CÔNG NGHỆ JAVA

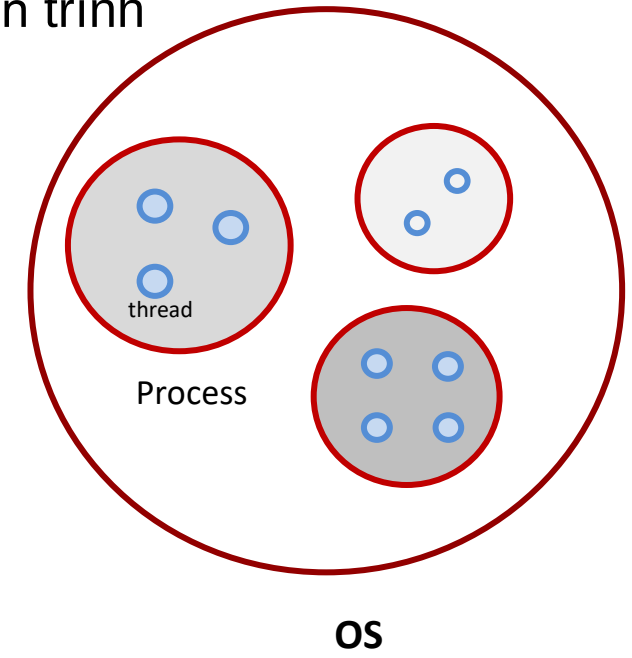
KHÁI NIỆM VỀ THREAD

Nội dung

- Khái niệm về luồng
- Trạng thái luồng
- Đồng bộ luồng

Luồng và đa luồng

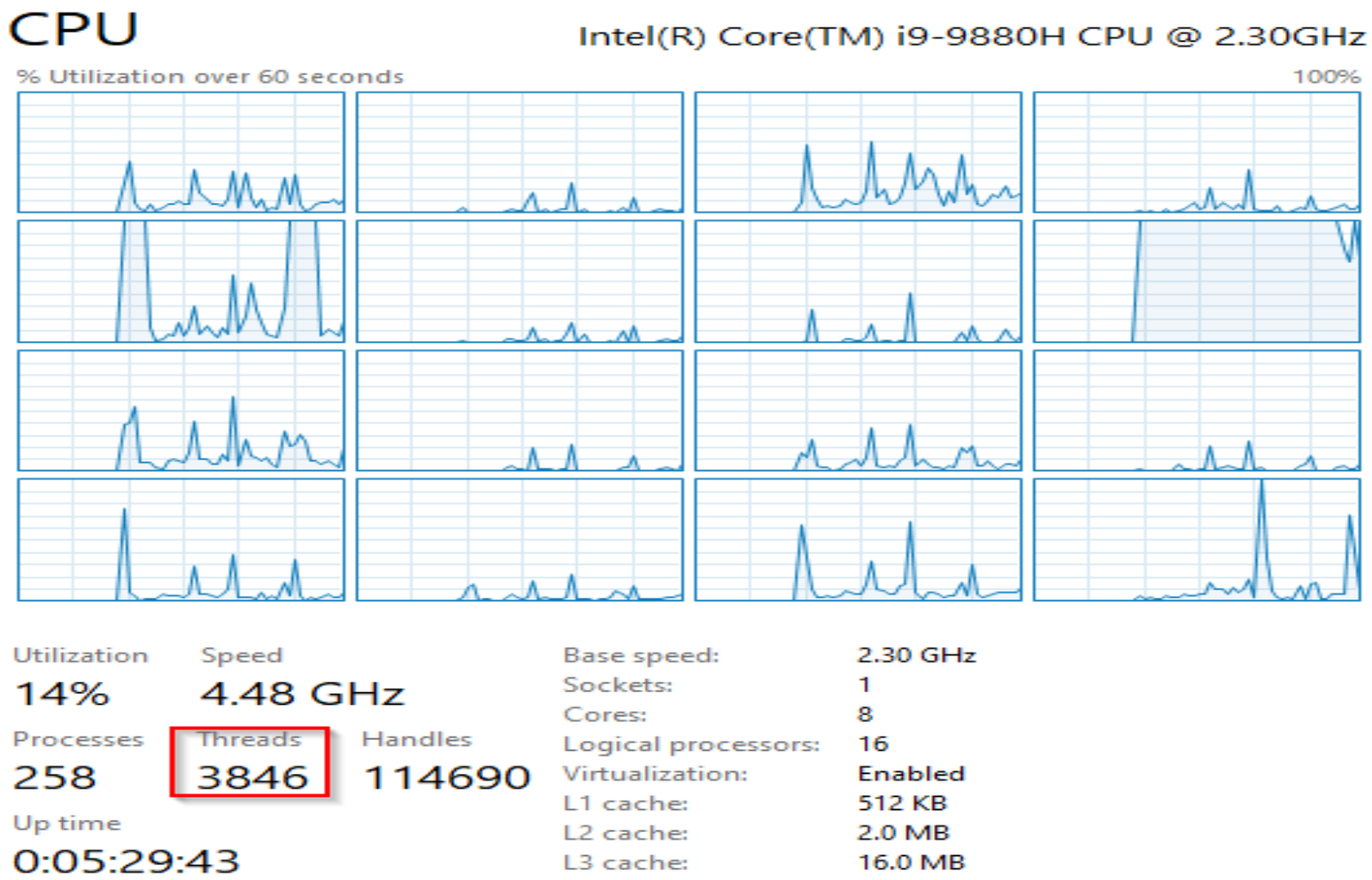
- Luồng - thread: Một dòng các lệnh mà CPU phải thực thi.
- Các hệ điều hành mới cho phép nhiều luồng được thực thi đồng thời → Nhiều ứng dụng chạy song song
- Như vậy
  - Một luồng là một chuỗi các lệnh nằm trong bộ nhớ.
  - 1 application thông thường tương ứng 1 tiến trình (process)
  - 1 process có thể có nhiều luồng



