LẬP TRÌNH NÂNG CAO

Lập trình JAVA

NỘI DUNG

- Thread
  - Cách tạo luồng trong Java
  - Đồng bộ hóa luồng
  - Các dòng nhập xuất
- File
- Socket

BIỂU THỨC LAMBDA

- Được sử dụng để viết mã ngắn gọn và dễ hiểu hơn, đặc biệt là khi làm việc với các API hoặc xử lý các sự kiện trong giao diện người dùng
- Là một cách để biểu diễn các phương thức ẩn danh (anonymous methods) mà không cần phải tạo một lớp con hoặc implement một interface.
- Cú pháp:

```
(parameters) -> statement;

(parameters) -> { statements; }
```

LAMBDA VỚI KHÔNG CÓ THAM SỐ

```
interface SayHello {
    void sayHello();
}
```

```
SayHello hello = () -> System.out.println("Hello, world!");
hello.sayHello();
```

```
Hello, world!
```

LAMBDA VỚI NHIỀU CÂU LỆNH

```
interface MathOperation {
    int operate(int a, int b);
}
```

```
MathOperation multiply = (a, b) -> {
    int result = a * b;
    return result;
};
System.out.println(multiply.operate(5, 3));
```

```
15
```

LAMBDA VỚI MỘT THAM SỐ

```
interface Greeting {
    void display(String name);
}
```

```
Greeting greeting = (name) -> System.out.println("Hello, " + name);
greeting.display("JAVA");
```

```
Hello, JAVA
```

LAMBDA VỚI NHIỀU THAM SỐ

```
interface Add {
    int add(int a, int b);
}
```

```
Add addition = (a, b) -> a + b;
System.out.println(addition.add(5, 3));
```

```
8
```

SỬ DỤNG LAMBDA

```
List<String> names = Arrays.asList("John", "Jane", "Paul", "Anna");
names.forEach(name -> System.out.println(name));

List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5, 6);
numbers.stream()
        .filter(n -> n % 2 == 0)
        .forEach(n -> System.out.println(n));
```

```
John
Jane
Paul
Anna
2
4
6
```

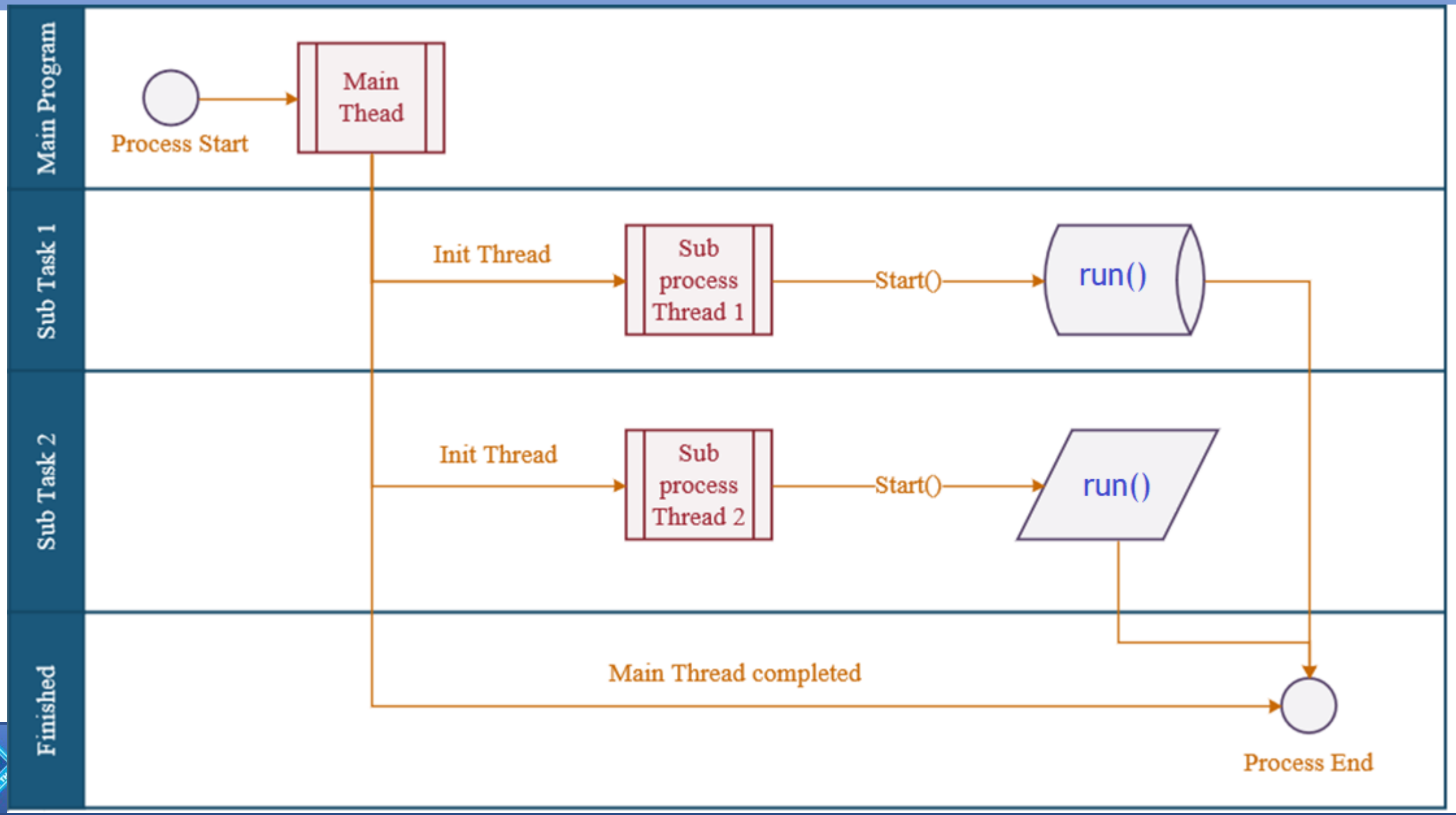
THREAD (1)

- Thread (luồng) là một đơn vị cơ bản để thực thi một chương trình.
- Mỗi thread có thể thực hiện một công việc hoặc một đoạn mã riêng biệt
- Multi-thread: nhiều thread có thể chạy song song → tận dụng tối đa các CPU đa lõi và giúp tăng hiệu suất của chương trình

THREAD (2)

- Khi chương trình Java thực thi, hàm main() sẽ tạo ra một thread (thread chính). Trong thread chính:
  - Có thể tạo các thread con.
  - Thread chính là thread kết thúc cuối cùng.
  - Khi thread chính ngừng thực thi, chương trình sẽ kết thúc

MULTI-THREAD (1)



MULTI-THREAD (2)

Ưu điểm

- Tăng hiệu suất (Chạy song song nhiều tác vụ. Tận dụng CPU đa lõi (multi-core)
- Cải thiện tính đáp ứng (tránh tình trạng "treo" đối với các tác vụ xử lý trong thời gian dài)
- Xử lý đồng thời nhiều yêu cầu

Khuyết điểm

- Phức tạp trong điều phối (race condition (cạnh tranh), deadlock, …)
- Đồng bộ dữ liệu
- Sử dụng nhiều tài nguyên (mỗi thread sử dụng bộ nhớ và CPU riêng)

MULTI-THREAD (3)

```
class Counter implements Runnable {
    private int count = 0;

    public void run() {
        for (int i = 0; i < 5; i++) {
            count++;
            System.out.println(Thread.currentThread().getName() + ": " + count);
            try {
                Thread.sleep(500);
            } catch (InterruptedException e) {
                System.out.println(e);
            }
        }
    }
}
```

```
public static void main(String[] args) {
    Counter counter = new Counter();

    Thread t1 = new Thread(counter, "Thread 1");
    Thread t2 = new Thread(counter, "Thread 2");

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

