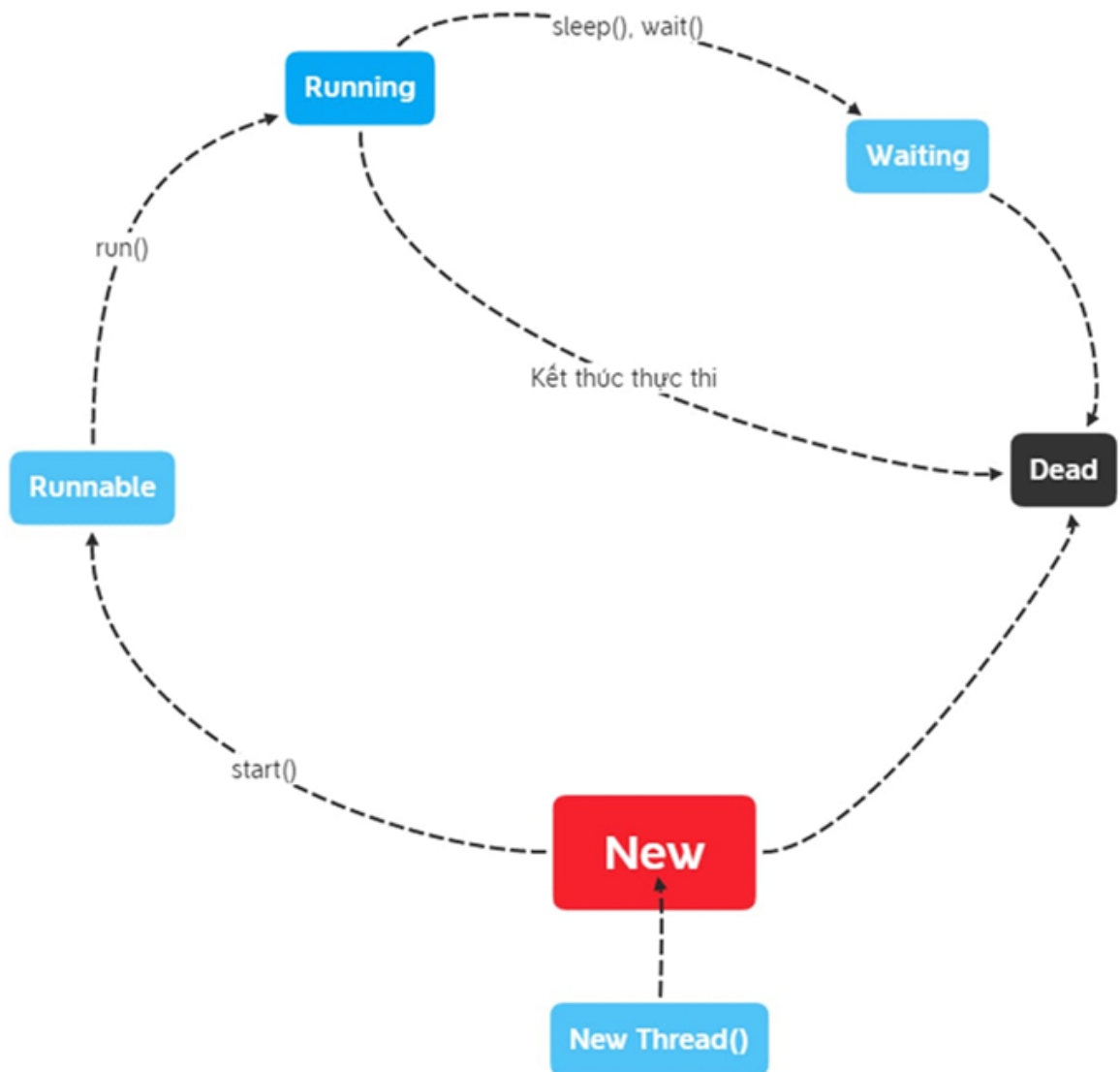
**Người thực hiện: Nguyễn Tuấn Anh
Bộ môn: Mạng máy tính – Khoa Công nghệ thông tin**

