LẬP TRÌNH MẠNG

Thread và Multi-thread

hungdn@ptit.edu.vn

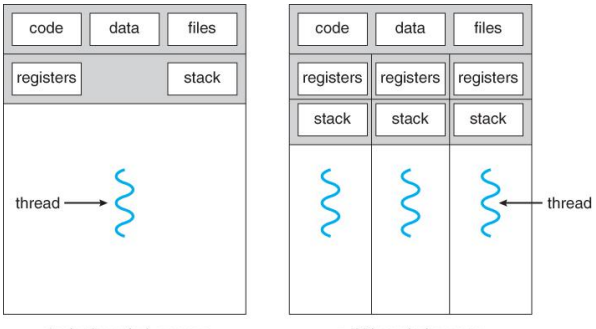
Nội dung

- Tổng quan về thread
  - Vòng đời của thread
- Làm việc với thread
- Đồng bộ thread
- Liên lạc giữa các thread

1. Tổng quan thread

Tổng quan thread

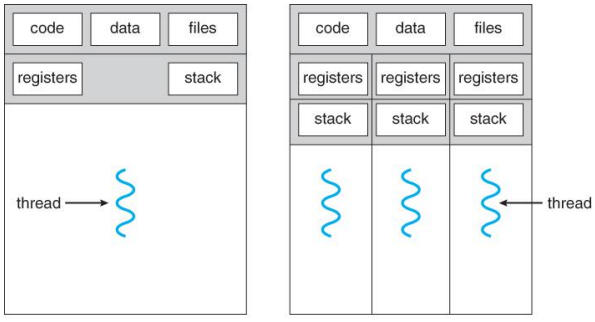
- **Thread là một tiến trình con (sub-process).**
  - Là một đơn vị xử lý nhỏ nhất của máy tính có thể thực hiện một công việc riêng biệt
  - Luồng trong Java được quản lý bởi máy ảo Java
- **Single thread vs Multi thread**
- **Ex: web browser**
  - Tab
  - Download
  - Video/audio
  - ...



single-threaded process multithreaded process

Multi thread

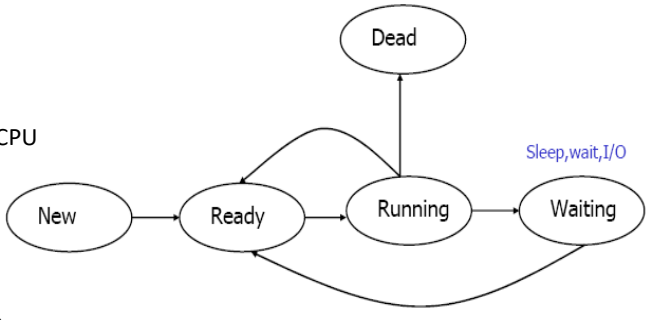
- **Đa nhiệm**
  - Tiến trình -> đa tiến trình
  - Thread -> đa luồng
- **Vấn đề**
  - Đồng bộ, tranh chấp tài nguyên
  - Trường hợp dead lock



single-threaded process multithreaded process

Vòng đời của một Thread

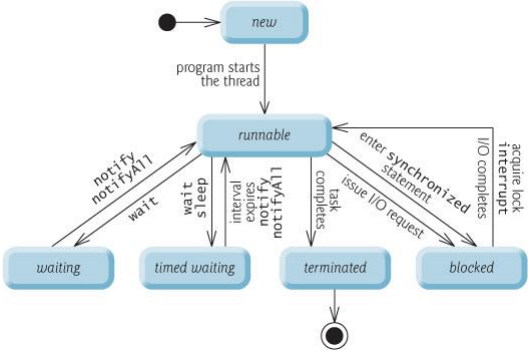
- **New**
  - Tiến trình mới được khởi tạo
- **Ready to run**
  - Sẵn sàng để chạy nếu được cấp tài nguyên
- **Running**
  - Lệnh của tiến trình đang được thực hiện bởi CPU
- **Resumed**
  - Quay trở lại trạng lại sẵn sàng để chạy sau khi Suspended hoặc blocked
- **Suspended**
  - Tự nguyện cho phép các luồng khác thực hiện
- **Blocked**
  - Chờ đợi một tài nguyên hoặc sự kiện nào đó xảy ra