```
Thread 1: 2
Thread 2: 2
Thread 2: 3
Thread 1: 4
Thread 1: 6
Thread 2: 5
Thread 2: 7
Thread 1: 8
Thread 2: 10
Thread 1: 10
```

THREAD TRONG JAVA

- Thread có thể được tạo ra bằng 2 cách:
  - Tạo lớp dẫn xuất từ lớp `Thread`
  - Tạo lớp hiện thực giao tiếp `Runnable`.

"Thread" CLASS (1)

Sử dụng lớp "`Thread`"

- Kế thừa từ lớp "`Thread`"
- Override phương thức "`run()`"
  - Các câu lệnh thực thi trong thread được viết trong phương thức "`run()`"
- Gọi phương thức "`start()`" để thực thi thread

"Thread" CLASS (2)

Sử dụng lớp "Thread"

- Kế thừa từ lớp "`Thread`"
- Override phương thức "`run()`"
  - Các câu lệnh thực thi trong thread được viết trong phương thức "`run()`"
- Gọi phương thức "`start()`" để thực thi thread

"Thread" CLASS (3)

```
class Thread1 extends Thread{
    @Override
    public void run() {
        for(int i=1; i<500; i++)
            System.out.println("Thread 1: " + i);
    }
}
class Thread2 extends Thread{
    @Override
    public void run() {
        for(int i=500; i<1000; i++)
            System.out.println("Thread 2: " + i);
    }
}
```

```
public static void main(String[] args) {
    Thread1 thr1 = new Thread1();
    thr1.start();
    Thread2 thr2 = new Thread2();
    thr2.start();
}
```

```
Thread 1: 44
Thread 1: 45
Thread 1: 46
Thread 2: 500
Thread 2: 501
Thread 1: 47
Thread 1: 48
Thread 2: 502
Thread 1: 49
Thread 1: 50
Thread 1: 51
Thread 2: 503
Thread 2: 504
Thread 1: 52
Thread 1: 53
```

"Runnable" INTERFACE(1)

Sử dụng interface "Runnable"

- Kế thừa interface "Runnable"
- Override phương thức "run()"
  - Các câu lệnh thực thi trong thread được viết trong phương thức "run()"
- Gọi phương thức "start()" để thực thi thread

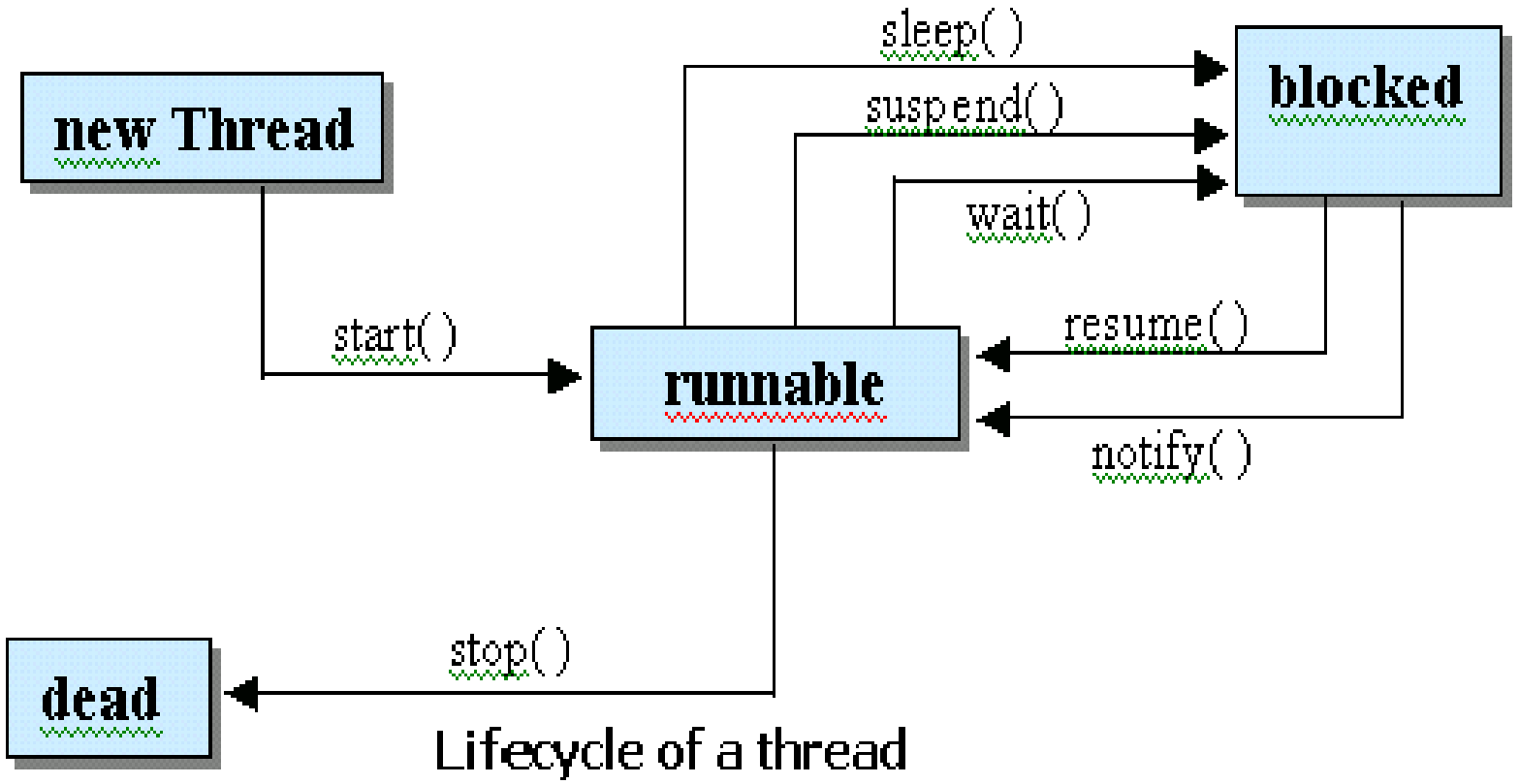
"Runnable" INTERFACE(2)

```
class Runnanle1 implements Runnable{
    @Override
    public void run() {
        for(int i=1; i<500; i++)
            System.out.println("Thread 1: " + i);
    }
}
class Runnanle2 implements Runnable{
    @Override
    public void run() {
        for(int i=500; i<1000; i++)
            System.out.println("Thread 2: " + i);
    }
}
```

```
public static void main(String[] args) {
    Runnanle1 run1 = new Runnanle1();
    Thread thr1 = new Thread( target: run1);
    thr1.start();

    Runnanle2 run2 = new Runnanle2();
    Thread thr2 = new Thread( target: run2);
    thr2.start();
}
```

