FPT Education

FPT POLYTECHNIC

LẬP TRÌNH JAVA 2

BÀI 6: ĐA TIẾN TRÌNH

PHẦN 1

www.poly.edu.vn

Mục tiêu

- Giải thích multitasking và multithreading
- Giải thích 'thread'
- Tạo thread
- Sử dụng các thông tin của thread
- Giải thích các trạng thái của thread
- Đồng bộ hóa tài nguyên dùng chung
- Giải thích được thread Deamon
- Sử dụng được finallize()

KHÁI NIỆM MULTITASKING VÀ MULTITHREADING

- Multitasking: Là khả năng chạy đồng thời nhiều chương trình cùng một lúc trên hệ điều hành.
  - Internet Explorer
  - Microsoft Excel
  - Window Media Player
- Multithreading: Là khả năng thực hiện đồng thời nhiều tiểu trình trong một chương trình.
  - 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.

TẠO THREAD

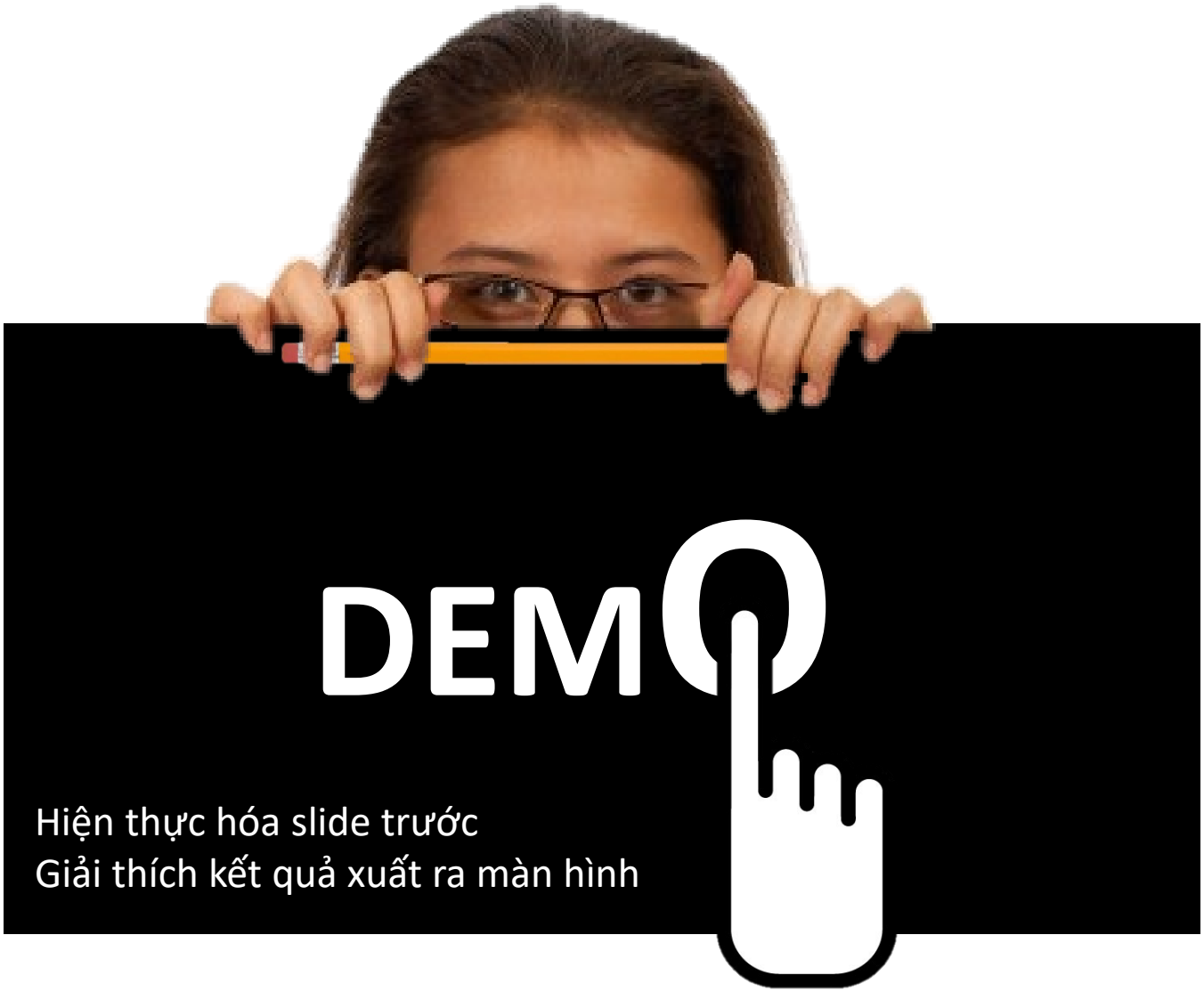
- Hệ thống xử lý đa luồng trong Java được xây dựng trên class Thread và interface Runnable trong package java.lang.

