

Chương 4: LẬP TRÌNH ĐA TIỂU TRÌNH

Khoa CNTT

ĐH GTVT TP.HCM

- 1 Giới thiệu
- 2 Lập trình multithread
- 3 Giải quyết tương tranh (xung đột) & Đồng bộ hóa

Concurrency

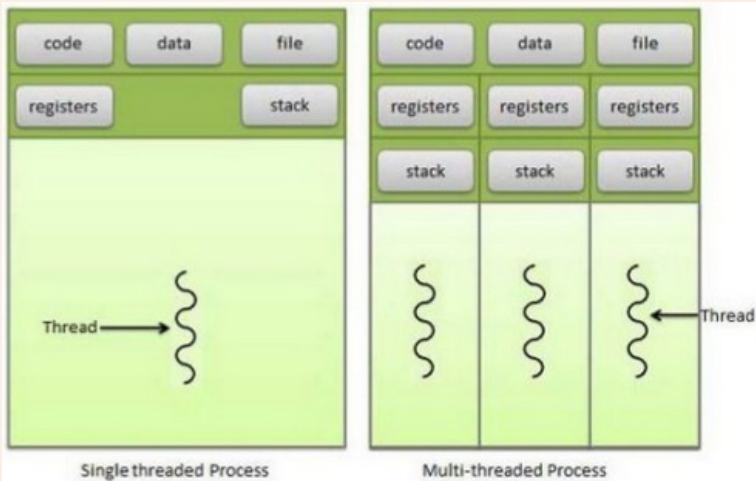
- * Máy tính ngày nay cho phép ta sử dụng một lúc nhiều ứng dụng, chẳng hạn như bạn vừa nghe nhạc, vừa đánh văn bản word, vừa download nhạc
- * Hay thậm chí là một ứng dụng đơn cũng thực hiện nhiều task ở cùng một thời điểm.
- * Ví dụ, trình soạn thảo văn bản word, nó luôn luôn sẵn sàng đáp ứng các sự kiện về keyboard và mouse, nó vừa phải reformat text và cập nhật lại màn hình.
- * Các phần mềm làm những task như vậy gọi là phần mềm đồng bộ.

Processes và Thread

- * Trong một tiến trình (process) có thể có nhiều threads chạy đồng thời.
- * Các threads chia sẻ cùng một tài nguyên của tiến trình, bao gồm bộ nhớ và các file, ...
- * Điều này làm cho giao tiếp hiệu quả nhưng lại tiềm ẩn bên trong nó các vấn đề về xử lý tranh chấp tài nguyên giữa các threads.

Multithread

Minh họa Multithread



Tạo thread: Implements interface Runnable

```
public class HelloRunnable implements Runnable {  
    public void run() {  
        System.out.println("Hello from a thread!");  
    }  
    public static void main(String args[]) {  
        (new Thread(new HelloRunnable())).start();  
    }  
}
```

Multithread

Tạo thread: Extends class Thread

```
public class HelloThread extends Thread {  
    public void run() {  
        System.out.println("Hello from a thread!");  
    }  
    public static void main(String args[]) {  
        (new HelloThread()).start();  
    }  
}
```

Question

Khi nào implements Runnable, còn khi nào extends Thread?

Thread bao gồm các trạng thái sau (1):

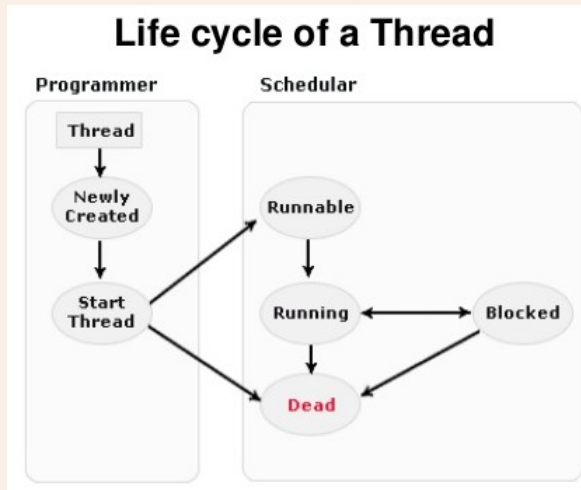
- * **New**: sau khi tạo thread
- * **Runnable**: sau khi start() → thread chuyển sang trạng thái Runnable
- * **Blocked**: thread ở trạng thái blocked nếu:
 - 1 Gọi sleep();
 - 2 Thread gọi 1 thao tác mà nó đang bị block trên IO
 - 3 Thread cố gắng dành lock - khóa (trong khi khóa này đang được giữ bởi thread khác)
 - 4 Thread đợi 1 điều kiện nào đó để thực thi