Kỹ thuật đa luồng

- Các máy tính hiện đại có thể chạy 1 lúc nhiều luồng
- Số luồng vật lý thường nhỏ hơn nhiều số luồng của hệ điều hành
- Các luồng được quản lý bằng 1 hàng đợi:
  - Mỗi luồng được cấp phát thời gian mà CPU thực thi là *ti* (cơ chế time-slicing – phân chia tài nguyên thời gian).
  - Luồng ở đỉnh hàng đợi được lấy ra để thực thi trước
  - Sau *ti* thời gian của mình, luồng này được đưa vào cuối hàng đợi và CPU lấy ra luồng kế tiếp.

Kỹ thuật đa luồng



Lợi ích của đa luồng

- Tăng hiệu suất sử dụng CPU: Phần lớn thời gian thực thi của 1 ứng dụng là chờ đợi nhập liệu từ user → hiệu suất sử dụng CPU chưa hiệu quả.
- Tạo được sự đồng bộ giữa các đối tượng

  Ví dụ: Trong 1 game thường có nhiều luồng xử lý hình ảnh, âm thanh và logic
- Quản lý được thời gian trong các ứng dụng như thi online, thời gian chơi một trò chơi.

Luồng trong Java

- **Main thread** - luồng chính : là luồng chứa các luồng khác. Đây chính là luồng cho Java. Application hiện hành (mức toàn application).
- **Child thread** - luồng con : là luồng được tạo ra từ luồng khác.
- Khi 1 application thực thi, main thread được chạy, khi gặp các phát biểu phát sinh luồng con, các luồng con được khởi tạo. Vào thời điểm luồng chính kết thúc, application kết thúc.

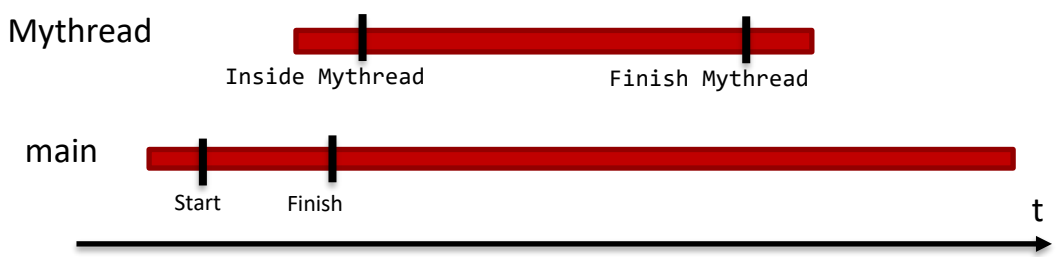
Lập trình luồng trong Java

- Cách 1: Xây dựng 1 lớp con của lớp java.lang.Thread, override hành vi run() để phù hợp với mục đích bài toán.
- Cách 2: Xây dựng 1 lớp có hiện thực interface Runnable và sử dụng cùng với lớp Thread.
- Chú ý:
  - Không cần import java.lang vì là gói cơ bản
  - java.lang.Thread là lớp Java xây dựng sẵn đã hiện thực interface Runnable
  - Interface java.lang.Runnable chỉ có 1 method run()