Vòng đời một thread trong Java

Trạng thái của luồng (Thread.State)

- **New:** được khởi tạo và chưa được start()
- **Runable**: được start() và bắt đầu được cấp phát tài nguyên dưới dự kiểm soát của CPU
- **Blocked**: chờ một tài nguyên nào đó hoặc chưa được chọn để chạy (đồng bộ)
- **Waiting**: chờ không giới hạn thời gian cho đến khi một luồng khác đánh thức nó
- **Timed Waiting**: chờ một khoảng thời gian hoặc một luồng khác đánh thức
- **Terminated**: khi công việc hoàn thành hoặc khi phương thức run() bị thoát



Thread trong Java

- Java chia Thread làm 02 loại
  - Thread thông thường
  - Deamon Thread
- Khác nhau ở cách thức ngừng hoạt động
  - Thread thông thường kết thúc khi hoàn thành công việc
  - Deamon Thread kết thúc khi tất cả các luồng thông thường kết thúc, dù có đang thực hiện công việc (runable)
- Để thiết lập một luồng là Deamon Thread
  - `setDeamon(boolean)`
  - Lưu ý thiết lập trước khi `start()` Thread
- Độ ưu tiên
  - `Thread.MIN_PRIORITY (1)`
  - `Thread.MAX_PRIORITY (10)`

2. Làm việc với Thread

Tạo thread (1)

Kế thừa lớp `java.lang.Thread`

1. Khai báo lớp kế thừa lớp `Thread`
2. Cài đặt (`override`) phương thức `run()`
3. Tạo thể hiện
4. Triệu gọi phương thức `start()`

```
public class FirstThread extends Thread {
        @Override
        public void run() {
                System.out.println("Run
                thread + " + this.getId());
        }
}
```

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

Ex: PrintNameThread (1)

```
class PrintNameThread extends Thread {
        PrintNameThread(String name) {
                super(name);
                start(); // start thread
        }
        public void run() {
                String name = getName();
                 for (int i = 0; i< 50; i++){
                        System.out.print(name);
                 }
        }
}
```

- **ThreadTest**

```
public class ThreadTest {
    public static void main(String args[]){
          PrintNameThread pnt1 = new PrintNameThread("A");
          PrintNameThread pnt2 = new PrintNameThread("B");
          PrintNameThread pnt3 = new PrintNameThread("C");
          PrintNameThread pnt4 = new PrintNameThread("D");
    }
}
```

Ex: PrintNameThread (2)

- Sample output

```
AAAAAAAAAAAAAAAAAAAAAAAACCCCCCCCCCCCCCCCCCCBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCAAAAAAAAAAADDDDDDAAAAAAAAAAAAAA
```

```
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAACCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCDDDDDDDDDDDDDDDDDDDDDBBBBBBDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
```

```
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAACCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCBBBBBBBBBBBBBBBBBBDDDDDDDBBBBBBBBBBBDDDDDDDDDDDDBBBBBBBBB
BBBBBBBBBBBBDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
```

```
AAAADDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBDDDDDDAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAADD
```

Tạo thread (2)

Thực thi giao tiếp `java.lang.Runnable`

1. Khai báo lớp thực thi interface `Runnable`
2. Cài đặt (`override`) phương thức `run()`
3. Tạo thể hiện
4. Tạo thể hiện của lớp Thread bằng phương thức khởi tạo `Thread(Runnable target)`
5. Triệu gọi phương thức `start()`

```
public class FirstThread implements Runnable {
        @Override
        public void run() {
                System.out.println("Run thread!");
        }
}
```

```
class ThreadTest {
        public static void main(String[] args) {
                Thread t1 = new Thread(new FirstThread());
                Thread t2 = new Thread(new FirstThread());

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

Ex: PrintNameThread (3)

