Fpt University

FPT POLYTECHNIC

LẬP TRÌNH JAVA

BÀI 3: Đa luồng

www.poly.edu.vn

Nhắc lại bài trước

- Các loại luồng dữ liệu
- Xử lý nhập xuất bằng luồng byte
- Truy cập file ngẫu nhiên
- Xử lý nhập xuất bằng luồng character
- Sử dụng try… catch trong nhập/xuất
- Chuyển đổi dữ liệu kiểu số

Nội dung bài học

- Khái niệm multitasking và multithreading
- Khái niệm ‘thread’ – luồng
- Thread hiện thời
- Các trạng thái của thread
- Khởi tạo thread
- Quản lý thread

Khái niệm Multitasking và Multithreading

Multitasking: Là khả năng chạy đồng thời một hoặc nhiều chương trình cùng một lúc trên một hệ điều hành.

- Internet Explorer
- Microsoft Excel
- Window Media Player

Multithreading: Là khả năng thực hiện đồng thời nhiều phần khác nhau của một chương trình được gọi là thread.

- Sheet1
- Sheet2
- Sheet3

Thread là gì?

Thread là **đơn vị nhỏ nhất** của mã thực thi mà đoạn mã đó thực hiện một nhiệm vụ cụ thể.

Một ứng dụng có thể được **chia nhỏ** thành nhiều nhiệm vụ và mỗi nhiệm vụ có thể được giao cho một thread.

Nhiều thread cùng thực hiện **đồng thời** được gọi là đa luồng (multithread).

Các quá trình đang chạy **dường như** là đồng thời, nhưng thực ra nó không phải là như vậy.

Current thread

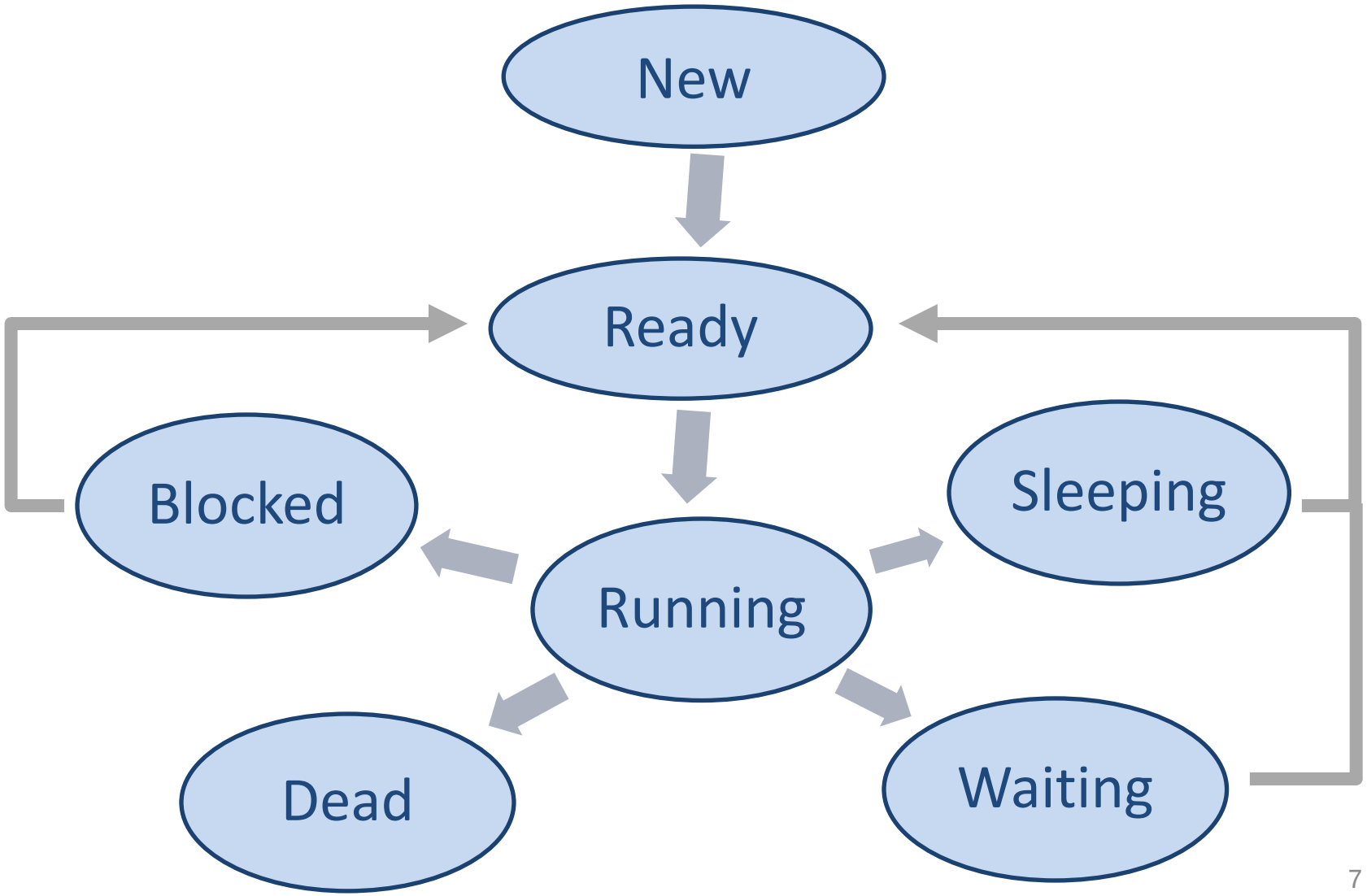
Current thread: *Là thread hiện tại đang hoạt động.*

```
public class CurrentThreadDemo {
```

```
run-single:
Chi tiet thread hien tai:Thread[main,5,main]
Do uu tien: 5
id cua thread:1
Ten thread:main
So luong thread dang hoat dong:1
Trang thai cua thread:RUNNABLE
Thread con ton tai:true
BUILD SUCCESSFUL (total time: 2 seconds)
```

```
}
```