Tạo luồng

```
package create_thread_demo;
public class Mythread extends Thread{
    public void run() {
        try {
            System.out.println("Inside Mythread");
            Thread.sleep(2000);
                System.out.println("Finish Mythread");
        } catch (InterruptedException e) {
                e.printStackTrace();
        }
    }
    public static void main(String[] args) throws InterruptedException {
        System.out.println("Start");
        Mythread t1 = new Mythread();
        t1.start();
        System.out.println("Finish");
    }
}
```

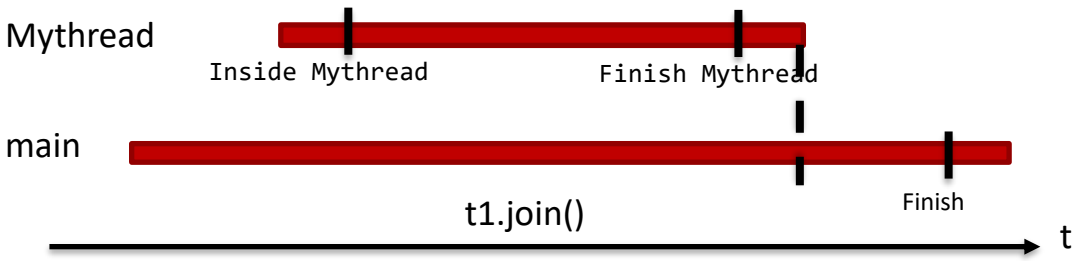
```
Start
Finish
Inside Mythread
Finish Mythread
```



Tạo luồng

```
package create_thread_demo;

public class Mythread extends Thread{
     public void run() {
          try {
               System.out.println("Inside Mythread");
               Thread.sleep(2000);
                    System.out.println("Finish Mythread");
          } catch (InterruptedException e) {
                    e.printStackTrace();
          }
     }

          public static void main(String[] args) throws InterruptedException {
          Mythread t1 = new Mythread();
          t1.start();
          t1.join();
          System.out.println("Finish");
     }
}
```

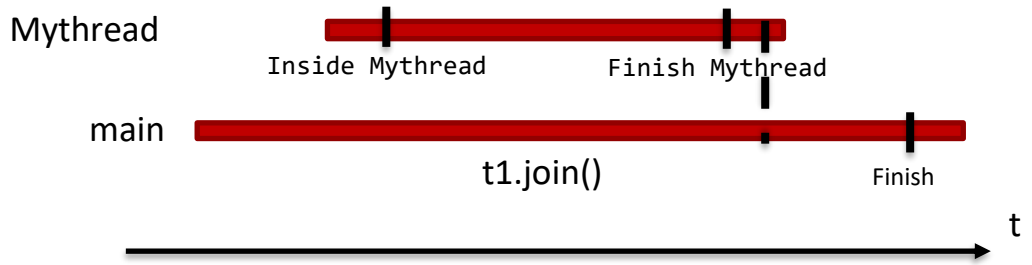
```
Inside Mythread
Finish Mythread
Finish
```



Tạo luồng

```
package create_thread_demo;

public class Mythread implements Runnable{
     public void run() {
          try {
               System.out.println("Inside Mythread");
               Thread.sleep(2000);
               System.out.println("Finish Mythread");
          } catch (InterruptedException e) {
                    e.printStackTrace();
          }
     }

     public static void main(String[] args) throws InterruptedException {
          Mythread mt1 = new Mythread();
          Thread thread = new Thread(mt1);
          thread.start();
          Thread.join();
          System.out.println("Finish");
     }
}
```

```
Inside Mythread
Finish Mythread
Finish
```