CÁCH 1: TẠO THREAD

```
public class MyThread extends Thread{
    @Override
    public void run() {
        for(int i=0; true; i++){
            System.out.println(i);
            try {
                Thread.sleep(1000);
            }
            catch (InterruptedException e) {
                break;
            }
        }
    }
}
```

3 tiểu trình đang chạy song song

```
public static void main(String[] args) {
    MyThread t1 = new MyThread();
    t1.start();
    MyThread t2 = new MyThread();
    t2.start();
}
```


DEMO
Hiện thực hóa slide trước
Giải thích kết quả xuất ra màn hình

CÁCH 2: TẠO THREAD

```
public class MyRunnable implements Runnable{
    @Override
    public void run() {
        for(int i=0; true; i++){
            System.out.println(i);
            try {
                Thread.sleep(1000);
            }
            catch (InterruptedException e) {
                break;
            }
        }
    }
}
```

3 tiểu trình đang chạy đang chạy song song

```
public static void main(String[] args) {
    Thread t1 = new Thread(new MyRunnable());
    t1.start();
    Thread t2 = new Thread(new MyRunnable());
    t2.start();
}
```

DEMO

```
public class ThreadDemo
        extends javax.swing.JFrame implements Runnable {
        ...
        private void btnT1ActionPerformed(java.awt.event.ActionEvent  evt) {
                // TODO add your handling code here:
                Thread t1 = new Thread(this);
                t1.start();
                btnT1.setEnabled(false);
        }
        ...
        int count = 0;
        @Override
        public void run() {
                while (true) {
                        try {
                                btnT1.setText("" + count);
                                Thread.sleep(10);
                                count++;
                        } catch (Exception e) {
                                break;
                        }
                }
        }
}
```

TẠO THREAD VỚI KỸ THUẬT LỚP NẶC DANH

- Nặc danh với Thread

```
new Thread(){
    public void run(){}
}.strart();
```

- Nặc danh với Runnable

```
new Thread(new Runnable(){
    public void run(){}
}).strart();
```

DEMO


```
Thread t1 = new Thread(){
    int count = 0;
    @Override
    public void run() {
        while(true){
            try {
                btnT1.setText(""+count);
                Thread.sleep(10);
                count++;
            }
            catch (Exception e) {
                break;
            }
        }
    }
};
t1.start();
btnT1.setEnabled(false);
```

SỰ KHÁC NHAU GIỮA 2 CÁCH

- Trong Java 1 lớp chỉ được kế thừa duy nhất một lớp khác. Nếu một lớp đã kế thừa một lớp khác rồi thì phải sử dụng cách 2.

```
public class StudentList extends ArrayList<Student> implements Runnable{
    @Override
    public void run() {...}
}
```

```
public static void main(String[] args) {
    Thread t = new Thread(new StudentList());
    t.start();
}
```

- Nhiều thread có thể dùng chung một Runnable

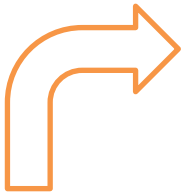
```
class MyRun implements Runnable{}
```

→

```
MyRun run = new MyRun()
Thread t1 = new Thread(run);
Thread t2 = new Thread(run);
```

THÔNG TIN THREAD

- Thread có rất nhiều thông tin hữu ích cần biết



```
Định danh: 1
Tên: main
Độ ưu tiên: 5
Trạng thái: RUNNABLE
Đang hoạt động: true
Số lượng thread: 2
0
1
2
```