Vòng đời của một thread



Các trạng thái của thread

New: Một thread ở trạng thái ‘new’ nếu bạn tạo ra một đối tượng thread nhưng chưa gọi phương thức start().

Ready: Sau khi thead được tạo, nó sẽ ở trạng thái sẵn sàng (ready) chờ phương thức start()gọi nó.

Các trạng thái của thread

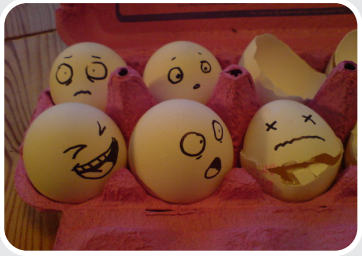
Running: Thread ở trạng thái chạy (đang làm việc)

Sleeping: Phương thức **sleep()** sẽ đưa thead vào trạng thái ‘sleeping’ - dừng lại tạm thời. Sau thời gian ‘sleeping’ thread lại tiếp tục hoạt động.

Các trạng thái của thread

Waiting: Khi method **wait()**hoạt động, thread sẽ rơi vào trạng trạng thái ‘waiting’-đợi. Method này được sử dụng khi hai hoặc nhiều thread cùng đồng thời hoạt động.

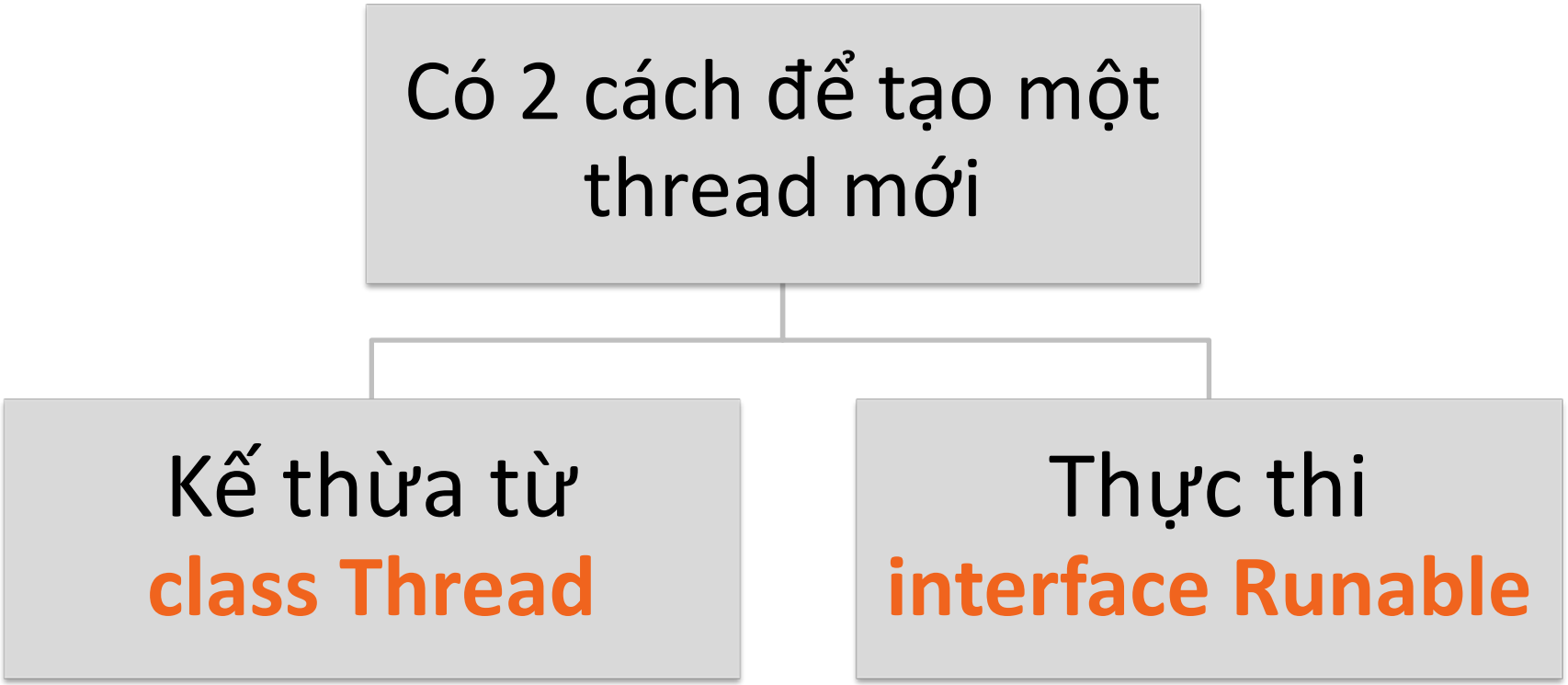
Blocked: Thread sẽ rơi vào trạng thái ‘blocked’-bị chặn khi thread đó đang đợi một sự kiện nào đó của nó như là sự kiện Input/Output.



Dead: Thread rơi vào trạng thái ‘dead’-ngừng hoạt động sau khi thực hiện xong phương thức **run()** hoặc gọi phương thức **stop().**

Khởi tạo thread

Hệ thống xử lý đa luồng trong Java được xây dựng trên class Thread và interface Runnable trong packaged java.lang.



Khởi tạo thread

Tạo thread bằng cách sử dụng interface Runable:

1. Viết 1 class thực thi **interface Runable** và viết lại phương thức '**public void run()**'
2. Tạo ra 1 object vừa thực thi interface Runnable.
3. Tạo ra 1 object của class Thread với tham số truyền vào là object thực thi interface Runable.
4. Gọi phương thức **start()** để chạy thread

Khởi tạo thread

File FirstThread.java

```
public class FirstThread implements Runnable {
    public void run() {