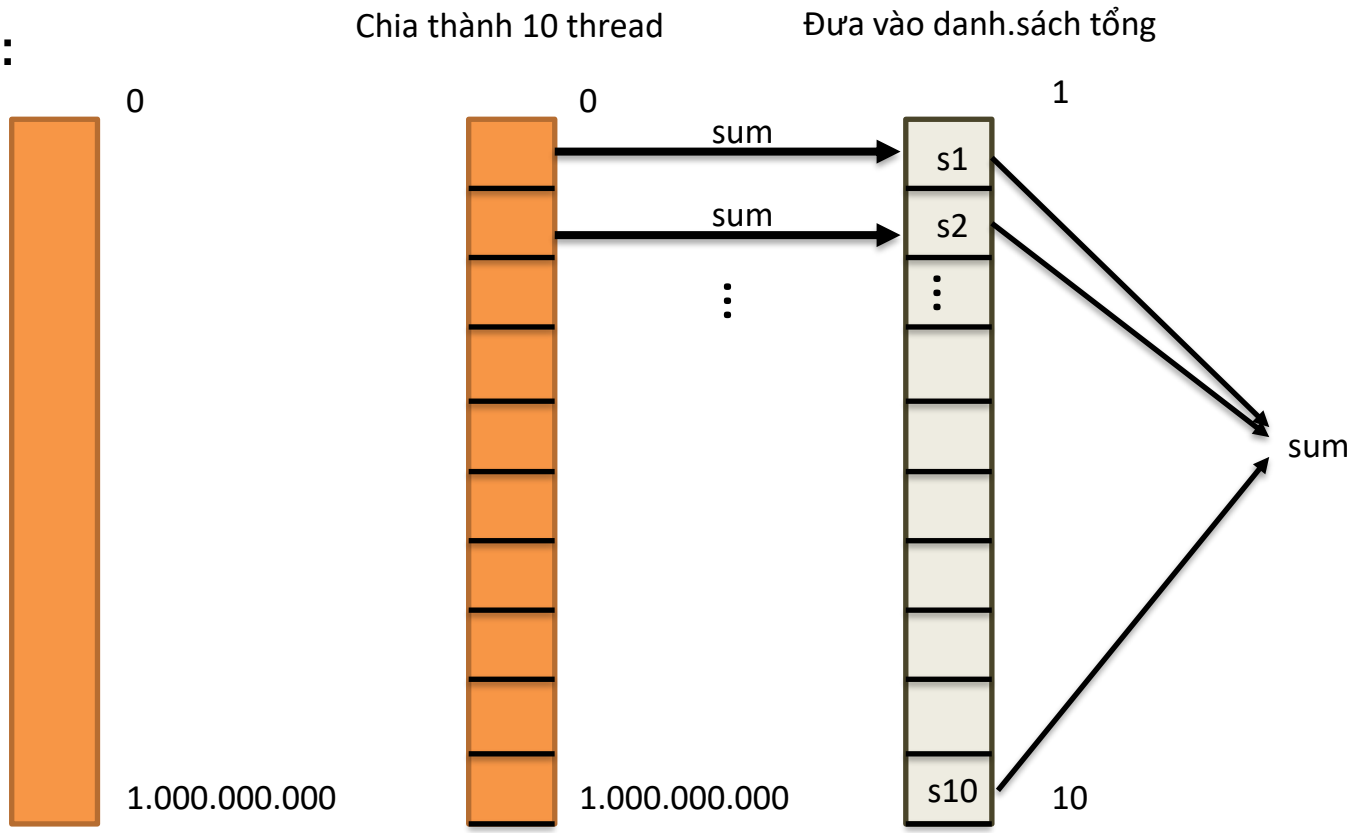
Bài tập

- Hãy chạy đoạn chương trình tính tổng các số từ 1 - 1.000.000.000 và ghi nhận thời gian hoàn thành
- Hãy dùng kỹ thuật đa luồng để tăng tốc độ tính toán (10 luồng)

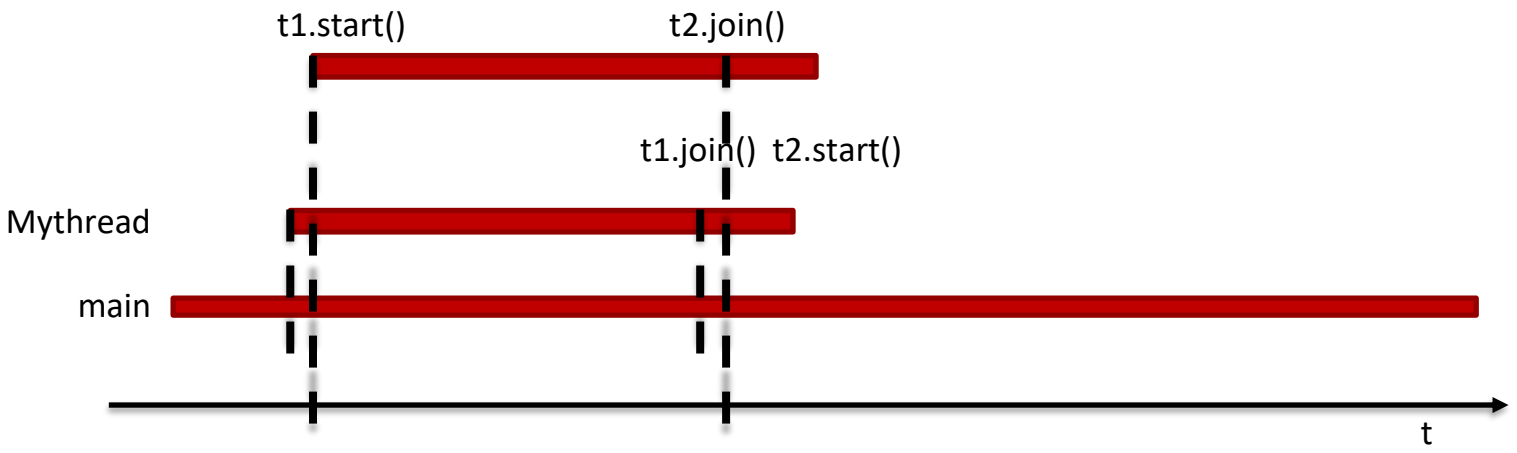
```
public class Calculator{
    public long sum() {
        long sum = 0;
        for (int i = 0; i < 1000000000; i++) {
                sum += i;
        }
        return sum;
    }

    public static void main(String[] args) {
        Calculator cal = new Calculator();
        long start = System.currentTimeMillis();
        long sum = cal.sum();
        long end = System.currentTimeMillis();
        System.out.println("sum: " + sum);
        System.out.println("Time is ms: " + (end - start));
    }
}
```