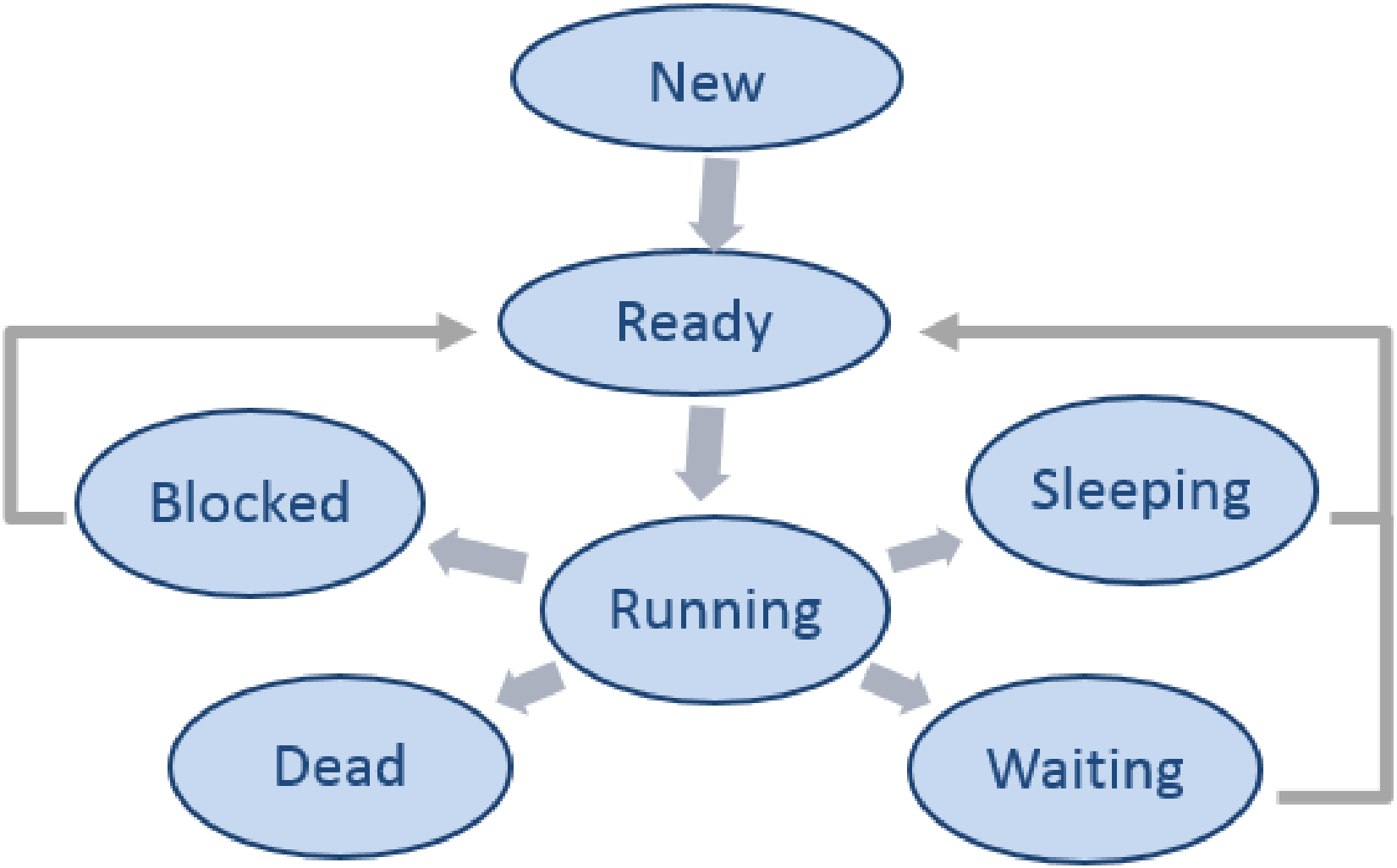
```
public static void main(String[] args) {
        MyThread t1 = new MyThread();
        t1.start();

        Thread t2 = Thread.currentThread();
        System.out.println("Định danh: " + t2.getId());
        System.out.println("Tên: " + t2.getName());
        System.out.println("Độ ưu tiên: " + t2.getPriority());
        System.out.println("Trạng thái: " + t2.getState());
        System.out.println("Đang hoạt động: " + t2.isAlive());
        System.out.println("Số lượng thread: " + Thread.activeCount());
}
```

THREAD API

- 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()

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)

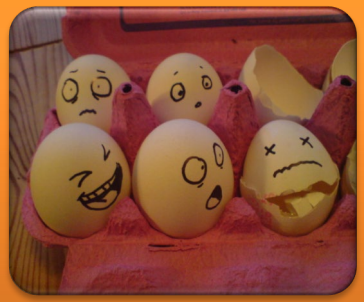
Sleeping: Phương thức **sleep()** sẽ đưa thead vào trạng thái 'sleeping' - dừng lại tạm thời. Sau thời gian 'sleeping' thread lại tiếp tục hoạt động.