```
class PrintNameThreadRunable implements Runnable {
        Thread thread;
        PrintNameThreadRunable(String name) {
                thread = new Thread(this, name);
                thread.start();
        }
        public void run() {
                String name = thread.getName();
                for (int i = 0; i < 50; i++) {
                        System.out.print(name);
                }
        }
}
```

```
public class PrintNameThreadRunableTest {
    public static void main(String[] args) {
        Thread pnt1 = new Thread(new PrintNameThreadRunable("A"));
        Thread pnt2 = new Thread(new PrintNameThreadRunable("B"));
        Thread pnt3 = new Thread(new PrintNameThreadRunable("C"));
        Thread pnt4 = new Thread(new PrintNameThreadRunable("D"));
    }
}
```

So sánh

- **Thừa kế lớp Thread**
  - Ít mã nguồn hơn
  - Không kế thừa lớp khác được
  - ?
- **Thực thi giao tiếp Runnable**
  - Nhiều mã nguồn hơn
  - Có thể kế thừa lớp khác được
  - ?

Một số phương thức lớp Thread

Static method

- Static Thread currentThread()
  - Trả về thread hiện tại
- Static void sleep(long msec)
  - Tạm dừng thread một khoảng thời gian
- Static void sleep(long msec,int nsec)
  - ...
- Static void yield()
  - Thông báo hệ thống trả CPU -> chọn thread khác để thực thi
- Static void wait()
  - Khóa thread cho đến khi phương thức notify hoặc notifyAll được gọi

Instance method

- String getName()
- Int getPriority()
- Boolean isAlive()
- Void join()
- Void join(long msec)
- Void join(long msec, int nsec)
- Void run()
- Void setName(String s)
- Void setPriority(int p)
- Void start()
- String toString()

Ex: phương thức join

```
class ThreadJoin extends Thread {
    public void run() {
        try {
            for (int i = 0; i < 3; i++) {
                Thread.sLeep(1000);
                System.out.println(" Thread  " +
                        Thread.currentThread().getName() + ". Round: " + i);
            }
        } catch (InterruptedException e) {
        }
    }
}
```

```
public class JoinThread {
    public static void main(String[] args) throws Exception{
        ThreadJoin tj1 = new ThreadJoin();
        ThreadJoin tj2 = new ThreadJoin();
        System.out.println("Thread: " + Thread.currentThread().getName() + " start");
        tj1.start();
        tj1.join();
        tj2.start();
        tj2.join();
        System.out.println("Thread: " + Thread.currentThread().getName() + " end");
    }
}
```

3. Đồng bộ thread

Đồng bộ (Synchronization)

- Bài toán đồng bộ
  - Các thread đồng thời có thể cùng yêu cầu (hoặc thay đổi) những tài nguyên bên ngoài hệ thống.
  - Nhiều trường hợp cần giao tiếp giữa các thread để *hiểu* trạng thái và hoạt động của nhau.
- Trong Java
  1. Đồng bộ đối tượng

```
synchronized(sync_object)
{
   // Access shared variables and other
   // shared resources
}
```

  2. Phương thức đồng bộ hoặc một phần của phương thức cần đồng bộ bằng cách đặt từ khóa **synchronized**

Ex: Không đồng bộ

```
class Sender
{
    public void send(String[] message) throws InterruptedException
    {
        System.out.println(Thread.currentThread().getName() + " start send: ");
        for(int i = 0; i < message.length;i++)
        {
            System.out.println(message[i]);
            Thread.sleep(1000);
        }
    }
}
```

```
class ThreaderSend extends Thread {
    private String[] message;
    Sender sender;
    public ThreaderSend(String[] message, Sender sender) {
        this.message = message;
        this.sender = sender;
    }
    public void run(){
        try {
            sender.send(message);
        } catch (InterruptedException ex) {}
    }
}
```

Ex: Không đồng bộ (Cont.)