```
Thread 2: 500
Thread 2: 501
Thread 2: 502
Thread 2: 503
Thread 1: 1
Thread 2: 504
Thread 1: 2
Thread 1: 3
Thread 2: 505
Thread 2: 506
Thread 2: 507
Thread 2: 508
Thread 2: 509
Thread 2: 510
Thread 1: 4
Thread 2: 511
Thread 1: 5
```

VÒNG ĐỜI CỦA THREAD



Lifecycle of a thread

PHƯƠNG THỨC CỦA "Thread" CLASS (1)

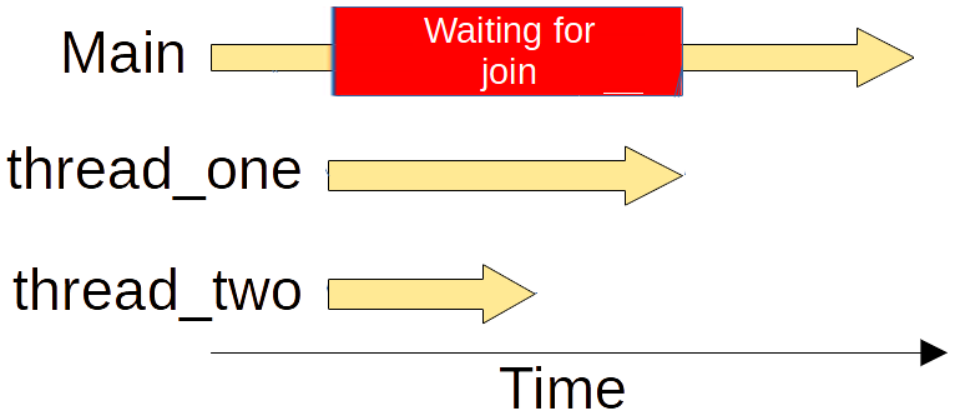
- `start()`: Khởi động thread. Khi gọi `start()`, Java sẽ gọi phương thức `run()` của luồng đó.
- `run()`: Chứa các câu lệnh sẽ thực thi của thread. Là phương thức sẽ được override khi tạo thread
- `sleep(long millis)`: Đặt thread hiện tại "ngủ" trong một khoảng thời gian xác định (đơn vị: mili giây).
- `join()`: Làm cho thread hiện tại chờ đợi cho đến khi thread khác hoàn thành.

PHƯƠNG THỨC CỦA "Thread" CLASS (2)

- `join(int milisecond)`: Làm cho thread hiện tại chờ đợi trong khoảng thời gian.
- `getId()`: Lấy ID của thread.
- `getName()`: Lấy tên của thread.
- `isAlive()`: Kiểm tra xem thread có đang sống (đang chạy) hay không.
- `setPriority(int priority)`: Thiết lập độ ưu tiên cho thread. Độ ưu tiên là một số nguyên trong khoảng từ 1 (thấp) đến 10 (cao). Giá trị mặc định là 5.

CHIẾM QUYỀN THỰC THI (1)

- Phương thức `join()` trong lớp Thread cho phép một thread (gọi là thread hiện tại) chờ cho đến khi một thread khác (gọi là thread được gọi) hoàn thành.
- Nếu gọi `join()` trong một thread, thread hiện tại sẽ dừng lại và không tiếp tục cho đến khi thread được gọi kết thúc.



CHIẾM QUYỀN THỰC THI (2)

```
public class ThreadDemo2 {
    public static void main(String[] args) throws InterruptedException {
        MyThread thread1 = new MyThread("Thread1");
        MyThread thread2 = new MyThread("Thread2");

        thread1.start();
        //thread1.join();
        System.out.println("Thread1 da hoan thanh.");

        thread2.start();
    }

}
class MyThread extends Thread {
    public MyThread(String name){
        super(name);
    }
    public void run() {
        for(int i=1; i<500; i++)
            System.out.println(Thread.currentThread().getName() + ": " + i);
    }
}
```

```
Thread1 da hoan thanh.
Thread2: 1
Thread2: 2
Thread1: 1
Thread2: 3
Thread1: 2
Thread2: 4
Thread1: 3
Thread2: 5
Thread1: 4
Thread2: 6
Thread1: 5
Thread2: 7
Thread1: 6
Thread2: 8
Thread1: 7
Thread2: 9
Thread1: 8
Thread2: 10
Thread1: 9
Thread2: 11
```

CHIẾM QUYỀN THỰC THI (3)

```
public class ThreadDemo2 {
    public static void main(String[] args) throws InterruptedExcepti
        MyThread thread1 = new MyThread("Thread1");
        MyThread thread2 = new MyThread("Thread2");

        thread1.start();
        thread1.join();
        System.out.println("Thread1 da hoan thanh.");

        thread2.start();
    }

}
class MyThread extends Thread {
    public MyThread(String name){
        super(name);
    }
    public void run() {
        for(int i=1; i<500; i++)
            System.out.println(Thread.currentThread().getName() + ": " + i);
    }
}
```

Các thread khác phải chờ đến khi thread1 được hoàn thành

```
Thread1: 1
Thread1: 2
Thread1: 3
Thread1: 4
Thread1: 5
Thread1: 6
.....
Thread1: 497
Thread1: 498
Thread1: 499
Thread1 da hoan thanh.
Thread2: 1
Thread2: 2
Thread2: 3
Thread2: 4
Thread2: 5
```

ĐỘ ƯU TIÊN (1)

- Mỗi thread có 1 độ ưu tiên (PRIORITY).
  - Quyết định mức ưu tiên khi cấp phát CPU cho các thread.
- Độ ưu tiên có giá trị từ 1-10.
- 3 chế độ ưu tiên được định nghĩa sẵn:
  - Thread.MIN_PRIORITY (1) : thấp nhất
  - Thread.NORM_PRIORITY (5): mặc định
  - Thread.MAX_PRIORITY (10): cao nhất

ĐỘ ƯU TIÊN (2)

- Làm việc với độ ưu tiên trong lớp `Thread`:
- `setPriority(int priority)`: Thiết lập độ ưu tiên cho thread
  - `priority`: giá trị từ 1 - 10
- `getPriority(int priority)`: trả về độ ưu tiên của thread

```
Thread t1 = new WorkingThread("Luồng 1");
Thread t2 = new WorkingThread("Luồng 2");
Thread t3 = new WorkingThread("Luồng 3");

t1.setPriority(1);
t2.setPriority(5);
t3.setPriority(10);

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

TẠM DỪNG

- `Thread.sleep(milisecond)`: tạm dừng thread trong 1 khoảng thời gian (mili giây)

```
public static void main(String[] args) throws InterruptedException {
    for(int i=1; i<10; i++){
        System.out.println(i);
        Thread.sleep(5000);
    }
}
```

ĐIỀU PHỐI THREAD

- Các thread với quyền ưu tiên cao có một cơ hội nhận thời gian sử dụng CPU để hoàn thành trước các luồng với quyền ưu tiên thấp hơn.
- JVM sử dụng giải thuật không độc quyền. Vì thế, nếu một thread quyền ưu tiên thấp đang được chạy, thread quyền có quyền ưu tiên cao hơn có thể giành quyền sử dụng CPU của nó.
- Nếu các thread có cùng quyền ưu tiên đang chờ đợi để thực hiện, một thread ngẫu nhiên sẽ được lựa chọn.

ĐỒNG BỘ HÓA (SYNCHRONIZATION) (1)

- Khi nhiều thread cùng đồng thời sử dụng 1 nguồn tài nguyên mà không có cơ chế bảo vệ tài nguyên, có thể xảy ra các vấn đề:
  - Data corruption: dữ liệu bị hỏng
  - Race condition: kết quả không xác định, không chính xác hoặc lỗi không mong muốn

➔ Đồng bộ hóa

```
balance = 1000  # Số dư ban đầu

Thread 1
balance = balance + 100  # Tăng số dư lên 100
Thread 2
balance = balance + 50   # Tăng số dư lên 50
```

ĐỒNG BỘ HÓA (SYNCHRONIZATION) (2)

- Là cơ chế giúp quản lý truy cập đồng thời đến các tài nguyên chia sẻ, đặc biệt trong môi trường đa luồng (multithreading).
- Đảm bảo rằng chỉ một thread có thể truy cập tài nguyên chia sẻ tại một thời điểm.

ĐỒNG BỘ HÓA (SYNCHRONIZATION) (3)

Cơ chế đồng bộ hóa: sử dụng

- Từ khóa "`synchronized`"
- Đối tượng `Lock` (`java.util.concurrent.locks`)
- Từ khóa "`volatile`"
- Gói `java.util.concurrent` (các lớp `ReentrantLock`, `ReadWriteLock`, `Semaphore`...)

