JAVA THREAD

1. Synchronous

- Phân biệt synchronous vs asynchronous

**Synchronous (Đồng bộ)**

* Các tác vụ chạy **tuần tự**, cái trước xong mới đến cái sau.
* Thread **bị block/chờ** đến khi tác vụ kết thúc.
* Dễ viết, dễ debug, kết quả dự đoán được.

**Asynchronous (Bất đồng bộ)**

* Các tác vụ có thể **chạy song song** (multi-thread) hoặc được xử lý **trong nền**.
* Thread chính **không bị block**, có thể làm việc khác trong khi chờ kết quả.
* Giúp tận dụng CPU, cải thiện hiệu năng.

- Phân biệt trường hợp sử dụng, ưu nhược điểm của async và sync

| **Hình thức** | **Ưu điểm** | **Nhược điểm** | **Trường hợp dùng** |
| --- | --- | --- | --- |
| **Synchronous** | Dễ viết, dễ debug, kết quả có thứ tự rõ ràng | Chậm, block thread, không tận dụng CPU | Khi xử lý ít tác vụ, hoặc logic cần tuần tự (ghi file, xử lý transaction) |
| **Asynchronous** | Nhanh, tận dụng CPU, thích hợp nhiều tác vụ I/O | Phức tạp hơn, khó debug, dễ gặp race condition | Web server xử lý nhiều request, gọi API song song, tác vụ nền |

- Tìm hiểu từ khóa : **synchronized**trong java

 Dùng để **đồng bộ hóa** khi nhiều thread truy cập chung một tài nguyên (biến, object).

 Đảm bảo tại một thời điểm chỉ **một thread** được truy cập vào khối code/ method synchronized.

 Ngăn chặn **race condition** và **data inconsistency**.

public synchronized void increment() {

counter++;

}

2. Thread

- Process là gì ?

 **Process (tiến trình)** là một chương trình đang được thực thi trên hệ điều hành.

 Mỗi process bao gồm:

* **Không gian địa chỉ bộ nhớ riêng biệt** (gồm Heap, Stack, Data segment).
* **Các tài nguyên hệ thống** đi kèm như file descriptor, socket, connection đến cơ sở dữ liệu.
* Ít nhất một **luồng chính (main thread)** để thực thi các chỉ thị của chương trình.

 Các process **không chia sẻ trực tiếp bộ nhớ** với nhau. Nếu cần trao đổi thông tin, chúng phải sử dụng cơ chế **Inter-Process Communication (IPC)** như pipe, socket, hoặc message queue.

- Thread là gì ?

 **Thread (luồng)** là đơn vị xử lý nhỏ nhất của CPU, tồn tại bên trong một process.

 Các thread trong cùng một process sẽ:

* **Chia sẻ tài nguyên chung**: heap, dữ liệu tĩnh, vùng code.
* **Có không gian stack riêng**: dùng để quản lý lời gọi hàm và biến cục bộ.

 Khi một ứng dụng Java khởi chạy, JVM mặc định sẽ tạo ra **main thread** để thực thi phương thức main().

- Có bao nhiêu cách để tạo 1 thread trong java ? Khác biệt giữa việc sử dụng cách cách đó gì ?

Trong Java, có ba cách phổ biến để tạo và thực thi thread:

**1. Kế thừa từ lớp Thread**

* Ghi đè phương thức run(), sau đó gọi start().
* Nhược điểm: Java không hỗ trợ đa kế thừa, nên nếu đã kế thừa lớp khác thì không thể kế thừa Thread.

vd:

class MyThread extends Thread {

@Override

public void run() {

System.out.println("Running thread");

}

}

new MyThread().start();

**2. Triển khai giao diện Runnable**

* Tách riêng logic xử lý (trong run()) khỏi cơ chế thực thi (Thread).
* Linh hoạt hơn vì vẫn có thể kế thừa lớp khác.

class MyRunnable implements Runnable {

@Override

public void run() {

System.out.println("Running via Runnable");

}

}

new Thread(new MyRunnable()).start();

**3. Sử dụng ExecutorService và Callable (cách hiện đại)**

* Quản lý luồng qua thread pool → tránh việc tạo quá nhiều thread không kiểm soát.
* Callable hỗ trợ trả về kết quả và ném checked exception.

ExecutorService executor = Executors.newFixedThreadPool(2);

Future<String> future = executor.submit(() -> "Result from thread");

System.out.println(future.get()); // chờ lấy kết quả

executor.shutdown();

- Thế nào là multi thread ? Sử dụng multi thread mang lại ưu nhược điểm gì ?

**Định nghĩa**

* **Multi-threading** là kỹ thuật trong đó một process có thể chạy đồng thời nhiều thread để thực hiện các tác vụ song song.
* Các thread cùng chia sẻ tài nguyên của process, nhờ đó hiệu quả xử lý cao hơn so với việc chạy nhiều process độc lập.

**Ưu điểm**

* **Tận dụng tối đa CPU đa nhân**: các thread có thể thực thi trên các core khác nhau.
* **Hiệu quả trong tác vụ I/O**: trong khi một thread chờ đọc/ghi dữ liệu, thread khác vẫn có thể tiếp tục chạy.
* **Phản hồi nhanh hơn**: ứng dụng có thể xử lý song song nhiều yêu cầu (ví dụ: server web).

**Nhược điểm**

* **Lập trình phức tạp**: phải xử lý các vấn đề như race condition, deadlock, starvation.
* **Khó debug và test**: do trạng thái thread thay đổi liên tục.
* **Chi phí context switching**: khi CPU chuyển đổi giữa các thread, hiệu năng bị giảm nếu số lượng thread quá lớn.

- Làm thế nào để biết được 1 thread, multi thread đã hoàn thành hay chưa?

**Phương thức join()**: cho phép một thread chờ đến khi thread khác kết thúc.

**Sử dụng Future.get()** (khi dùng ExecutorService): chặn (block) cho đến khi có kết quả trả về.

- Có giới hạn việc tạo ra bao nhiêu thread trong 1 ứng dụng java hay không?

 Java **không đặt giới hạn cứng** về số lượng thread có thể tạo ra.

 Số thread tối đa phụ thuộc vào:

* **Bộ nhớ RAM**: mỗi thread cần stack riêng (thường 512 KB – 1 MB).
* **Cấu hình JVM**: tham số -Xss (stack size).
* **Hệ điều hành**: mỗi OS có giới hạn về số thread.

 Thực tế, nếu tạo quá nhiều thread, hiệu năng sẽ giảm nghiêm trọng do **context switching**.

 **Giải pháp khuyến nghị**: sử dụng **Thread Pool (ExecutorService)** để kiểm soát số lượng thread.