HÀ NỘI- 1/2025

MỤC LỤC

Chương 1 -LẬP TRÌNH ĐA TUYẾN.....	3
1.1. TIỀN TRÌNH.....	3
1.2. QUẢN LÝ THREAD	4
1.3. CÁCH MỘT SỬ DỤNG LỚP THREAD	4
1.4. CÁCH HAI IMPOLEMENTS INTERFACE RUNNABLE	5
1.5. ĐỒNG BỘ	6
1.5.1. Đồng bộ hóa sử dụng cho phương thức	6
1.5.2. Lệnh synchronized	8
1.6. TRAO ĐỔI DỮ LIỆU GIỮA CÁC THREAD	9
Chương 2 -XÂY DỰNG CHƯƠNG TRÌNH PHÂN PHỐI SẢN PHẨM	10
2.1. ĐẶT VẤN ĐỀ	10
2.2. XÂY DỰNG CHƯƠNG TRÌNH.....	10
KẾT LUẬN	14
TÀI LIỆU THAM KHẢO	15

1.1. TIẾN TRÌNH



Hình 1.1. Lập trình trên Visual Studio Code và chạy trực tiếp chương trình
Vòng đời của thread trong java được kiểm soát bởi JVM

Java định nghĩa các trạng thái của luồng trong các thuộc tính static của lớp **Thread.State**:

- **NEW** : Đây là trạng thái khi luồng vừa được khởi tạo bằng phương thức khởi tạo của lớp **Thread** nhưng chưa được **start()**. Ở trạng thái này, luồng được tạo ra nhưng chưa được cấp phát tài nguyên và cũng chưa chạy. Nếu luồng đang ở trạng thái này mà ta gọi các phương thức ép buộc **stop, resume, suspend ...** sẽ là nguyên nhân xảy ra ngoại lệ **IllegalThreadStateException**.

- **RUNNABLE** : Sau khi gọi phương thức `start()` thì luồng test đã được cấp phát tài nguyên và các lịch điều phối CPU cho luồng test cũng bắt đầu có hiệu lực. Ở đây, chúng ta dùng trạng thái là `Runnable` chứ không phải `Running`, vì luồng không thực sự luôn chạy mà tùy vào hệ thống mà có sự điều phối CPU khác nhau.
- **WAITING** : Thread chờ không giới hạn cho đến khi một luồng khác đánh thức nó.
- **TIMED_WAITING** : Thread chờ trong một thời gian nhất định, hoặc là có một luồng khác đánh thức nó.
- **BLOCKED**: Đây là 1 dạng của trạng thái “Not Runnable”, là trạng thái khi Thread vẫn còn sống, nhưng hiện tại không được chọn để chạy. Thread chờ một monitor để unlock một đối tượng mà nó cần.
- **TERMINATED** : Một thread ở trong trạng thái terminated hoặc dead khi phương thức `run()` của nó bị thoát.

1.2. QUẢN LÝ THREAD



Hình 1.2. Quản lý tiến trình

1.3. CÁCH MỘT SỬ DỤNG LỚP THREAD

Để lập trình đa tiến trình, lớp dẫn xuất từ lớp `Thread` được cài đặt sau đó override phương thức `run()`. Khi tiến trình cần chạy thì phương thức `start()` của tiến trình được gọi thực hiện.

App0401.java: Tạo ra hai tiến trình T1 và T2, sau đó gọi thực thi chúng. Hai tiến trình thay nhau in lên màn hình số nguyên trong khoảng từ 1 đến 99;

```

package DATUYEN; // Tạo thư mục là DATUYEN để lưu các App về đa tuyến
public class App0401
{
    public static void main(String[] args)
    {
        MyThread t1=new MyThread(); // Tạo tiến trình
        MyThread t2=new MyThread();
        t1.start(); // Gọi thực thi tiến trình
        t2.start();
    }
}
  
```

```

}
class MyThread extends Thread // Kế thừa lớp Thread
{
    public void run() // Override phương thức run
    {
        int i=0;
        while(i<100)
        {
            System.out.println(getName() + " "+i);
            i++;
        }
    }
}

```

Kết quả thực hiện chương trình:

```

Thread-0 1 Tiến trình 0 in số 1
Thread-0 2
Thread-0 3
..
Thread-1 85
Thread-0 98
Thread-1 99

```

1.4. CÁCH HAI IMPOLEMENTS INTERFACE RUNNABLE

Nếu không thể tạo class kế thừa từ Thread, ta có thể cho nó implement interface Runnable. Khi đó, class phải hiện thực phương thức **void** run() của interface này.