FPT POLYTECHNIC

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().**

FPT Education

FPT POLYTECHNIC

LẬP TRÌNH JAVA 2

BÀI 6: ĐA TIẾN TRÌNH

PHẦN 2

www.poly.edu.vn

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
- Đọc/ghi độ ưu tiên của Thread
  - final void setPriority(int p)
  - final int getPriority()
- Trong trường hợp xảy ra tranh chấp tài nguyên thì thread có độ ưu tiên cao hơn sẽ thực hiện

PHƯƠNG THỨC JOIN()

- Join() được sử dụng để đợi một thread nào đó kết thúc

```
MyThread t1 = new MyThread();
MyThread t2 = new MyThread();
   t1.start();
   t1.join();
   t2.start();
```

- Chương trình trên có 3 thread đang hoạt động
- Thread hiện tại gọi t1.join() có nghĩa là thread hiện tại phải đợi t1 kết thúc mới được chạy tiếp vì vậy t2.start() sẽ không chạy khi nào t1 chưa kết thúc

ĐỒNG BỘ HÓA THREAD

- Nếu nhiều thread đang hoạt động đồng thời mà sử dụng chung một tài nguyên nào đó thì sẽ xảy ra xung đột
- Đồng bộ hóa chính là việc sắp xếp thứ tự các thread 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

- Khi nhiều thread cùng gọi một phương thức được khai báo với synchronized thì cái gọi sau sẽ phải đợi

```
public class MyRunable implements Runnable{
    @Override
    public void run() {...}
}
```

→ t1 và t2 chạy đồng thời

```
MyRunable run = new MyRunable();
Thread t1 = new Thread(run);
Thread t2 = new Thread(run);
t1.start();
t2.start();
```

Cả 2 thread t1 và t2 dùng chung run

```
public class MyRunable implements Runnable{
    @Override
    public synchronized void run() {...}
}
```

→ t1 chạy xong mới đến t2

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

```
public synchronized void run(){
        ……
}
```

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

```
public void run(){
    synchronized(this){
            …
    }
}
```

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() trên đối tượng chia sẻ.
- Các phương thức wait(), notify() and notifyAll() chỉ được gọi từ bên trong một phương thức hoặc block 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.

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 !!!");
    }
```

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();
}
```

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();
    }
}
```

2 thread th1 và th2 dùng chung c

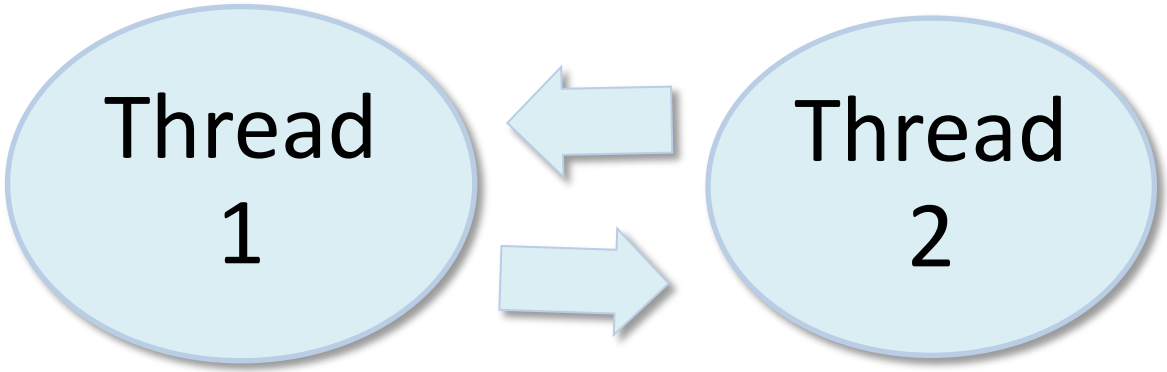
HOẠT ĐỘNG CỦA WAIT() VÀ NOTIFY()

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 tiểu 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 NỘI DUNG BÀI HỌC

- ☑ Giải thích multitasking và multithreading
- ☑ Giải thích 'thread'
- ☑ Tạo thread
- ☑ Sử dụng các thông tin của thread
- ☑ Giải thích các trạng thái của thread
- ☑ Đồng bộ hóa tài nguyên dùng chung
- ☑ Giải thích được thread Deamon
- ☑ Sử dụng được finallize()