```
class UnSynchronized {
    public static void main(String[] args) {
        Sender sender = new Sender();
        ThreaderSend step1 = new ThreaderSend(new String[] {"one","two","three"}, sender);
        ThreaderSend step2 = new ThreaderSend(new String[] {"four","five","six"}, sender);
        ThreaderSend step3 = new ThreaderSend(new String[] {"seven","eight","nine"}, sender);
        step1.start();
        step2.start();
        step3.start();
    }
}
```

- Sample output

```
Thread-2 start send:
seven
Thread-0 start send:
one
Thread-1 start send:
four
eight
five
two
six
three
nine
```

```
Thread-1 start send:
four
Thread-0 start send:
one
Thread-2 start send:
seven
two
five
eight
six
nine
three
```

```
Thread-1 start send:
four
Thread-0 start send:
one
Thread-2 start send:
seven
two
five
eight
three
six
nine
```

Ex: Đồng bộ khóa đối tượng

```
class ThreaderSend extends Thread {
    ...
    public void run(){
        try {
            synchronized(sender)
            {
                sender.send(message);
            }
        } catch (InterruptedException ex) {}
}}
```

- Sample output

```
Thread-0 start send:
one
two
three
Thread-2 start send:
seven
eight
nine
Thread-1 start send:
four
five
six
```

```
Thread-0 start send:
one
two
Three
Thread-1 start send:
four
five
six
Thread-2 start send:
seven
eight
nine
```

Ex: Đồng bộ khóa phương thức

```
class Sender
{
    synchronized public void send(String[] message) throws InterruptedException     {
        System.out.println(Thread.currentThread().getName() + " start send: ");
        for(int i = 0; i < message.length;i++)
        {
            System.out.println(message[i]);
            Thread.sleep(1000);
        }
    }
}
```

- Sample output

```
Thread-0 start send:
one
two
three
Thread-2 start send:
seven
eight
nine
Thread-1 start send:
four
five
six
```

```
Thread-0 start send:
one
two
Three
Thread-1 start send:
four
five
six
Thread-2 start send:
seven
eight
nine
```

Ex: Đồng bộ khóa một phần phương thức

```
class Sender {
    public void send(String[] message) throws InterruptedException {
        System.out.println(Thread.currentThread().getName() + " start send: ");
        synchronized (this) {
            for (int i = 0; i < message.length; i++) {
                System.out.println(message[i]);
                Thread.sleep(1000);
            }
        }
    }
}
```

- Sample output

```
Thread-1 start send:
four
Thread-0 start send:
Thread-2 start send:
five
six
seven
eight
nine
one
two
three
```

```
Thread-1 start send:
four
Thread-2 start send:
Thread-0 start send:
five
six
one
two
three
seven
eight
nine
```

```
Thread-0 start send:
one
Thread-1 start send:
Thread-2 start send:
two
three
seven
eight
nine
four
five
six
```

4. Liên lạc giữa các thread

Interthread communication

Problem: đồng bộ/liên lạc

- Xử lý bài toán đồng bộ giữa các luồng có nhiều kỹ thuật khác nhau
- Java sử dụng một cơ chế hoàn toàn khác, sử dụng giao tiếp giữa các luồng để thực hiện việc đồng bộ
  - Cơ chế trong đó một luồng tạm dừng chạy/xử lý công việc/tài nguyên quan trọng và chờ đợi sự cho phép của một luồng khác
  - `Inter Thread Communication` là một trong những cơ sở khác biệt trong phát triển ứng dụng đa luồng của java.
  - Các ứng dụng sử dụng `inter thread communication` nhanh hơn các ứng dụng khác (trong Java)
- Cơ chế này được cài đặt bởi các phương thức
  - `wait()`
  - `notify()`
  - `notifyAll()`

Cơ chế Inter Thread Communication

- **wait()**
  - Thread từ bỏ tài nguyên và chuyển sang trạng thái sleep cho đến khi một phương thức khác sử dụng cùng tài nguyên triệu gọi một phương thức notify