Sau đó, ta tạo thread liên kết với Runnable thông qua constructor tương ứng của lớp Thread. Khi thread được start, nó sẽ gọi thực thi lệnh trong phương thức run() của Runnable liên kết với nó.

App0402.java:

```

package DATUYEN;
class MyNumber implements Runnable
{
    private String threadName;
    public MyNumber(String name){threadName=name; }
    public void run() // Cài đặt phương thức run của Runnable
    {
        int i=0;
        while(i<10)
        {
            System.out.println(threadName+" "+i);
            i++;
        }
    }
}
public class App0402
{
    public static void main(String[]args)
    {

```

```

Runnable r1=new MyNumber("Thread 1"); // Tạo runnable
Runnable r2=new MyNumber("Thread 2");
Thread t1=new Thread(r1);    // Tạo thread liên kết
Thread t2=new Thread(r2);    // với runnable
t1.start();
t2.start();
}
}

```

Kết quả thực hiện chương trình:

```

Thread 2 0
Thread 2 1
Thread 2 2
Thread 1 0
Thread 1 1
Thread 2 3

```

Cách hay được sử dụng và được yêu thích là dùng **interface Runnable**, bởi vì nó không yêu cầu phải tạo một lớp kế thừa từ lớp Thread. Trong trường hợp ứng dụng thiết kế yêu cầu sử dụng đa kế thừa, chỉ có interface mới có thể giúp giải quyết vấn đề. Ngoài ra, **Thread Pool** rất hiệu quả và có thể được cài đặt, sử dụng rất hơn giản.

1.5. ĐỒNG BỘ

1.5.1. Đồng bộ hóa sử dụng cho phương thức

Nếu nhiều thread cùng truy cập vào một vùng nhớ, lỗi có thể xảy ra. Để chỉ cho phép trong một lúc chỉ có tối đa một thread truy cập vào vùng nhớ, ta cần đồng bộ hóa bằng cách dùng từ khóa **synchronized**.

App0403.java

```

package DATUYEN;
public class App0403 extends Thread
{
    public static void main(String[]args)
    {
        MyThread[] T=new MyThread[3];
        for(int i=0;i<T.length;i++)
        {
            T[i]=new MyThread();
            T[i].start();
        }
    }
}
class MyThread extends Thread
{
    public static int x=5;
    public static boolean stop=false;
    public static /*synchronized*/ void Trans()
    {

```

```

        if (x>0)
        {
            try
            {
                sleep(100);
            }
            catch(Exception E)
            {
                return;
            }
            x--;
        }
    }
}
public void run()
{
    do
    {
        Trans();
        System.out.println("x = "+x);
    }
    while(x>0);
}
}

```

Kết quả thực hiện chương trình:

```

x = 2
x = 2
x = 2
x = 1
x = 0
x = -1

```

Theo đoạn code thì các tiểu trình sẽ chỉ chạy khi $x > 0$. Nhưng kết quả thực thi khi không đồng bộ.

Chúng ta giải quyết vấn đề này bằng cách thêm từ khóa synchronized như sau `public static synchronized void Trans(){ }`

Kết quả thực hiện chương trình:

```

x = 4
x = 3
x = 2
x = 1
x = 0
x = 0

```

$$x = 0$$

Như vậy, trong thực tế nếu không dùng đồng bộ hóa, thì lại xảy ra trường hợp $x < 0$. Nguyên nhân như sau:

- Khi không đồng bộ, lúc $x = 1$, một tiểu trình nào đó kiểm tra thấy thỏa yêu cầu > 0 và giảm x . Nhưng trong lúc x chưa kịp bị giảm thì tiểu trình khác cũng kiểm tra và thấy x vẫn thỏa nên lại giảm x . Kết quả là x sẽ bị giảm nhiều lần và sau cùng x sẽ có giá trị âm.
- Khi thêm từ khóa **synchronized** vào, trong một thời điểm chỉ có một tiểu trình được truy cập vào phương thức `Trans()` nên sẽ không bị lỗi nữa.