"synchronized" KEYWORD (1)

- Từ khóa "synchronized": khối lệnh/phương thức chỉ được thực thi bởi 1 thread.

```
class Counter0 {
    private int count = 0;

    public  void increment() {
        count++;
    }

    public int getCount() {
        return count;
    }
}
```

```
public static void main(String[] args) throws InterruptedException {
    Counter0 counter = new Counter0();

    Thread t1 = new Thread(() -> {
        for (int i = 0; i < 1000; i++) {
            counter.increment();
        }
    });

    Thread t2 = new Thread(() -> {
        for (int i = 0; i < 1000; i++) {
            counter.increment();
        }
    });

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

    t1.join();
    t2.join();

    // Expected output: 2000
    System.out.println("Count: " + counter.getCount());

}
```

```
Count: 1718
```

"synchronized" KEYWORD (2)

- Từ khóa "synchronized": khối lệnh/phương thức chỉ được thực thi bởi 1 thread.

```
class Counter0 {
    private int count = 0;

    public synchronized void increment() {
        count++;
    }

    public int getCount() {
        return count;
    }
}
```

```
public static void main(String[] args) throws InterruptedException {
    Counter0 counter = new Counter0();

    Thread t1 = new Thread(() -> {
        for (int i = 0; i < 1000; i++) {
            counter.increment();
        }
    });

    Thread t2 = new Thread(() -> {
        for (int i = 0; i < 1000; i++) {
            counter.increment();
        }
    });

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

    t1.join();
    t2.join();

    // Expected output: 2000
    System.out.println("Count: " + counter.getCount());
}
```

```
Count: 2000
```

"synchronized" KEYWORD (3)

- Từ khóa "synchronized": khối lệnh/phương thức chỉ được thực thi bởi 1 thread.

```
public void increment() {
    synchronized (this) {
        count++;
    }
}
```

"synchronized" KEYWORD (4)

- Từ khóa "synchronized": khối lệnh/phương thức chỉ được thực thi bởi 1 thread.

```
class Counter1 {
    private int count = 0;

    public void increment() {
        synchronized (this) {
            count++;
        }
    }

    public int getCount() {
        return count;
    }
}
```

```
public static void main(String[] args) throws InterruptedException {
    Counter1 counter = new Counter1();
    Thread t1 = new Thread(() -> {
        for (int i = 0; i < 1000; i++) {
            counter.increment();
        }
    });

    Thread t2 = new Thread(() -> {
        for (int i = 0; i < 1000; i++) {
            counter.increment();
        }
    });

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

    t1.join();
    t2.join();

    System.out.println("Count: " + counter.getCount());
    // Expected output: 2000
}
```

```
Count: 2000
```

"Lock" INTERFACE (1)

- Gói `java.util.concurrent.locks` chứa nhiều lớp Lock
- Các lớp Lock được sử dụng để kiểm soát tốt hơn các tình huống đồng bộ hóa như thời gian khóa, không gian khóa, …)

"Lock" INTERFACE (2)

```java
class Counter2 {
    private int count = 0;
    private final Lock lock = new ReentrantLock();

    public void increment() {
        lock.lock(); // Khoa
        try {
            count++;
        } finally {
            lock.unlock(); //Mo khoa
        }
    }

    public int getCount() {
        return count;
    }
}
```

```java
public class LockInterface {
    public static void main(String[] args) throws InterruptedException {
        Counter2 counter = new Counter2();
        Thread t1 = new Thread(() -> {
            for (int i = 0; i < 1000; i++) {
                counter.increment();
            }
        });

        Thread t2 = new Thread(() -> {
            for (int i = 0; i < 1000; i++) {
                counter.increment();
            }
        });

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

        t1.join();
        t2.join();

        System.out.println("Count: " + counter.getCount());
        // Expected output: 2000
    }
}
```

```
Count: 2000
```

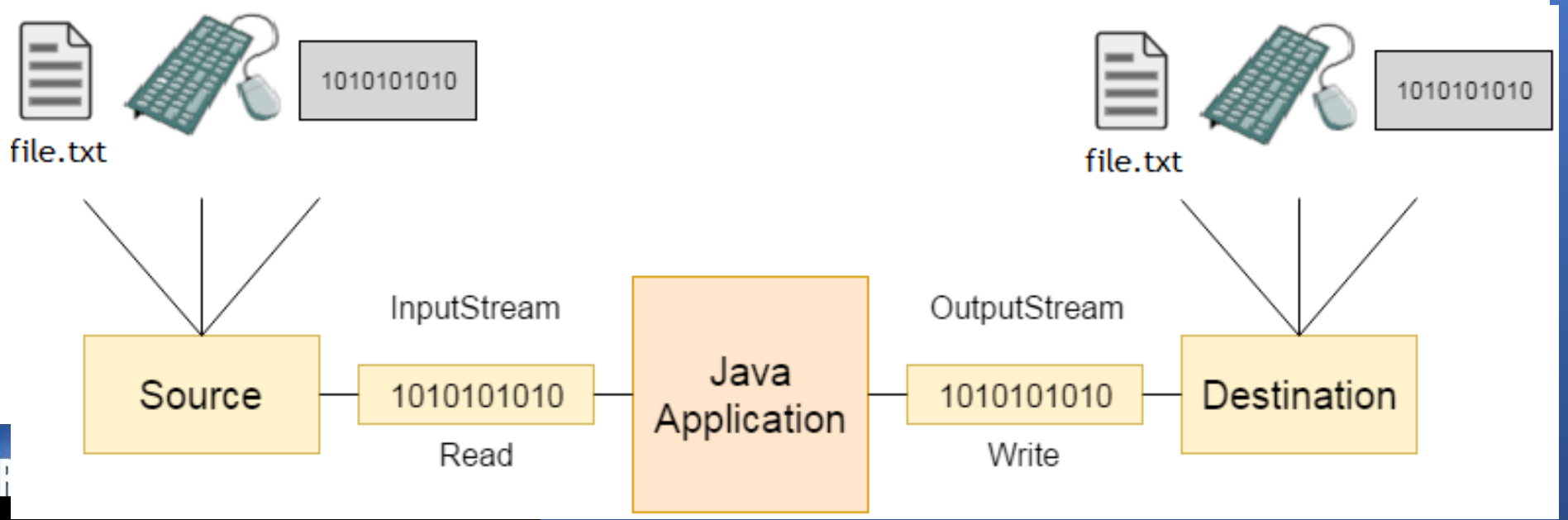
"volatile" KEYWORD

- Đảm bảo rằng giá trị của một biến sẽ luôn được đọc từ bộ nhớ chính, thay vì từ bộ đệm của CPU.
- Điều này giúp tránh tình trạng caching (lưu trữ tạm thời) và đảm bảo tính nhất quán khi nhiều luồng truy cập cùng một biến.

```
private volatile int count = 0;
public void increment() {
    count++;
}

public int getCount() {
    return count;
}
```

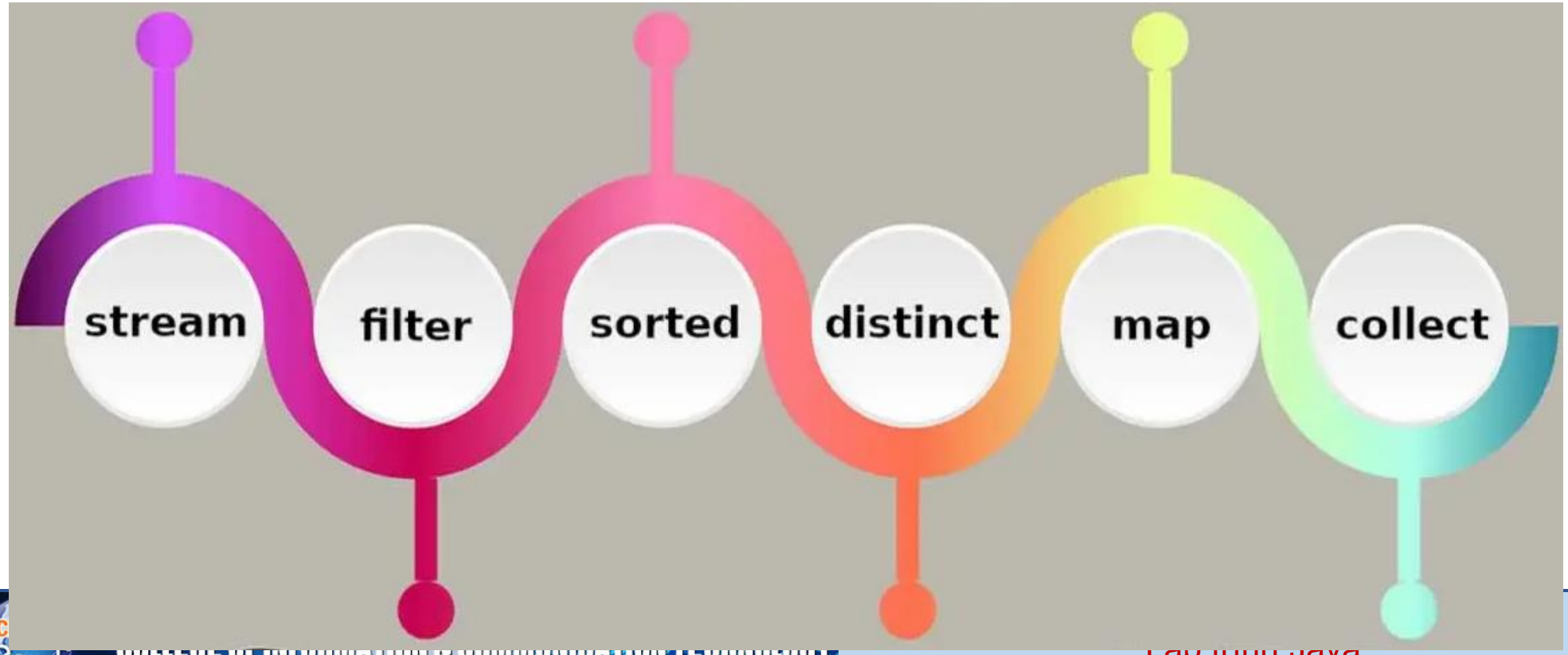
JAVA STREAM (1)