        for (int i = 1; i <= 10; i++) {
            System.out.println("Messag from First Thread : " + i);
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {

                System.out.println("First Thread error: " + e);
            }
        }
    }
}
```

Khởi tạo thread

File SecondThread.java

```
public class SecondThread implements Runnable {
    public void run() {
        for (int i = 1; i <= 10; i++) {
            System.out.println("Messag from Second Thread: " + i);
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                System.out.println("Second Thread error:" + e);
            }
        }
    }
}
```

Khởi tạo thread

File ThreadDemo.java

```
public class ThreadDemo {
    public static void main(Strin
        //Creating an object of t
        FirstThread firstThread =
        SecondThread secondThread

        //Starting the first thre
        Thread thread1 = new Thr
        thread1.start();

        //Starting the second thr
        Thread thread2 = new Thr
        thread2.start();
    }
}
```

```
Messag from Second Thread : 1
Messag from First Thread : 1
Messag from Second Thread : 2
Messag from First Thread : 2
Messag from Second Thread : 3
Messag from First Thread : 3
Messag from Second Thread : 4
Messag from First Thread : 4
Messag from Second Thread : 5
Messag from First Thread : 5
Messag from Second Thread : 6
Messag from First Thread : 6
Messag from Second Thread : 7
Messag from First Thread : 7
Messag from Second Thread : 8
Messag from First Thread : 8
Messag from Second Thread : 9
Messag from First Thread : 9
Messag from Second Thread : 10
Messag from First Thread : 10
```