Một điều cần lưu ý là khi đồng bộ hóa, tốc độ xử lý sẽ chậm hơn khi không đồng bộ, nên ta không được làm dụng việc đồng bộ mà chỉ dùng khi cần thiết.

1.5.2. Lệnh **synchronized**

Lệnh **synchronized** cho phép đồng bộ hóa một đối tượng mà không cần yêu cầu bạn tác động một phương thức **synchronized** trên đối tượng đó.

Cú pháp

```
synchronized (expr)
    statement
```

Khi có được khóa, `statement` được xử lý giống như nó là một phương thức **synchronized** trên đối tượng đó.

Ví dụ: Chuyển các phần tử trong mảng thành các số không âm, [App0404.java](#)

```
public static void abs(int[] v)
{
    synchronized(v)
    {
        for(int i=0;i<v.length;i++)
        {
            if(v[i]<0) v[i]=-v[i];
        }
    }
}
```

1.6. TRAO ĐỔI DỮ LIỆU GIỮA CÁC THREAD

Để trao đổi dữ liệu giữa các thread, ta dùng các lớp `PipedOutputStream` (xuất dữ liệu) và `PipedInputStream` (nhận dữ liệu). Lưu ý: cần import package `java.io`.

Để kết nối stream xuất và nhận, ta dùng phương thức `connect()` của các pipe. Sau khi đã kết nối, ta dùng các phương thức `read()` và `write()` để đọc / ghi dữ liệu.

Chương 2 -XÂY DỰNG CHƯƠNG TRÌNH PHÂN PHỐI SẢN PHẨM

2.1. ĐẶT VẤN ĐỀ

Hệ thống kinh doanh có **kho hàng**, trong kho hàng có **nhiều mặt hàng**, mỗi mặt hàng có **nhiều nhà cũng sản xuất**, mỗi nhà sản xuất cũng có thể sản xuất **nhiều mặt hàng**. Có nhiều khách hàng mà mỗi khách hàng có thể mua nhiều mặt hàng. Hai việc sản xuất và mua hàng(bán hàng) luôn luôn diễn ra đồng thời nghĩa là nhà sản xuất không đợi người mua hàng mua hết hàng rồi mới sản xuất hàng, người mua hàng cũng vậy. Các hoạt động như vậy coi như các tiến trình xảy ra đồng thời.

2.2. XÂY DỰNG CHƯƠNG TRÌNH

1- Lớp nhà sản xuất

```
package BAOCAOSHHTK120242025;
public class Producer extends Thread {
    private int id;// ma nsx
    private Buffer buffer_kho; // Mã kho chứa hàng trong kho lớn(buffer),
và khả năng sx Capacity)
    public Producer() {
        id = 0;
        buffer_kho.setCapacity(0);
    }
    public Producer(int i, Buffer buffer) {
        this.id = i;
        this.buffer_kho = buffer;// 1 nhà sx có 1 kho (1 buffer: khả năng
sx, danh sách sp của mình)
    }
    public void setData(int i, int c) {
        id = i;
        buffer_kho.setCapacity(c);
    }
    public void setID(int i) {
        id = i;
    }
    public void printProcedure()
    {
        System.out.println("|"+id+"\t"+buffer_kho.getCapacity()+"|");
    }
    @Override
    public void run() {
        int i = 0;
        while (true) {
            if (buffer_kho.getCapacity() > buffer_kho.getSize()) // khả
năng =6 > hàng thực còn tức là vẫn còn chỗ trống
            {
                try {
```

```

        buffer_kho.addProduct(i++, this.id);// thêm mã nhsx và
        mã sp(thêm vào mảng products trong buffer
        sleep(500);//(long) Math.random() *
100);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
}
}
}
}
}
}