- Là một "công cụ" mạnh mẽ dùng để xử lý dữ liệu bằng các hàm và hỗ trợ lập trình song song.
  - Dữ liệu có thể là collection, array, file, …
- Không phải là một cấu trúc dữ liệu (không lưu dữ liệu và không thể thay đổi dữ liệu).



JAVA STREAM (2)

- Được tạo bởi các class bên trong package `java.util.stream`
- Hỗ trợ các thao tác map, filter, sorted, collect, forEach, reduce, …



CÁC LOẠI STREAM

2 loại stream:

- **Stream tuần tự**: Xử lý từng phần tử một, theo thứ tự chúng xuất hiện trong nguồn.
- **Stream song song**: Xử lý các phần tử song song, thường là trên nhiều stream, để cải thiện hiệu suất với các tập dữ liệu lớn.

TẠO STREAM

- 2 cách tạo stream:
  - ARRAY/LIST.stream()
  - Stream.of()

ARRAY / LIST.stream()

```
List<String> list = Arrays.asList("a", "b", "c", "d");
Stream<String> stream = list.stream();

String[] array = {"a", "b", "c", "d"};
Stream<String> stream = Arrays.stream(array);
```

Stream.of()

```
Stream<String> stream = Stream.of("a", "b", "c", "d");
```

XỬ LÝ DỮ LIỆU (1)

```
Stream<String> stream =
        Stream.of("Jack", "Brown", "Ann", "Danny", "Ann", "Akkan");
```

```
stream.forEach(System.out::println);
```

```
Jack
Brown
Ann
Danny
Ann
Akkan
```

```
System.out.println(stream.count());
```

```
6
```

XỬ LÝ DỮ LIỆU (2)

```
Stream<String> stream =
        Stream.of("Jack", "Brown", "Ann", "Danny", "Ann", "Akkan");
```

```
stream.filter(s -> s.startsWith("A")).forEach(System.out::println);
```

```
Ann
Ann
Akkan
```

```
stream.map(String::toUpperCase).forEach(System.out::println);
```

```
JACK
BROWN
ANN
DANNY
ANN
AKKAN
```

XỬ LÝ DỮ LIỆU (3)

```
Stream<String> stream =
        Stream.of("Jack", "Brown", "Ann", "Danny", "Ann", "Akkan");
```

```
stream.sorted().forEach(System.out::println);
```

```
Akkan
Ann
Ann
Brown
Danny
Jack
```

```
stream.distinct().forEach(System.out::println);
```

```
Jack
Brown
Ann
Danny
Akkan
```

XỬ LÝ DỮ LIỆU (4)

```
List<Integer> numbers = Arrays.asList(7, 3, 1, 6, 9, 4, 8, 2, 5);
numbers.stream()
    .filter(n -> n % 2 != 0)  // Keep odd numbers only
    .map(n -> n * n)          // Square the numbers
    .sorted()                 // Sort the numbers
    .forEach(System.out::println); // Print each number
```

```
1
9
25
49
81
```

STREAM SONG SONG

```
List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9);
numbers.parallelStream()
        .filter(n -> n % 2 != 0)
        .map(n -> n * n)
        .forEach(System.out::println);
```

```
25
81
49
9
1
```

"FILE" CLASS (1)

- Khai báo bên trong package `java.io`
- Đại diện cho 1 tập tin/thư mục
- Cung cấp các phương thức làm việc với tập tin/thư mục: truy xuất thông tin, thuộc tính của file, tạo, xóa, đổi tên, di chuyển, …)

"FILE" CLASS (2)

Phương thức	Ý nghĩa	Ví dụ
`new File()`	Tạo đối tượng File	`File file = new File("path/to/file.txt");`
`exists()`	Kiểm tra tồn tại của tệp	`boolean exists = file.exists();`
`isDirectory()`	Kiểm tra nếu là thư mục hoặc tệp	`boolean isDirectory = file.isDirectory();`
`isFile()`	Kiểm tra nếu là thư mục hoặc tệp	`boolean isFile = file.isFile();`
`mkdir()`	Tạo thư mục	`boolean created = file.mkdir();`

"FILE" CLASS (3)

Phương thức	Ý nghĩa	Ví dụ
`delete()`	Xóa tệp hoặc thư mục	`boolean deleted = file.delete();`
`renameTo()`	Đổi tên tệp hoặc thư mục	`File newFile = new File("path/to/newname.txt"); boolean renamed = file.renameTo(newFile);`
`length()`	Lấy thông tin về tệp	`long length = file.length(); // (byte) long lastModified = file.lastModified();`
`listFiles()`	Lấy danh sách các tệp trong thư mục	`File[] files = file.listFiles();`

"FILE" CLASS (4)

```
File file = new File("data.txt");
if (file.exists()) {
    System.out.println("File da ton tai.");
} else {
    System.out.println("File chua ton tai.");
    try {
        boolean created = file.createNewFile();
        if (created) {
            System.out.println("File da duoc tao.");
        } else {
            System.out.println("Khong the tao file.");
        }
    } catch (Exception e) {
        System.out.println("Lỗi: " + e.getMessage());
    }
}
System.out.println("Kich thuoc file: " + file.length() + " bytes");
```

TRUY XUẤT TẬP TIN / THƯ MỤC

- Được thực hiện bởi các lớp trong package `java.nio.file`
- **Path**: đường dẫn đến một tệp hoặc thư mục.

```
Path path = Paths.get("file.txt");
```

- **Paths**: tạo đối tượng Path từ các chuỗi đường dẫn.

```
Path path = Paths.get("directory", "file.txt");
```

- **Files**: thao tác với tệp, thư mục, và các thuộc tính của chúng.