Khởi tạo thread

Tạo thread bằng cách sử dụng kế thừa class Thread

Phương thức	Ý nghĩa
final String getName()	Lấy ra tên của thread
final int getPriority()	Lấy ra thứ tự ưu tiên của thread
final boolean isAlive()	Kiểm tra 1 thread vẫn còn chạy hay không
final void join()	Chờ đến khi 1 thread ngừng hoạt động
void run()	Chạy một một thread
static void sleep(long milliseconds)	Tạm ngừng hoạt động của 1 thread với một khoảng thời gian là mili giây
void start()	Bắt đầu 1 thread bằng cách gọi run()

Khởi tạo thread

```
class SimpleThread extends Thread {
    public SimpleThread(String str) {
        super(str);
    }
    @Override
    public void run() {
        for (int i = 0; i < 10; i++) {
            System.out.println(i + " " + getName());
        }
        System.out.println("==DONE== " + getName());
    }
    public static void main(String args[]) {
        SimpleThread s1 = new SimpleThread("Tp Ha Noi");
        SimpleThread s2 = new SimpleThread("Tp Ho Chi Minh");
        s1.start();
        s2.start();
    }
}
```

```
0 Tp Ho Chi Minh
0 Tp Ha Noi
1 Tp Ho Chi Minh
1 Tp Ha Noi
2 Tp Ha Noi
2 Tp Ho Chi Minh
3 Tp Ha Noi
3 Tp Ho Chi Minh
4 Tp Ha Noi
4 Tp Ho Chi Minh
5 Tp Ho Chi Minh
5 Tp Ha Noi
6 Tp Ha Noi
6 Tp Ho Chi Minh
7 Tp Ha Noi
7 Tp Ho Chi Minh
8 Tp Ha Noi
8 Tp Ho Chi Minh
9 Tp Ha Noi
9 Tp Ho Chi Minh
==DONE== Tp Ha Noi
==DONE== Tp Ho Chi Minh
```

Khởi tạo thread

Thực hiện 1 công việc bằng nhiều thread, cách 1

```
class Multi3 implements Runnable{
public void run(){
System.out.println("task one");
}

public static void main(String args[]){
Thread t1 =new Thread(new Multi3());//passing annonymous object of Multi3 class
Thread t2 =new Thread(new Multi3());

t1.start();
t2.start();

 }
}
```

```
OUTPUT
task one
task one
```

Khởi tạo thread

Cách 2

```
class Multi extends Thread{
 public void run(){
   System.out.println("task one");
 }
 public static void main(String args[]){
  Multi t1=new Multi();
  Multi t2=new Multi();
  Multi t3=new Multi();

  t1.start();
  t2.start();
  t3.start();
 }
}
```

OUTPUT

```
task one
task one
task one
```

Khởi tạo thread

Thực hiện nhiều công việc bằng nhiều thread

```
class task1 extends Thread {
    public void run() {
        System.out.println("Task 1");
    }
}
class task2 extends Thread {
    public void run() {
        System.out.println("Task 2");
    }
}
public class Mutitask1 {
    public static void main(String[] args) {
        task1 th1 = new task1();
        task2 th2 = new task2();
        th1.start();
        th2.start();
    }
}
```

Khởi tạo thread

Thực hiện nhiều công việc bằng nhiều thread

```
public class Multitask2 {
    public static void main(String[] args) {
        Task1 t1 = new Task1();
        Task2 t2 = new Task2();
        Thread th1 = new Thread(t1);
        Thread th2 = new Thread(t2);
        th1.start();
        th2.start();
    }
}
```

Khởi tạo thread

Sự khác nhau giữa **thực thi** interface Runnable và **kế thừa** từ class Thread

```
class a_ExtendsThread extends Thread {
    private int counter = 0;

    public void run() {
        counter++;
        System.out.println("ExtendsThread: Counter=" + counter);
    }
}
class a_ImplementsRunnable implements Runnable {
    private int counter = 0;

    public void run() {
        counter++;
        System.out.println("Implements Runnable:Counter=" + counter);
    }
}
```