Bài tập

- **Gợi ý:**

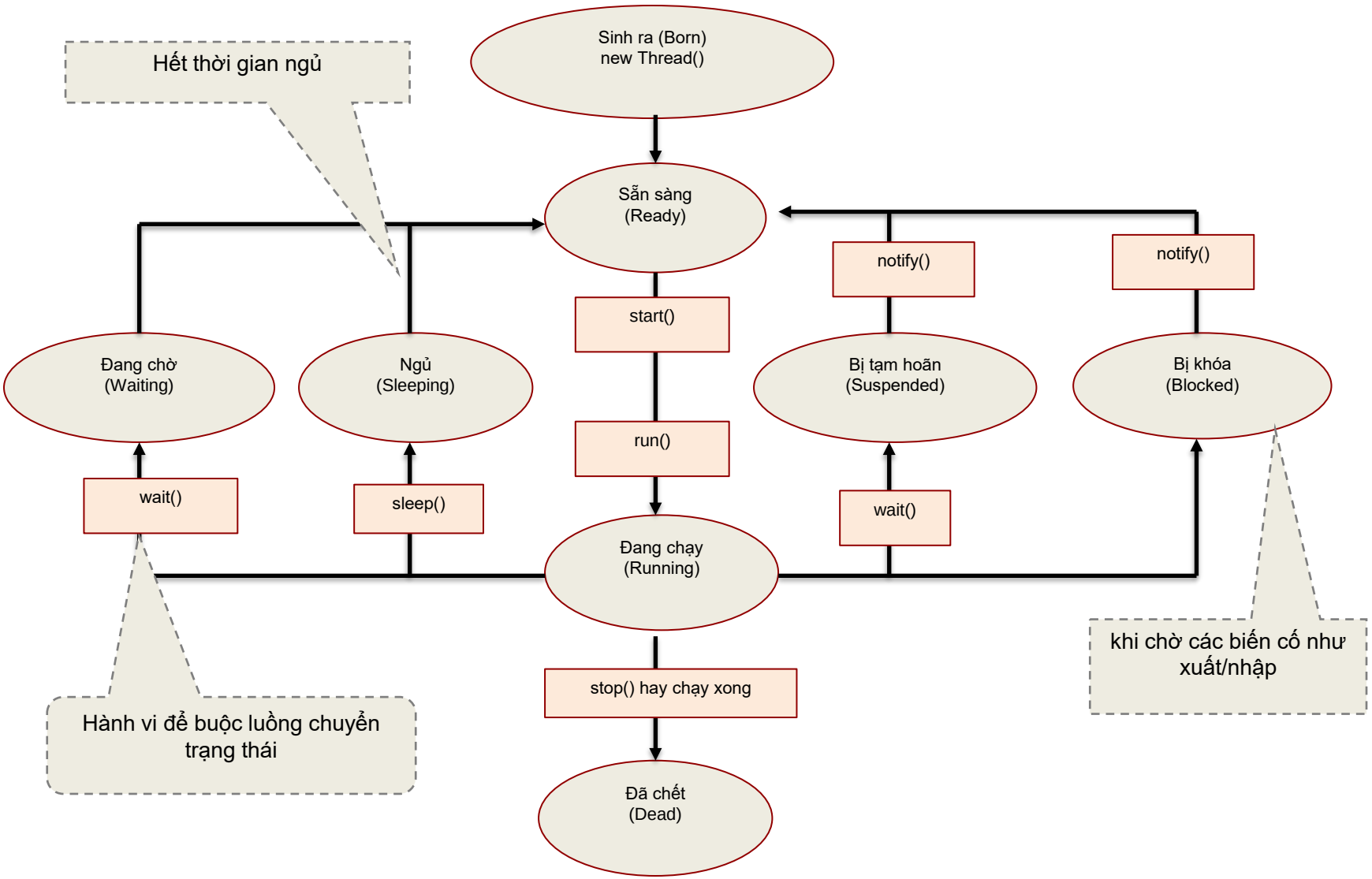

Chia thành 10 thread
Đưa vào danh.sách tổng
0
1.000.000.000
0
1.000.000.000
sum
sum
1
s1
s2
s10
10
sum

Bài tập



TRẠNG THÁI CỦA LUỒNG

Trạng thái của luồng



Trạng thái của luồng

- Luồng sau khi sinh ra (born) không được chạy ngay mà chỉ là sẵn sàng (ready) chạy. Chỉ khi nào phương thức start() được gọi thì luồng mới thực thi (chạy code phương thức run()).
- Luồng đang thực thi có thể bị tạm ngưng bằng phương thức sleep() một thời khoảng và sẽ lại ready sau khi đáo hạn thời gian. Luồng đang ngủ không sử dụng tài nguyên CPU.
- Khi nhiều luồng cùng được thực thi, nếu có 1 luồng giữ tài nguyên mà không nhả ra sẽ làm cho các luồng khác không dùng được tài nguyên này (đói tài nguyên). Để tránh tình huống này, Java cung cấp cơ chế Wait-Notify(đợi-nhận biết). Phương thức wait() giúp đưa 1 luồng vào trạng thái chờ.

Trạng thái của luồng

- Khi một luồng bị tạm ngưng hay bị treo, luồng rơi vào trạng thái tạm hoãn (suspended). Phương thức wait() dùng cho mục đích này.
- Khi 1 suspended thread được mang ra thực thi tiếp, trạng thái của luồng là resumed. Phương thức notify() được dùng cho mục đích này.
- Khi 1 luồng chờ biến cố như xuất/nhập dữ liệu. Luồng rơi vào trạng thái blocked.
- Khi 1 luồng thực thi xong phương thức run() hay gặp phương thức stop(), ta nói luồng đã chết (dead).

Methods thông dụng của lớp Thread

Method	Mục đích
`final String getName()`	Lấy tên của luồng
`final boolean isAlive()`	Kiểm tra luồng còn sống hay không?
`Final void setName( String NewName)`	Đặt tên mới cho luồng