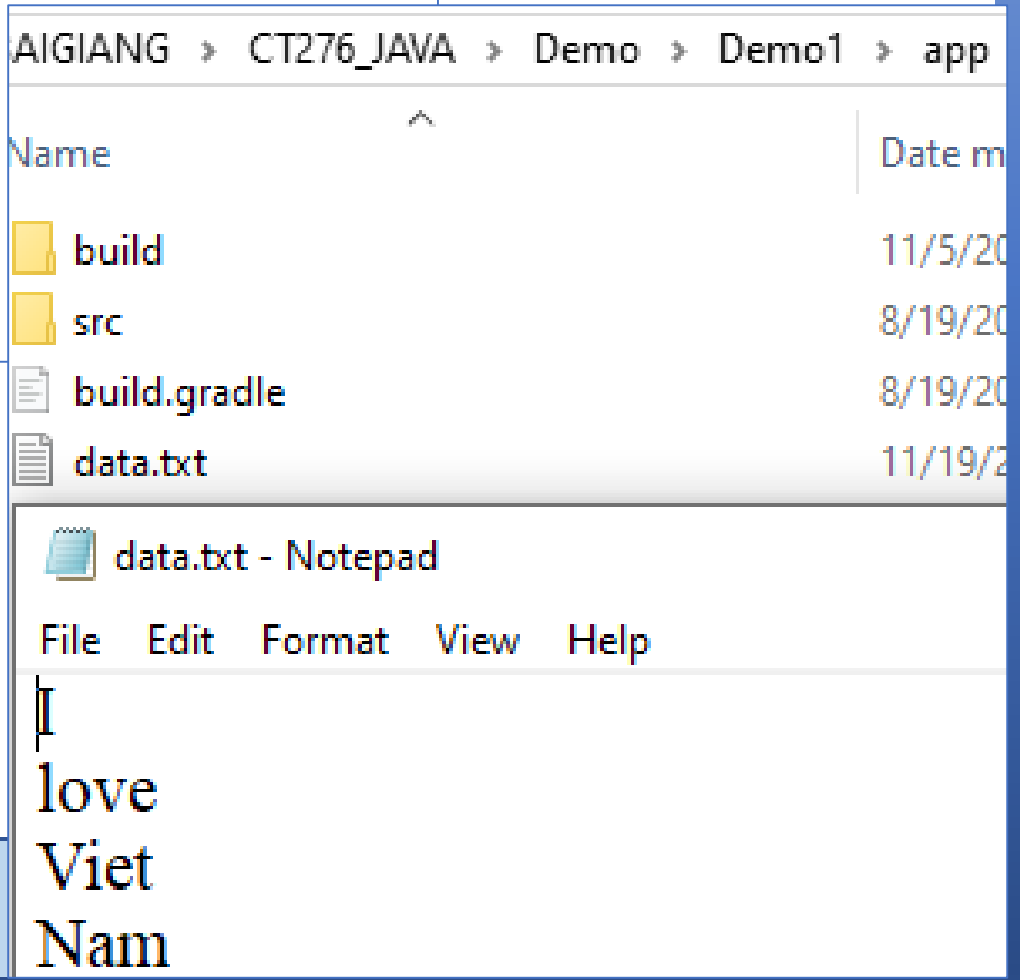
```
Files.readAllLines(path);

Files.copy(source, target);
```

ĐỌC FILE

```
Path path = Paths.get("data.txt");
try {
    List<String> lines = Files.readAllLines(path);
    for (String line : lines) {
        System.out.println(line);
    }
} catch (IOException e) {
}
```

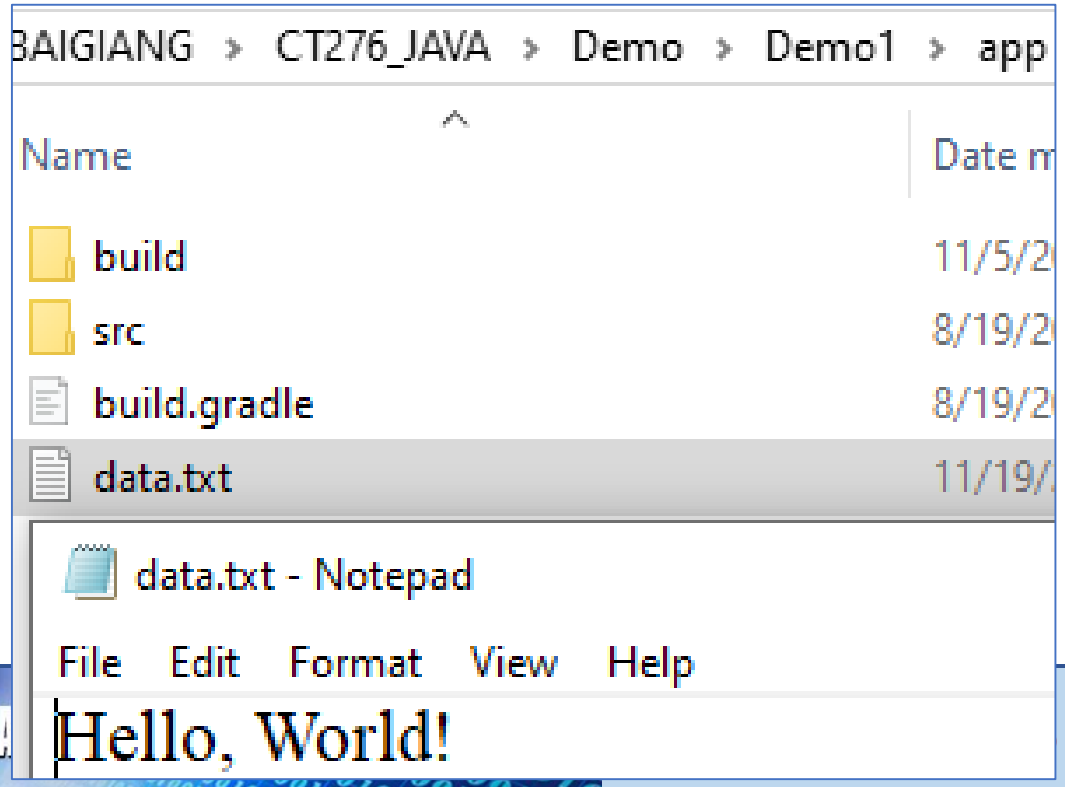
- Files.readAllLines()
- Files.readAllBytes()

```
I
love
Viet
Nam
```



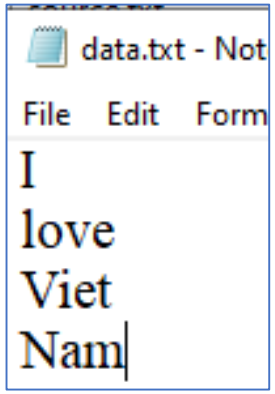
GHI FILE (1)

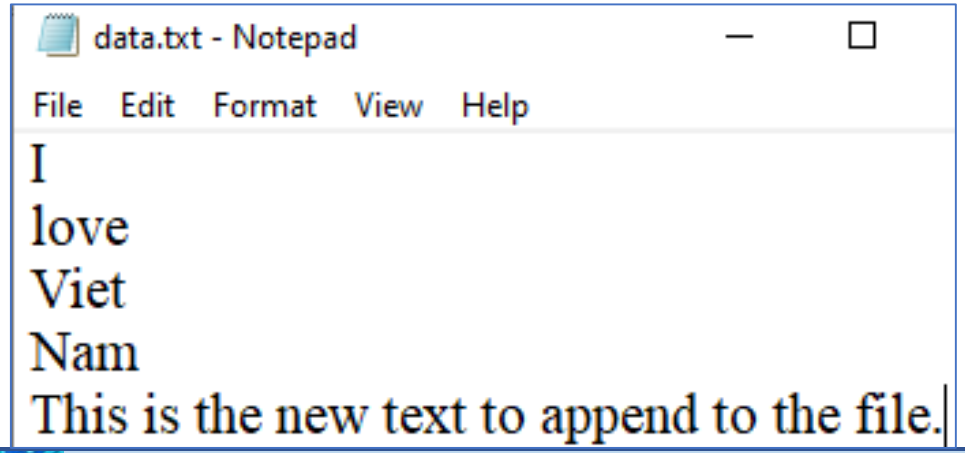
```
Path path = Paths.get("data1.txt");
try {
    Files.write(path, "Hello, World!".getBytes());
} catch (IOException e) {
}
```



GHI FILE (2)

```
String filePath = "data.txt";
String textToAppend = "This is the new text to append to the file.\n";

try {
    // StandardOpenOption.APPEND ensures that the content is appended
    Files.write(Paths.get(filePath), textToAppend.getBytes(),
            StandardOpenOption.APPEND, StandardOpenOption.CREATE);
} catch (IOException e) {
}
```