```

2- Lớp Kho hàng

```

package BAOCAOSHHTK120242025;
import java.util.ArrayList;
public class Buffer {
    // khả năng chứa tối đa
    private int capacity;
    // danh sách sản phẩm
    private ArrayList<Integer> products = new ArrayList<>(); // dùng vector
    để lưu mảng sp có mã, có tên..
    public Buffer() {
        this.capacity = 0;
    }
    public Buffer(int capacity) {
        this.capacity = capacity;
    }
    public void addProduct(int product, int producerId) {
        System.out.println(">>>>-----");
    };
        System.out.println("Nhà sản xuất: " + producerId + "đã thêm sản
        phẩm " + product);
        products.add(product);
        System.out.println("Số lượng tồn kho: " + products.size());// lấy
        ra số lượng sp
    }
    public void removeProduct(int consumerId) {
        System.out.println("<<<<-----");
    };
        System.out.println("Khách hàng " + consumerId + "đã mua sản phẩm "
        + products.get(0));// vị trí đầu tiên
        products.remove(0);
        System.out.println("Số lượng tồn kho: " + products.size());
        //Số lượng tồn kho; lượng hàng thực sự có, thực tế còn nhiều chỗ
        trống
    }
    public int getSize() {
        return this.products.size();
    }
    public int getCapacity() {
        return capacity;
    }
}

```

```

    }
    public void setCapacity(int c) {
        capacity = c;
    }
}

```

3- Lớp khách hàng

```

package BAOCAOSHHTHK120242025;
public class Consumer extends Thread {
    private int id;
    private Buffer buffer;

    public Consumer(){}
    public Consumer(int id, Buffer buffer){
        this.id = id;
        this.buffer= buffer;
    }

    @Override
    public void run(){
        @SuppressWarnings("unused")
        int i = 0;
        while (true) {
            if(buffer.getSize()>0){
                try {
                    buffer.removeProduct(this.id);
                    sleep(50);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        }
    }
}

```

4- Chương trình chính

```

package BAOCAOSHHTHK120242025;
import java.nio.channels.NetworkChannel;

@SuppressWarnings("unused")
public class SanxuatCungungHh {
    static public void menu() {
        System.out.println("1.Nhap nha san xuat moi");
        System.out.println("2.Nhap khach hang moi");
        System.out.println("0.Thoat");
    }
    public static void main(String[] args) {
        Buffer buffer = new Buffer(50);
        Producer p1 = new Producer(1, buffer);// mã nhà sx
        Producer p2 = new Producer(2, buffer);
        Consumer c1 = new Consumer(1,buffer);
    }
}

```

```
        Consumer c2 = new Consumer(1,buffer);  
        p1.start();  
        p2.start();  
        c1.start();  
        c2.start();  
    }  
}
```

KẾT LUẬN

Kết quả của nghiên cứu có thể được đưa vào giảng dạy trong học phần Lập trình mạng, ứng dụng cho bài toán quản lý sản xuất hàng hoá.

TÀI LIỆU THAM KHẢO

Tài liệu tiếng Việt:

- [1]. Nguyễn Tuấn Anh, *Kỹ thuật lập trình hướng đối tượng bằng C++*, Nhà xuất bản Giáo dục Việt Nam, 2011.
- [2]. Nguyễn Tuấn Anh, *Lập trình mạng, Trường Đại học Mở-Đia chất*
- [3]. Nguyễn Phương Lan- Hoàng Đức Hải, *Java lập trình mạng*, Nhà xuất bản Giáo dục.
- [4]. Nguyễn Thúc Hải, *Mạng máy tính và các hệ thống mở*, Nhà xuất bản Giáo dục.

Tài liệu tiếng Anh:

- [5]. Eliot Rusty Harold, *JavaTM Network Programming, Third Edition*, O'Reilly, 2005.
- [6]. Qusay H. Mahmoud, *Advanced Socket Programming*, <http://java.sun.com>, December 2001.