Khởi tạo thread

Sự khác nhau giữa **thực thi** interface Runnable
và **kế thừa** từ class Thread

OUTPUT

```
Implements Runnable:Counter=1
Implements Runnable:Counter=2
ExtendsThread: Counter=1
ExtendsThread: Counter=1
}
```

Quản lý thread

- Thứ tự ưu tiên giữa các tiến trình
- Phương thức join()
- Đồng bộ hóa thread
- Đồng bộ hóa block
- Mối quan hệ giữa các thread
- Hiện tượng dead lock (bế tắc)
- Daemon thread
- Gabage Collection thread
- Phương thức finalize()

Thứ tự ưu tiên thread

Các hằng số biểu thị độ ưu tiên

- NORM_PRIORITY 5
- MAX_PRIORITY 10
- MIN_PRIORITY 1

Giá trị mặc định cho thứ tự ưu tiên

- NORM_PRIORITY

Hai phương thức

- final void setPriority(int p)
- final int getPriority()

Phương thức join()

```
public class MyThread extends Thread {
    private String name;
    public MyThread(String name){
        this.name=name;
    }
    public void run() {
        try {
            System.out.println("\nTrong run() "+name);
            for (int i=0;i<10;i++){
                System.out.print(i+" ");
                Thread.sleep(100);
            }
        } catch (InterruptedException ex) {
            System.out.println("Catch: "+ex);
        }
    }
```

Phương thức join()

```
public static void main(String[] args) {
    try {
        MyThread th1 = new MyThread("Thread 1");
        MyThread th2 = new MyThread("Thread 2");
        th1.start();
        th1.join();
        th2.start();
    } catch (InterruptedException ex) {
        System.out.println(ex);
    }
}
```

Phương thức join()

```
OUTPUT
Trong run() Thread 1
0 1 2 3 4 5 6 7 8 9
Trong run() Thread 2
0 1 2 3 4 5 6 7 8 9
```

Đồng bộ hóa thread

- Đồng bộ hóa chính là việc **sắp xếp thứ tự** các luồng khi truy xuất vào **cùng đối tượng** sao cho không có sự xung đột dữ liệu.
- Để đảm bảo rằng một nguồn **tài nguyên chia sẻ** được sử dụng bởi một thread tại một thời điểm, chúng ta sử dụng **đồng bộ hóa** (synchronization).

Đồng bộ hóa Thread (tiếp)

- Một ‘monitor’- là một công cụ giám sát hỗ trợ cho việc đồng bộ hóa các luồng.
- Tại một thời điểm chỉ có 1 thread được vào ‘monitor’.
- Khi một thread vào được ‘monitor’ thì tất cả các thread khác sẽ phải đợi đến khi thread này ra khỏi ‘monitor’.
- Để đưa một thread vào ‘monitor’, chúng ta phải gọi một phương thức có sử dụng từ khóa synchronized.
- Sau khi thread đang chiếm giữ monitor này kết thúc công việc và thoát khỏi monitor thì luồng tiếp theo mới có thể ‘vào được’ monitor.

Đồng bộ hóa thread (tiếp)

Không sử dụng từ khóa 'synchronized'

```
public class SynchronizedMethod implements Runnable {
      public void run() {
        for (int i = 0; i < 5; i++) {
            System.out.println(Thread.currentThread().getName() + i);
            try {
                Thread.sleep(200);
            } catch (Exception iex) {
                System.out.println(""+iex);
            }
        }
        System.out.println("*Finish "+Thread.currentThread().getName());
    }
```

Đồng bộ hóa thread (tiếp)

Không sử dụng từ khóa 'synchroniz

```
public static void main(String argu[]) {
    SynchronizedMethod m1 = new Synchron
    Thread t1 = new Thread(m1);
    Thread t2 = new Thread(m1);
    Thread t3 = new Thread(m1);
    t1.setName(" Thread 1: ");
    t2.setName(" Thread 2: ");
    t3.setName(" Thread 3: ");
    t1.start();
    t3.start();
    t2.start();
}
```

```
run:
 Thread 1: 0
 Thread 3: 0
 Thread 2: 0
 Thread 1: 1
 Thread 3: 1
 Thread 2: 1
 Thread 1: 2
 Thread 3: 2
 Thread 2: 2
 Thread 1: 3
 Thread 3: 3
 Thread 2: 3
 Thread 1: 4
 Thread 2: 4
 Thread 3: 4
*Finish  Thread 1:
*Finish  Thread 2:
*Finish  Thread 3:
```