`final void join () throws interruptedException`	Chờ luồng này chết
`public final boolean isDaemon()`	Kiểm tra xem luồng này có phải là luồng daemon
`static void sleep (long milisec)`	Trì hoãn luồng 1 thời gian
`void start()`	thực thi luồng
`static int activeCount()`	Đếm số luồng đang tích cực
`static void yield()`	Tạm dừng luồng hiện hành để các luồng khác tiếp tục thực thi

Hai loại luồng

- **Luồng Daemon**: luồng hệ thống, chạy ở mức nền (background - chạy ngầm), là những luồng cung cấp các dịch vụ cho các luồng khác. Các quá trình trong JVM chỉ tồn tại khi các luồng daemon tồn tại. JVM có ít nhất 1 luồng daemon là luồng “garbage collection”
- **Luồng do user tạo ra**

Ví dụ 1 số phương thức của Thread

```
package create_thread_demo;

public class Mythread extends Thread{
    public void run() {
        try {
            System.out.println("Inside " + this.getName());
            Thread.sleep(2000);
        System.out.println("Finish " + this.getName());
        } catch (InterruptedException e) {
                e.printStackTrace();
    }
}
public static void main(String[] args) throws InterruptedException {
        Mythread t1 = new Mythread();
        t1.setName("test thread 1");
        t1.start();
        Mythread t2 = new Mythread();
        t2.setName("test thread 2");
        t2.start();
        System.out.println("Thread count: " + Thread.activeCount());
        t1.join();
        t2.join();
        System.out.println("Finish");
    }
}
```

```
Thread count: 3
Inside test thread 2
Inside test thread 1
Finish test thread 2
Finish test thread 1
Finish
```