KIỂM TRA FILE/FOLDER

```
Path path = Paths.get("data.txt");
if (Files.exists(path)) {
    System.out.println("File ton tai.");
} else {
    System.out.println("File khong ton tai.");
}
if (Files.isDirectory(path)) {
    System.out.println("Day la folder.");
} else {
    System.out.println("Day la File.");
}
```

```
File ton tai.
Day la File.
```

SAO CHÉP FILE

```
Path source = Paths.get("source.txt");
Path destination = Paths.get("destination.txt");
try {
    Files.copy(source, destination, StandardCopyOption.REPLACE_EXISTING);
    System.out.println("Sao chep thanh cong.");
} catch (IOException e) {
}
```

XÓA FILE

```
Path path = Paths.get("destination.txt");
try {
    Files.delete(path);
    System.out.println("File duoc xoa.");
} catch (IOException e) {
}
```

LẬP TRÌNH MẠNG

- Giúp kết nối các ứng dụng với nhau thông qua mạng máy tính.
- Hỗ trợ thư viện mạng mạnh mẽ để phát triển các ứng dụng có khả năng giao tiếp qua mạng
- Các khái niệm:
  - Sockets: là một "end-point" trong giao tiếp giữa 2 máy thông qua mạng.
  - URL và URI: địa chỉ tài nguyên.
  - Datagrams: gửi và nhận dữ liệu

GIAO THỨC TRUYỀN TẢI

- TCP (Transmission Control Protocol): đảm bảo độ tin cậy của kết nối (truyền tải dữ liệu đảm bảo không mất dữ liệu và đúng thứ tự).
  - Socket và ServerSocket: giao tiếp qua TCP.
- UDP (User Datagram Protocol): Là một giao thức không kết nối và không đảm bảo độ tin cậy.
  - UDP nhanh hơn TCP nhưng không đảm bảo rằng các gói dữ liệu sẽ đến đích hay không.
  - DatagramSocket và DatagramPacket: giao tiếp qua UDP

SOCKET

- Socket là một “end-point” trong giao tiếp giữa 2 máy tính thông qua mạng.
- Có 2 loại: Socket (client) và ServerSocket (server).
  - Được định nghĩa trong package `java.net`
- Hỗ trợ thực hiện các kết nối mạng, cho phép các ứng dụng giao tiếp với nhau qua các giao thức như TCP/IP
- Cho phép gửi và nhận dữ liệu giữa các máy tính hoặc thiết bị trong mạng.

"ServerSocket" CLASS (1)

- Thuộc gói `java.net`
- Được sử dụng để tạo một server socket,
  - Lắng nghe các kết nối đến từ các client trên một port (cổng.)
  - Giao tiếp với các client

"ServerSocket" CLASS (2)

- `ServerSocket(int port)`: Tạo một ServerSocket để lắng nghe các kết nối từ client trên cổng port.
- `Socket accept()`: Chấp nhận kết nối từ client. Giá trị trả về là 1 đối tượng Socket client.
- `getLocalPort()`: trả về port mà ServerSocket đang lắng nghe.
- `close()`: Đóng ServerSocket, ngừng lắng nghe các kết nối mới.

"Socket" CLASS (1)

- Thuộc gói `java.net`
- Được sử dụng để tạo một client socket,
  - Kết nối đến server socket thông qua một port.
  - Giao tiếp với các server

"Socket" CLASS (2)

- `Socket(String host, int port)`: Khởi tạo một kết nối tới server theo địa chỉ host (tên máy chủ hoặc IP) và cổng port.
- `InputStream getInputStream()`: Trả về một đối tượng InputStream để đọc dữ liệu từ server.
- `OutputStream getOutputStream()`: Trả về một đối tượng OutputStream để gửi dữ liệu đến
- `server.close()`: Đóng kết nối socket.

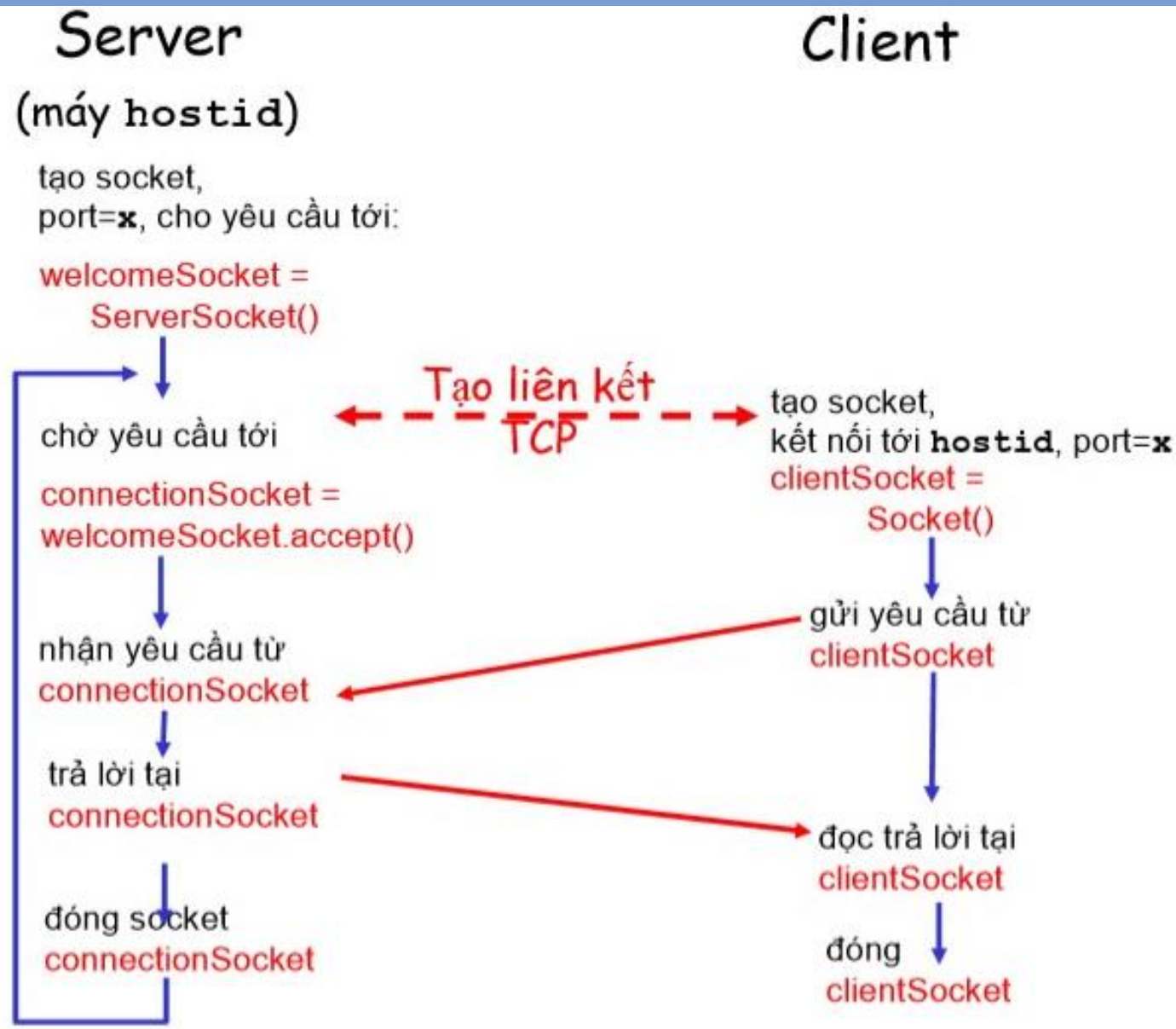
GIAO TIẾP BẰNG SOCKET (1)

Tạo server socket (Server)

Tạo client socket (Client)