Đồng bộ hóa thread (tiếp)

Sử dụng từ khóa 'synchronized'

```
public class SynchronizedMethod implements Runn
    /* here we made run () as  synchronized met
     * by using synchronized key word. */
    public synchronized void run() {
        for (int i = 0; i < 5; i++) {
            System.out.println(Thread.currentT                i);
            try {
                Thread.sleep(200);
            } catch (Exception iex) {
                System.out.println(""+iex);
            }
        }
        System.out.println("*Finish "+Thread.cu                ame());
    }
```

```
run:
 Thread 1: 0
 Thread 1: 1
 Thread 1: 2
 Thread 1: 3
 Thread 1: 4
*Finish  Thread 1:
 Thread 2: 0
 Thread 2: 1
 Thread 2: 2
 Thread 2: 3
 Thread 2: 4
*Finish  Thread 2:
 Thread 3: 0
 Thread 3: 1
 Thread 3: 2
 Thread 3: 3
 Thread 3: 4
*Finish  Thread 3:
```

Đồng bộ hóa block

Đồng bộ hóa một **đoạn code** trong một phương thức của một đối tượng bằng cách sử dụng **synchronized**.

Với việc đồng bộ hóa block, chúng ta có thể **khóa chính xác** đoạn code mình cần.

Đồng bộ hóa block

Đồng bộ hóa method có thể được viết lại bằng đồng bộ hóa block như sau:

```
public void f(){
  synchronized(this){
            ...
    }
 }
```

```
public synchronized void f(){
  .........
    }
```

Mối quan hệ giữa các thead

Java cũng cung cấp cơ chế giao tiếp liên-quá trình bằng cách sử dụng phương thức **wait(), notify()** và **notifyAll().**

Các phương thức **wait(), notify()** and **notifyAll()** chỉ được gọi từ bên trong một phương thức được đồng bộ hóa (synchronized method).

Mối quan hệ giữa các thead

Phương thức **wait()** sẽ đưa thread vào trạng thái 'sleeping'.

Phương thức **notify()** 'đánh thức' thread đầu tiên đang ở trạng thái 'sleeping' bởi vì phương phức **wait()** bị gọi.

Phương thức **notifyAll()** 'đánh thức' tất cả các thread đang ở trạng thái 'sleeping' bởi vì phương thức wait() bị gọi.

Khi tất cả các thread thoát khỏi trạng thái **spleeping**, thread có độ ưu tiên cao nhất sẽ chạy đầu tiên.

Mối quan hệ giữa các thead

notify()

notify() đánh thức thread đầu tiên đang ở trạng thái sleeping vì phương thức wait() bị gọi.

Thread 1

notifyAll()

notifyAll() đánh thức tất cả các thead đang ở trạng thái sleeping vì vì phương thức wait() bị gọi.

Thread 1

Thread 2

Thread 3

Mối quan hệ giữa các thead

Hoạt động của wait() và notify()

```
public class Customer {

    int amount = 1000;

    public synchronized void withdraw(int m) {
        System.out.println("Ban dang rut tien...");
        if (amount < m) {
            System.out.println("Khong du tien de rut !");
            try {
                wait();
            } catch (Exception e) {
                System.out.println(e);
            }
        }
        amount = amount - m;
        System.out.println("Ban da rut tien thanh cong !!!");
    }
```

Mối quan hệ giữa các thead

Hoạt động của wait() và notify()

```
synchronized void deposit(int m) {
    System.out.println("Ban dang nap tien...");
    amount = amount + m;
    System.out.println("Nap tien thanh cong !!!");
    notify();
}
```

Mối quan hệ giữa các thead

Hoạt động của wait() và notify()

```
    public static void main(String[] args) {
        final Customer c = new Customer();
        Thread th1 = new Thread(){
            public void run() {
                c.withdraw(1500);
            }
        };
        th1.start();
        Thread th2 = new Thread(){
            public void run() {
                c.deposit(2000);
            }
        };
        th2.start();
    }
}
```