Thread bao gồm các trạng thái sau (2):

* **Dead** (terminated): thread ở trạng thái này khi:

- ❶ Thực thi xong phương thức run()
- ❷ Xảy ra 1 exception chưa được catch

Thread bao gồm các trạng thái sau (3):



Yield, Sleep & Wait

- * **Yield:** This static method is essentially used to notify the system that the current thread is willing to "give up CPU" for a while. Thread scheduler will select a different thread to run.
- * **Sleep:** pauses the current thread for a given period of time (millisecond). Sleep can be interrupted.
- * **Wait:** means it will be blocked until `notify()` or `notifyAll()` is called.

Thread Scheduler & Priority

- * **Scheduler** là một thành phần của JVM, có vai trò quyết định thread nào sẽ chạy (dựa trên độ ưu tiên của thread).
- * **Priority** có giá trị từ 1 đến 10, giá trị mặc định là 5.

Tương tranh (race condition)

Khi nào xảy ra tương tranh?

- * Tương tranh xảy ra khi có hai hay nhiều thread cùng tranh giành truy cập vào tài nguyên chung của chương trình, trong khi tài nguyên chung này đòi hỏi phải được truy cập theo trình tự.
- * Đoạn mã lệnh bên trong một thread gây ra tình huống tương tranh được gọi là đoạn mã tới hạn (critical section).
- * Có thể tránh được tình huống tương tranh bằng cách đồng bộ hóa các đoạn mã tới hạn một cách đúng đắn, sao cho tài nguyên chung không được phép truy cập đồng thời bởi nhiều hơn 1 thread.

Tương tranh (race condition)

Minh họa



Tương tranh (race condition)

```
public class BanVeXeBuyt implements Runnable {
    private int soGheTrong = 2;
    @Override
    public void run() {KhachThread khach = (KhachThread) Thread.currentThread();
        boolean datDuoc = this.banVe(khach.laySoGheDat(), khach.getName());
        if (datDuoc == true) {"Chuc mung " +
System.out.println(Thread.currentThread().getName() + ", " +
khach.laySoGheDat() + " ghe da duoc dat.");
        } else {System.out.println("Xin loi " + Thread.currentThread().getName()
+ " khong du so ghe yeu cau (" + khach.laySoGheDat() + ")");
        }
    }
}

//...
```

Tương tranh (race condition)

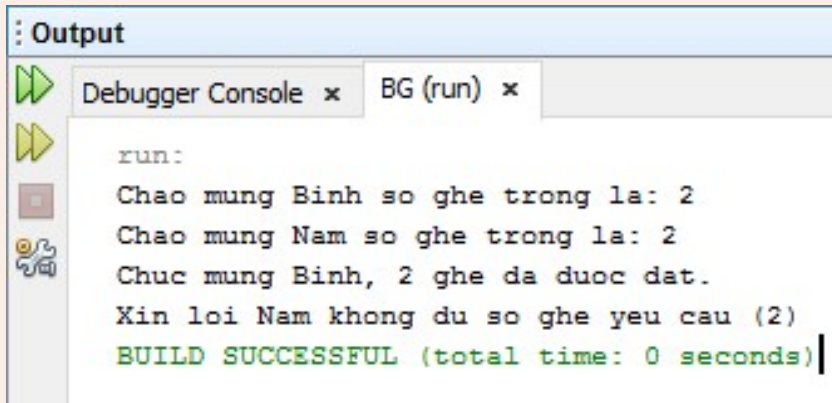
```
//...
private boolean banVe(int soGheDat, String hoTen) {
    System.out.println("Chao mung " + hoTen + " so ghe trong la: " +
this.laySoGheTrong());
    if (soGheDat > this.laySoGheTrong()) {
        return false;
    } else {
        soGheTrong = soGheTrong - soGheDat;
        return true;
    }
}
private int laySoGheTrong() {
    return soGheTrong;
}
}
```