Yêu cầu kết nối từ Client đến Server

Gởi và nhận dữ liệu giữa Client và Server



1. Tạo Server

```
public static void main(String[] args) {
    try {
        // Khởi tạo một ServerSocket nghe kết nối từ client tại cổng 12345
        ServerSocket serverSocket = new ServerSocket(12345);
        System.out.println("Server is listening on port 12345...");

        // Chấp nhận kết nối từ client
        Socket socket = serverSocket.accept();
        System.out.println("Client connected");

        // Tạo các luồng để nhận và gửi dữ liệu
        BufferedReader input = new BufferedReader(new InputStreamReader(socket.getInputStream()));
        PrintWriter output = new PrintWriter(socket.getOutputStream(), true);

        String clientMessage;
        while ((clientMessage = input.readLine()) != null) {
            System.out.println("Received from client: " + clientMessage);
            output.println("Echo: " + clientMessage);  // Echo lại thông điệp cho client
        }

        socket.close();
        serverSocket.close();
    } catch (IOException e) {
    }
}
```

2. Tạo Client

```
try {
    // Kết nối tới server tại địa chỉ localhost và cổng 12345
    Socket socket = new Socket("localhost", 12345);

    // Tạo các luồng để gửi và nhận dữ liệu
    BufferedReader input = new BufferedReader(new InputStreamReader(System.in));
    PrintWriter output = new PrintWriter(socket.getOutputStream(), true);
    BufferedReader serverInput = new BufferedReader(new InputStreamReader(socket.getInputStream()));

    String message;
    while (true) {
        System.out.print("Enter message: ");
        message = input.readLine();

        // Gửi tin nhắn tới server
        output.println(message);

        // Nhận phản hồi từ server
        String serverMessage = serverInput.readLine();
        System.out.println("Server says: " + serverMessage);

        if (message.equalsIgnoreCase("exit")) {
            break;
        }
    }
    socket.close();
} catch (IOException e) {
}
```

GIAO TIẾP BẰNG SOCKET (4)

tput - SocketDemo (run) × Server

```
run:
Server is listening on port 12345...
```

Client kết nối đến Server

SocketDemo (run) × Server

```
run:
Server is listening on port 12345...
Client connected
```

SocketDemo (run) #2 × Client

```
run:
Enter message: I love Viet Nam
Server says: Echo: I love Viet Nam
Enter message:
```

SocketDemo (run) × Server

```
run:
Server is listening on port 12345...
Client connected
Received from client: I love Viet Nam
```

TRUYỀN TẢI DỮ LIỆU VỚI UDP

- UDP (User Datagram Protocol) là một giao thức không kết nối (connectionless) và không đảm bảo thứ tự, độ tin cậy khi truyền tải dữ liệu.
- Các gói tin UDP có thể bị mất hoặc đến không theo đúng thứ tự
- Có tốc độ nhanh vì không cần phải thiết lập kết nối trước và không có quá trình xác nhận dữ liệu như trong TCP.
- Được sử dụng trong các ứng dụng cần hiệu suất cao và chấp nhận mất gói tin khi truyền tải (truyền phát video, các trò chơi trực tuyến, hoặc các dịch vụ thời gian thực, …)
- Được hỗ trợ bởi `DatagramSocket` và `DatagramPacket` trong gói java.net.

"DatagramSocket" CLASS (1)

- Được sử dụng để gửi và nhận các datagram
- Đại điện cho client và server
- Khởi tạo:
  - `DatagramSocket()`
  - `DatagramSocket(int port)`
  - `DatagramSocket(int port, InetAddress laddr)`

"DatagramSocket" CLASS (2)

Các phương thức:

- **send**(DatagramPacket p): gởi 1 datagram.
- **receive**(DatagramPacket p): nhận 1 datagram.
- **setSoTimeout**(int timeout): thiết lập thời gian timeout (milliseconds), giới hạn thời gian đợi để nhận dữ liệu. Đối tượng SocketTimeoutException được sinh ra nếu đếnthời gian timeout
- **close**(): đóng socket.

DATAGRAM

- Dữ liệu truyền tải được đóng gói thành gói tin, gọi là “datagram”.
- Datagram:
  - Là một message đọc lập, tự mô tả
  - Được gởi đi từ máy này sang máy khác thông qua môi trường mạng
  - Không đảm bảo về thời gian đến và nội dung đến trong quá trình truyền tải

"DatagramPacket" CLASS

- Đại diện 1 một datagram (gói tin UDP)
- Là một mảng byte chứa:
  - Dữ liệu cần truyền tải
  - Thông tin truyền tải: địa chỉ, cổng (port) .
- Khởi tạo:
  - `DatagramPacket(byte[] buf, int length)`
  - `DatagramPacket(byte[] buf, int length, InetAddress address, int port)`

UDP CLIENT SERVER (1)

UDP SERVER

```
DatagramSocket serverSocket = new DatagramSocket( port: Integer.parseInt( s: "5001"));
System.out.println("Server Started. Listening for Clients on port 5001" + "...");
// Assume messages are not over 1024 bytes
byte[] receiveData = new byte[1024];
DatagramPacket receivePacket;
while (true) {
  // Server waiting for clients message
   receivePacket = new DatagramPacket( buf: receiveData,  length: receiveData.length);
   serverSocket.receive( p: receivePacket);
  // Get the client's IP address and port
  InetAddress IPAddress = receivePacket.getAddress();
  int port = receivePacket.getPort();
  // Convert Byte Data to String
  String clientMessage = new String( bytes: receivePacket.getData(), offset: 0, length: receivePacket.getLength());
  // Print the message with log header
  long time = System.currentTimeMillis();
  Timestamp timestamp = new Timestamp(time);
  System.out.println("[" + timestamp.toString() + " ,IP: " + IPAddress + " ,Port: " + port +"]   " + clientMessage);
}
```

UDP CLIENT SERVER (2)

UDP CLIENT

```
DatagramPacket sendPacket;
 byte[] sendData;
 // Create a Datagram Socket
 DatagramSocket clientSocket = new DatagramSocket();
 // Set client timeout to be 1 second
 clientSocket.setSoTimeout( timeout: 1000);
 Scanner input = new Scanner( source: System.in);
 while (true) {
   String cmd = input.nextLine();
   // If client types quit, close the socket and exit
   if (cmd.equals( anObject: "QUIT")) {
     clientSocket.close();
     System.exit( status: 1);
   }
   sendData = cmd.getBytes();
   sendPacket = new DatagramPacket( buf: sendData,  length: sendData.length,
                address: InetAddress.getByName( host: "127.0.0.1"),  port: 5001);
   clientSocket.send( p: sendPacket);
 }
```

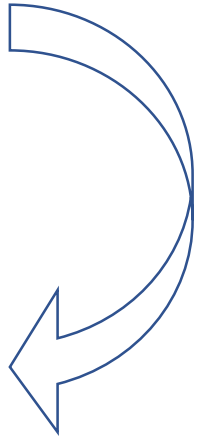
UDP CLIENT SERVER (3)

Run (Server)

```
JAVA_HOME="C:\Program Files\Java\jdk-17.0.5"
cd D:\OTML\advance\app; ..\gradlew.bat --configure-on-
Configuration on demand is an incubating feature.
> Task :app:compileJava
> Task :app:processResources NO-SOURCE
> Task :app:classes

> Task :app:runSingle
Server Started. Listening for Clients on port 5001...
```

Run (Client)

```
JAVA_HOME="C:\Program Files\Ja
cd D:\OTML\advance\app; ..\gra
Configuration on demand is an
> Task :app:compileJava
> Task :app:processResources N
> Task :app:classes
> Task :app:runSingle
Hello
I am Viet Nam
```



Run (Server)

```
JAVA_HOME="C:\Program Files\Java\jdk-17.0.5"
cd D:\OTML\advance\app; ..\gradlew.bat --configure-on-demand -PrunCla
Configuration on demand is an incubating feature.
> Task :app:compileJava
> Task :app:processResources NO-SOURCE
> Task :app:classes

> Task :app:runSingle
Server Started. Listening for Clients on port 5001...
[2024-11-24 15:17:49.254 ,IP: /127.0.0.1 ,Port: 61528]  Hello
[2024-11-24 15:18:15.26 ,IP: /127.0.0.1 ,Port: 61528]  I am Viet Nam
```

CÂU HỎI