# Process

Process (tiến trình) là một chương trình đang được thực thi. Nó bao gồm mã chương trình, dữ liệu và trạng thái của chương trình đó. Mỗi process hoạt động độc lập và có không gian bộ nhớ riêng biệt. Hệ điều hành quản lý các tiến trình để đảm bảo rằng chúng hoạt động một cách hiệu quả và không gây ra xung đột về tài nguyên.

# Thread

Thread (luồng) là một đơn vị nhỏ hơn của tiến trình. Một process có thể bao gồm nhiều thread, và tất cả chúng chia sẻ cùng một không gian bộ nhớ của tiến trình đó. Điều này cho phép các thread trong cùng một process giao tiếp và chia sẻ dữ liệu với nhau một cách dễ dàng hơn so với các process khác.

# Cách để tạo 1 thread trong Java

1. **Kế thừa class Thread**

* Tạo một class con kế thừa từ class Thread và ghi đè phương thức run().
* Sau đó, tạo một object của class con và gọi phương thức start() để bắt đầu thread.
* Không thể kế thừa từ class khác vì Java không hỗ trợ đa kế thừa.
* Ví dụ:

class MyThread extends Thread {

public void run() {

// Code

}

}

MyThread thread = new MyThread();

thread.start();

1. **Realizing Runnable Interface**

* Tạo một lớp thực hiện giao diện Runnable và ghi đè phương thức run().
* Tạo một đối tượng Thread và truyền đối tượng Runnable vào nó, sau đó gọi start() trên đối tượng Thread.
* Co thể kế thừa từ một class khác đồng thời thực hiện Runnable, và cho phép tách biệt logic của thread khỏi class thực thi.
* Ví dụ:

class MyRunnable implements Runnable {

public void run() {

// Code

}

}

Thread thread = new Thread(new MyRunnable());

thread.start();

# Multi-thread

1. **Khái niệm**

Là khả năng của một ứng dụng thực hiện nhiều thread đồng thời. Điều này cho phép thực hiện nhiều tác vụ song song, cải thiện hiệu suất và khả năng đáp ứng của ứng dụng.

1. **Ưu điểm**

* **Tăng hiệu suất**: Sử dụng tài nguyên CPU hiệu quả hơn bằng cách thực hiện nhiều tác vụ cùng lúc.
* **Cải thiện khả năng đáp ứng**: Các thread có thể xử lý các tác vụ nền mà không làm gián đoạn trải nghiệm của người dùng.
* **Tối ưu hóa tài nguyên**: Thread trong cùng một process chia sẻ tài nguyên, giảm chi phí cho việc tạo và quản lý thread.

1. **Nhược điểm**

* **Khó khăn trong lập trình**: Việc lập trình với nhiều thread có thể phức tạp hơn, đặc biệt là khi xử lý đồng bộ hóa.
* **Lỗi race condition**: Khi nhiều thread cố gắng truy cập và thay đổi cùng một tài nguyên, có thể xảy ra lỗi race condition nếu không được quản lý đúng cách.
* **Chi phí quản lý**: Quá nhiều thread có thể làm tăng chi phí cho việc quản lý chúng.

1. **Nhận biết 1 thread, multi-thread đã hoàn thành hay chưa**

* Để kiểm tra xem một thread đã hoàn thành hay chưa, có thể sử dụng phương thức isAlive() của lớp Thread. Nếu phương thức trả về false, điều đó có nghĩa là thread đã hoàn thành.

if (!thread.isAlive()) {

// Code

}

* Ngoài ra, có thể sử dụng phương thức join() để chờ cho một thread hoàn thành. Phương thức này sẽ làm cho thread hiện tại tạm dừng cho đến khi thread được gọi hoàn thành.

thread.join();

1. **Giới hạn tạo thread trong 1 ứng dụng Java**

* **Bộ nhớ**: Mỗi thread cần một không gian bộ nhớ nhất định, và nếu tạo quá nhiều thread, có thể gặp lỗi OutOfMemoryError.
* **Tài nguyên hệ thống**: Số lượng thread có thể bị giới hạn bởi hệ thống điều hành, vì mỗi thread yêu cầu một lượng tài nguyên từ hệ thống.