```
public final void wait() throws
InterruptedException
public final void wait(long timeout) throws
InterruptedException
```

- **notify()**
  - Đánh thức **một** thread đã gọi phương thức wait() trong cùng đối tượng
  - Nếu nhiều hơn một thread -> lựa chọn thread nào để đánh thức

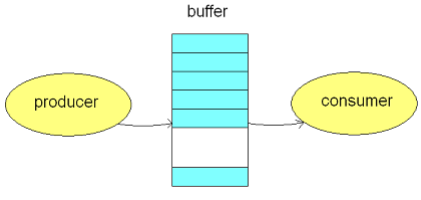
```
public final void notify()
```

- **notifyAll()**
  - Đánh thức tất cả các thread đã gọi phương thức wait() trong cùng một đối tượng

```
public final void notifyAll()
```

Ex: producer / consumer problem

- **Một cách đơn giản**
  - Hai tiến trình/luồng truy cập vào cùng một tài nguyên (thường shared) tại cùng thời điểm
  - Một trong hai sẽ không thể thực hiện được (failed) vì tài nguyên đang được sử dụng bởi tiến trình/luồng còn lại
- **Xử lý**
  - Bài toán xử lý đồng bộ đa luồng
- **Mở rộng**
  - Producer không thể thêm tài nguyên bộ đệm đầy
  - Consumer không thể lấy tài nguyên khi bộ đệm rỗng
  - 1 – 1 producer / consumer



Ex: Không đồng bộ

```
class Producer extends Thread {
    SharedData sharedData;
    public Producer(SharedData sharedData) {
        this.sharedData = sharedData;
        this.start();
    }
    public void run() {
        for (int i = 0; i < 5; i++) {
            sharedData.produce((int) (Math.random() * 100));
        }
    }
}
```

```
class Consumer extends Thread {
    SharedData sharedData;
    public Consumer(SharedData sharedData) {
        this.sharedData = sharedData;
        this.start();
    }
    public void run() {
        for (int i = 0; i < 5; i++) {
            sharedData.consume();
        }
    }
}
```

```
class SharedData {
    int data;
    public void produce(int value) {
        data = value; //
        System.out.println("produce: " + value);
    }
    public void consume() {
        System.out.println("consume: " + data);
    }
}
```

Ex: Không đồng bộ (Cont)

```
public class ProducerConsumerEx {
    public static void main(String[] args) {
        SharedData s = new SharedData();
        new Producer(s);
        new Consumer(s);
    }
}
```

- **Sample Output**

```
produce: 51
consume: 51
consume: 52
consume: 52
consume: 52
consume: 52
produce: 52
produce: 85
produce: 20
produce: 98
```

```
consume: 51
produce: 51
produce: 41
consume: 51
produce: 55
consume: 55
consume: 79
consume: 79
produce: 79
produce: 45
```

```
consume: 18
produce: 18
consume: 18
consume: 23
consume: 23
produce: 23
consume: 23
produce: 66
produce: 99
produce: 59
```

```
produce: 56
consume: 56
consume: 55
consume: 55
consume: 55
consume: 55
produce: 55
produce: 24
produce: 17
produce: 63
```

Ex: đồng bộ

```
class Producer extends Thread {
    SharedData sharedData;
    public Producer(SharedData sharedData) {
        this.sharedData = sharedData;
        this.start();
    }
    public void run() {
        for (int i = 0; i < 5; i++) {
            sharedData.produce((int) (Math.random() * 100));
        }
    }
}
```

```
class Consumer extends Thread {
    SharedData sharedData;
    public Consumer(SharedData sharedData) {
        this.sharedData = sharedData;
        this.start();
    }
    public void run() {
        for (int i = 0; i < 5; i++) {
            sharedData.consume();
        }
    }
}
```

```
class SharedData {
    int data;
    public synchronized void produce(int value) {
        data = value; //
        System.out.println("produce: " + value);
    }
    public synchronized void consume() {
        System.out.println("consume: " + data);
    }
}
```

Ex: đồng bộ

```
public class ProducerConsumerEx {
    public static void main(String[] args) {
        SharedData s = new SharedData();
        new Producer(s);
        new Consumer(s);
    }
}
```

- **Sample Output**

```
produce: 1
consume: 1
consume: 1
consume: 1
consume: 1
consume: 1
produce: 80
produce: 5
produce: 80
produce: 36
```