Độ ưu tiên của luồng

- **Các luồng cùng chia sẻ thời gian của CPU.**
  - **Luồng ở cuối hàng đợi sẽ lâu được CPU thực thi**
  - **Có nhu cầu thay đổi độ ưu tiên của luồng.**
  - **Java cung cấp 3 hằng mô tả độ ưu tiên của 1 luồng (các độ ưu tiên khác dùng 1 số nguyên từ 1.. 10).**
- **NORM_PRIORITY : mang trị 5**
- **MAX_PRIORITY : mang trị 10**
- **MIN_PRIORITY : mang trị 1**
- **Độ ưu tiên mặc định của 1 luồng là NORMAL_PRIORITY. Luồng con có cùng độ ưu tiên với luồng cha (do đặc điểm thừa kế).**

Độ ưu tiên của luồng

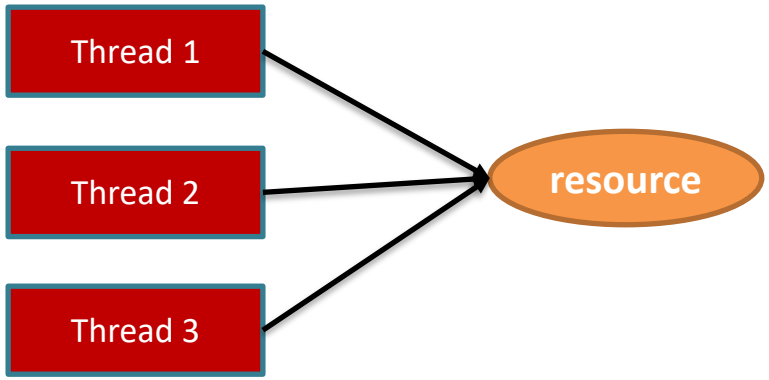
- **final void setPriority(int newPriority)**
- **final int   getPriority()**

- **Như vậy, luồng không được thực thi khi:**
  - **Luồng không có được độ ưu tiên cao nhất để dành lấy thời gian của CPU**
  - **Luồng bị cưỡng bức ngủ bằng hành vi sleep()**
  - **Luồng bị chờ do hành vi wait()**
  - **Luồng bị khóa vì đang chờ I/O**

ĐỒNG BỘ GIỮA CÁC LUỒNG

Khái niệm

- **Đồng bộ giữa các luồng nhằm kiểm soát truy xuất vào tài nguyên chung**
  - **Điều khiển thứ tự truy xuất**
  - **Điều khiển truy xuất đồng thời: Chỉ cho 1 luồng truy xuất vào 1 thời điểm**
- **Các phương pháp đồng bộ giữa các luồng:**
  - **Đồng bộ phương thức**
  - **Đồng bộ khối**



Đồng bộ theo synchronize method

```
// Table.java
class Table {
     void printTable(int n) {// method not synchronized
          for (int i = 1; i <= 5; i++) {
                    System.out.println(n * i);
               try {Thread.sleep(400);}
               catch (Exception e) {System.out.println(e);}
               }}}

// MyThread1.java
public class MyThread1 extends Thread {
     Table t;
     private int multiplier;
     MyThread1(Table t, int multiplier) {
          this.t = t;
          this.multiplier = multiplier;
     }
     public void run() {
          t.printTable(multiplier);
     }
     public static void main(String args[]) {
          Table obj = new Table();// only one object
          MyThread1 t1 = new MyThread1(obj, 5);
          MyThread1 t2 = new MyThread1(obj, 20);
          t1.start();
          t2.start();
     }}
```

```
5
20
40
10
60
15
20
80
25
100
```

Đồng bộ theo khối

- Đồng bộ một khối tác vụ.
- Người lập trình có thể không muốn dùng các **synchronized** method để đồng bộ truy xuất đến đối tượng.

Đồng bộ theo synchronize method

```
// Table.java
class Table {
    void printTable(int n) {
        synchronized(this){
            for (int i = 1; i <= 5; i++) {
                System.out.println(n * i);
                try {
                        Thread.sleep(400);
                } catch (Exception e) {
                        System.out.println(e);
                }}}}}
```