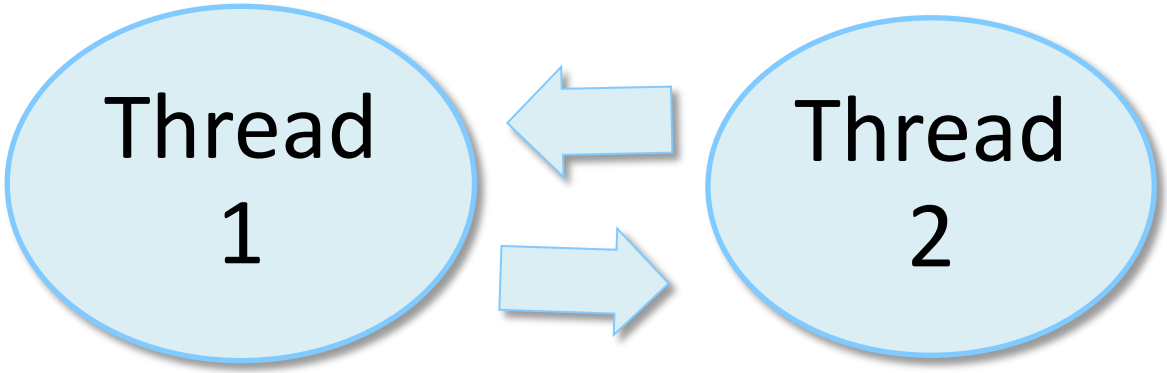
Mối quan hệ giữa các thead

OUTPUT

```
Ban dang rut tien...
Khong du tien de rut !
Ban dang nap tien...
Nap tien thanh cong !!!
Ban da rut tien thanh cong !!!
```

Hiện tượng dead lock

Dead lock: (khóa chết hoặc bế tắc) Là tình huống xảy khi hai hay nhiều tiến trình **chờ đợi lẫn nhau**, tiến trình này chờ tiến trình kia kết thúc công việc thì mới tiếp tục được công việc của mình. Do vậy, các tiến trình này mãi mãi ở trạng thái chờ đợi lẫn nhau (waiting forever).



Daemon threads

Có hai loại thread trong Java:

- Thread người dùng (user thread): Là thread do người dùng tạo ra.
- Daemon threads: Là các thread làm việc ở chế độ nền, cung cấp các dịch vụ cho các thread khác.

Daemon threads

Khi 1 thread của user kết thúc hoạt động, JVM sẽ kiểm tra xem còn thread nào đang chạy không.

Nếu có thì sẽ lên lịch làm việc cho thread tiếp theo.

Nếu chỉ còn các thread ‘daemon’ thì thread này cũng kết thúc hoạt động.

Daemon threads

- Chúng ta có thể thiết lập 1 thread là thread 'daemon' nếu chúng ta không muốn chương trình chính phải đợi đến khi 1 thread kết thúc.
- Class Thread có 2 phương thức làm việc với thread 'Daemon':
  - public final void setDaemon(boolean value)
    Thiết lập 1 thread là thread 'daemon'
  - public final boolean isDaemon()
    Kiểm tra xem thread có phải là ' daemon' không.

Garbage Collection

- Garbage Collection là một trong các thread Daemon (là luồng thu dọn các dữ liệu không dùng đến – dọn rác)
- Garbage Collection sẽ tự động dọn dẹp: giải phóng vùng bộ nhớ không còn cần thiết nữa.
- Một object đủ điều kiện để thu gom nếu không có tham chiếu đến nó hoặc giá trị của nó là null.
- Garbage Collection một thread chạy riêng biệt với độ ưu tiên thấp.

Phương thức finalize ()

- Là phương thức được sử dụng cho việc dọn dẹp các vùng tài nguyên không được dùng nữa trước khi hủy bỏ các đối tượng.
- Sau khi kết thúc chương trình, trước khi trả điều khiển về cho hệ điều hành, phương thức finalize() sẽ được gọi bởi thead 'Gabage collector' để thực hiện công việc dọn dẹp.

Tổng kết bài học

- Khái niệm multitasking và multithreading
- Khái niệm ‘thread’ – luồng
- Thread hiện thời
- Các trạng thái của thread
- Khởi tạo thread

Tổng kết bài học

Quản lý thread

- Thứ tự ưu tiên giữa các tiến trình
- Phương thức join()
- Đồng bộ hóa thread
- Đồng bộ hóa block
- Mối quan hệ giữa các thread
- Hiện tượng dead lock (bế tắc)
- Daemon thread (luồng hiểm)
- Gabage Collection thread
- Phương thức finalize()