Tương tranh (race condition)

```
public class KhachThread extends Thread {  
    private int soGheDat;  
    public KhachThread(int gh, Runnable daiLy, String hoTen) {  
        super(daiLy, hoTen);  
        this.soGheDat = gh;  
    }  
    public int laySoGheDat() {  
        return soGheDat;  
    }  
}  
  
//in main method:  
BanVeXeBuyt bus = new BanVeXeBuyt();  
KhachThread nam = new KhachThread(2, bus, "Nam");  
KhachThread binh = new KhachThread(2, bus, "Binh");  
nam.start();  
binh.start();
```

Tương tranh (race condition)

Kết quả



```
run:
Chao mung Binh so ghe trong la: 2
Chao mung Nam so ghe trong la: 2
Chuc mung Binh, 2 ghe da duoc dat.
Xin loi Nam khong du so ghe yeu cau (2)
BUILD SUCCESSFUL (total time: 0 seconds)
```

Đồng bộ (Synchronization)

Synchronized method

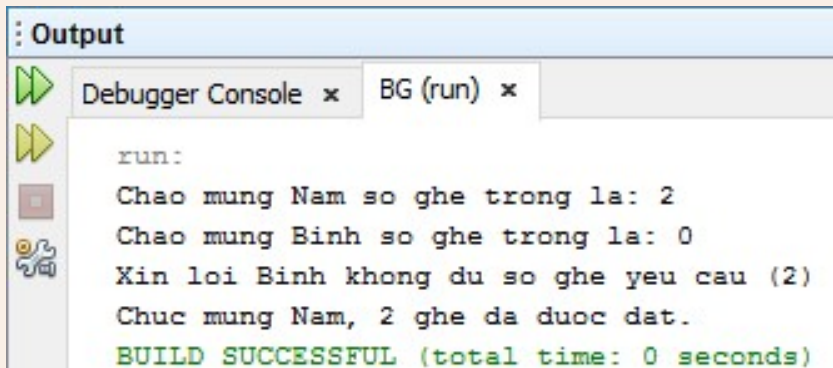
```
private synchronized boolean banVe(int soGheDat, String hoTen) {  
    System.out.println("Chao mung " + hoTen + " so ghe trong la: " +  
        this.laySoGheTrong());  
    if (soGheDat > this.laySoGheTrong()) {  
        return false;  
    } else {  
        soGheTrong = soGheTrong - soGheDat;  
        return true;  
    }  
}
```

Synchronized block

```
private boolean banVe(int soGheDat, String hoTen) {  
    synchronized(this){/*critical section*/}  
}
```

Đồng bộ (Synchronization)

Kết quả sau khi **synchronized**



The screenshot shows an IDE's Output window with a tab labeled "BG (run)". The output text is as follows:

```
run:  
Chao mung Nam so ghe trong la: 2  
Chao mung Binh so ghe trong la: 0  
Xin loi Binh khong du so ghe yeu cau (2)  
Chuc mung Nam, 2 ghe da duoc dat.  
BUILD SUCCESSFUL (total time: 0 seconds)
```

Tương tranh (race condition)

Sử dụng ReentrantLock (`java.util.concurrent.locks.ReentrantLock`)

```
Lock lock = new ReentrantLock();  
lock.lock();  
//critical section  
lock.unlock();
```

Tương tranh (race condition)

Difference between Lock Interface and synchronized keyword

- * Having a timeout trying to get access to a synchronized block is not possible. Using `Lock.tryLock(long timeout, TimeUnit timeUnit)`, it is possible.
- * The synchronized block must be fully contained within a single method. A Lock can have its calls to `lock()` and `unlock()` in separate methods.

—Hết—