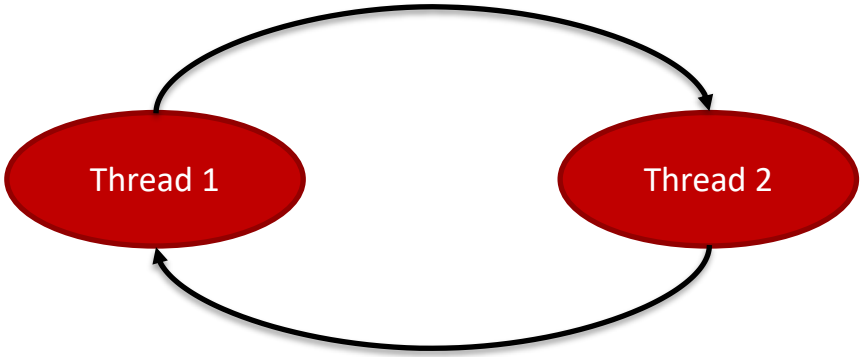
Đồng bộ theo synchronize method

```
// MyThread1.java
public class MyThread1 extends Thread {
    Table t;
    private int multiplier;

    MyThread1(Table t, int multiplier) {
        this.t = t;
        this.multiplier = multiplier;
    }
    public void run() {
        t.printTable(multiplier);
    }

    public static void main(String args[]) {
        Table obj = new Table();// only one object
        MyThread1 t1 = new MyThread1(obj, 5);
        MyThread1 t2 = new MyThread1(obj, 20);
        t1.start();
        t2.start();
    }
}
```

```
5
10
15
20
25
20
40
60
80
100
```

Deadlock

- **Deadlock** – tình huống bế tắc, đóng băng- xẩy ra khi các luồng chờ tài nguyên của nhau hình thành một chu trình.
  - Ví dụ: Thread 1 đang đợi tài nguyên của thread 2, trong khi thread 2 đợi tài nguyên của thread 1



Deadlock

```
public class TestDeadlockExample1 {
   public static void main(String[] args) {
     final String resource1 = "test1";
     final String resource2 = "test2";

// t1 tries to lock resource1 then resource2
  Thread t1 = new Thread() {
    public void run() {
      synchronized (resource1) {
         System.out.println("Thread 1: locked
  resource 1");
       try {
         Thread.sleep(100);
       } catch (Exception e) {
       }

       synchronized (resource2) {
         System.out.println("Thread 1: locked
       resource 2");
  }}}};
```

```
  // t2 tries to lock resource2 then
  resource1

  Thread t2 = new Thread() {
    public void run() {
    synchronized (resource2) {
       System.out.println("Thread 2:
  locked resource 2");
       try {
       Thread.sleep(100);
       } catch (Exception e) {
       }
        synchronized (resource1) {
       System.out.println("Thread 2:
       locked resource 1");
       }
  }}};
  t1.start();
  t2.start();
  }
}
```

```
Thread 1: locked resource 1
Thread 2: locked resource 2
```

Tóm tắt

- Luồng là biện pháp chia công việc thành các đơn vị cụ thể nên có thể được dùng để thay thế vòng lặp.
- Java cung cấp kỹ thuật lập trình đa luồng bằng lớp Thread và interface Runnable.
- Lập trình đa luồng làm tăng hiệu suất sử dụng CPU trên những hệ thống "bận rộn".
- Chương trình đa luồng thường phức tạp và khó debug
- Khi 1 ứng dụng Java thực thi, có 1 luồng đang chạy đó là luồng chính (main thread). Luồng chính rất quan trọng vì (1) Đây là luồng có thể sinh ra các luồng con, (2) Quản lý việc kết thúc ứng dụng vì luồng main chỉ kết thúc khi tất cả các luồng con của nó đã kết thúc.

Tóm tắt

- Hiện thực 1 luồng bằng 1 trong 2 cách:
  - Hiện thực 1 lớp con của lớp Thread, override phương thức run() của lớp này.
  - Khai báo lớp mà ta xây dựng là implement của interface Runnable và định nghĩa phương thức run().
- Mỗi java thread có 1 độ ưu tiên từ 1 (MIN) đến 10 (MAX) với 5 là trị mặc định. JVM không bao giờ thay đổi độ ưu tiên của luồng.
- Luồng daemon là luồng chạy ngầm nhằm cung cấp dịch vụ cho các luồng khác.

Tóm tắt

- Dữ liệu có thể bị mất nhất quán(hư hỏng) khi có 2 luồng cùng truy xuất dữ liệu tại cùng 1 thời điểm.
- Đồng bộ là 1 quá trình bảo đảm tài nguyên (dữ liệu, file,…) chỉ được 1 luồng sử dụng tại 1 thời điểm.
- Deadlock xẩy ra khi 2 luồng có sự phụ thuộc vòng trên một cặp đối tượng quản lý việc đồng bộ (synchronized object).

Tài liệu tham khảo

- https://www.javatpoint.com/synchronization-in-java
- https://www.javatpoint.com/creating-thread