```
consume: 0
consume: 0
consume: 0
consume: 0
consume: 0
produce: 72
produce: 44
produce: 97
produce: 78
produce: 29
```

```
consume: 0
consume: 0
consume: 0
consume: 0
produce: 50
produce: 49
produce: 49
produce: 80
produce: 14
consume: 14
```

```
produce: 75
consume: 75
consume: 75
consume: 75
consume: 75
consume: 75
produce: 63
produce: 18
produce: 55
produce: 40
```

Ex: producer/consumer

```
class  SharedData {
    int data;
    boolean produced = false;
    public synchronized void produce(int value) {
        if(produced) {// đã cung cấp giá trị
            try{
              this.wait();
            }catch(InterruptedException ex){}
        }
        data = value; //
        System.out.println("produce: " + value);
        produced = true;
        notify();
    }
    public synchronized void consume() {
        if(!produced){// chưa cung cấp giá trị mới
            try{
              this.wait();
            }catch(InterruptedException ex){}
        }
        System.out.println("consume: " + data);
        produced = false;
        notify();
    }
}
```

- **Sample Output**

```
produce: 57
consume: 57
produce: 35
consume: 35
produce: 74
consume: 74
produce: 72
consume: 72
produce: 43
consume: 43
```

```
produce: 68
consume: 68
produce: 91
consume: 91
produce: 35
consume: 35
produce: 85
consume: 85
produce: 90
consume: 90
```

```
produce: 84
consume: 84
produce: 87
consume: 87
produce: 92
consume: 92
produce: 82
consume: 82
produce: 86
consume: 86
```

```
produce: 88
consume: 88
produce: 68
consume: 68
produce: 39
consume: 39
produce: 88
consume: 88
produce: 74
consume: 74
```

Demo

- **Tạo luồng**
  - Kế thừa thread
  - Cài đặt interface Runnable
- **Phương thức join**
  - ...
  - ...
- **Đồng bộ**
  - Đối tượng
  - Phương thức / một phần phương thức
- **Mô tả**
  - SharedData
  - Producer
  - Consumer
  - PCTest
- **Vấn đề đồng bộ đa luồng**
  - Sử dụng synchronized
- **Vấn đề bài toán producer/consumer**
  - Sử dụng wait(), notify(), notifyAll()

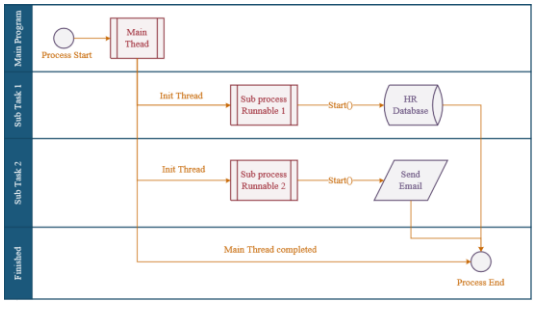
Exercise

1. **Hoàn thiện / mở rộng bài toán producer/consumer**
   1. Kích thước bộ đệm
   2. Tốc độ producer / consumer
2. **Xây dựng bộ đếm thời gian chạy theo giờ, phút, giây**
   - Gồm 03 thread tương ứng giờ, phút, giây
   - Thread giây: mỗi giây đếm tăng 1, đến 60 thì reset về 0 và yêu cầu thread phút tăng 1, tiếp tục quá trình
   - Thread phút: khi bộ đếm tăng lên 60 thì yêu cầu thread giờ tăng lên 01 và tiếp tục quá trình
   - Thread giờ: hiển thị giờ từ 0-23

1. **Ý tưởng bài tập lớn cho các bài toán có mô hình producer/consumer**
   1. Chat group (voice/video)
   2. Stream (Video/File ..)

Tổng kết

- **Luồng / Đa luồng**
- **Làm việc với luồng trong Java**
  - Tạo luồng
  - Đồng bộ trong đa luồng
  - Liên lạc giữa các luồng trong Java
- **Chủ đề**
  - Các bài toán nghiệp vụ có ứng dụng kỹ thuật đa luồng
  - ...



Q & A