Synchronous vs Asynchronous

**1. Phân biệt Synchronous và Asynchronous**

**Synchronous (Đồng bộ):**

* **Synchronous** có nghĩa là các tác vụ được thực hiện theo trình tự tuần tự. Một tác vụ chỉ được bắt đầu khi tác vụ trước đó đã hoàn thành.
* Trong lập trình, khi một hàm được gọi trong một quá trình **synchronous**, chương trình sẽ phải chờ hàm này hoàn thành rồi mới tiếp tục thực hiện các bước tiếp theo.

Vd:

System.out.println("Bắt đầu");

Thread.sleep(2000); // Chờ 2 giây

System.out.println("Kết thúc");

**Asynchronous (Bất đồng bộ):**

* **Asynchronous** có nghĩa là các tác vụ có thể được thực hiện song song hoặc không cần phải chờ tác vụ trước đó hoàn thành. Chương trình có thể tiếp tục thực hiện các tác vụ khác trong khi chờ một tác vụ chưa hoàn tất.
* Trong lập trình bất đồng bộ, hàm được gọi có thể không cần phải hoàn thành ngay lập tức, mà nó sẽ được thực hiện trong background, và chương trình sẽ không bị "chặn".

**Ví dụ** (với Java và lập trình multi-threading):

System.out.println("Bắt đầu");

new Thread(() -> {

try {

Thread.sleep(2000); // Chờ 2 giây trong một thread khác

} catch (InterruptedException e) {

e.printStackTrace();

}

System.out.println("Tác vụ bất đồng bộ");

}).start();

System.out.println("Kết thúc");**2. Phân biệt trường hợp sử dụng, ưu nhược điểm của Synchronous và Asynchronous**

**Synchronous:**

* **Trường hợp sử dụng**:
  + Khi cần đảm bảo các tác vụ được thực hiện theo thứ tự, và không có sự bất đồng bộ.
  + Các tác vụ nhỏ, nhanh gọn, không tiêu tốn nhiều thời gian.
* **Ưu điểm**:
  + Đơn giản, dễ hiểu, dễ quản lý vì các tác vụ được thực hiện tuần tự.
  + Phù hợp với các tác vụ mà không cần phải chờ đợi hoặc có độ trễ thấp.
* **Nhược điểm**:
  + Nếu một tác vụ mất nhiều thời gian để hoàn thành (ví dụ: I/O, mạng), toàn bộ chương trình sẽ bị chặn lại, gây mất hiệu quả.
  + Không tối ưu cho các hệ thống cần thực hiện nhiều tác vụ đồng thời hoặc phải xử lý các tác vụ mất nhiều thời gian.

**Asynchronous:**

* **Trường hợp sử dụng**:
  + Khi có các tác vụ dài, như gọi API, truy vấn cơ sở dữ liệu, hoặc xử lý các tác vụ cần thời gian (I/O, đọc/ghi file).
  + Khi cần thực hiện nhiều tác vụ song song để cải thiện hiệu suất, tránh tắc nghẽn.
* **Ưu điểm**:
  + Tăng hiệu suất của ứng dụng, vì có thể thực hiện nhiều tác vụ cùng lúc.
  + Không chặn luồng chính, giúp giao diện người dùng (UI) không bị "đóng băng" trong các ứng dụng desktop hoặc web.
* **Nhược điểm**:
  + Phức tạp hơn trong việc lập trình và quản lý, vì phải xử lý các callback, promise, hoặc đồng bộ hóa giữa các tác vụ.
  + Debug và xử lý lỗi trong các chương trình bất đồng bộ có thể khó khăn hơn.

**3. Tìm hiểu từ khóa synchronized trong Java**

**synchronized trong Java:**

* Từ khóa **synchronized** được sử dụng để đảm bảo rằng chỉ một luồng (thread) có thể truy cập vào một khối mã hoặc một phương thức tại một thời điểm, nhằm tránh xảy ra các tình huống "race condition" khi nhiều luồng cùng truy cập và thay đổi dữ liệu đồng thời.
* Khi một phương thức hoặc khối mã được đánh dấu là **synchronized**, bất kỳ luồng nào muốn thực hiện nó sẽ phải "giữ khóa" (lock) đối tượng đó trước khi được phép tiếp tục. Các luồng khác sẽ phải chờ cho đến khi luồng hiện tại hoàn thành và giải phóng khóa.

**Cách sử dụng synchronized:**

1. **Synchronized method**: Đảm bảo rằng toàn bộ phương thức chỉ có một thread truy cập vào.

public synchronized void increment() {

count++;

}

Trong ví dụ này, nếu nhiều thread cùng gọi phương thức increment, chỉ một thread sẽ được phép thực thi tại một thời điểm.

2. **Synchronized block**: Cho phép khóa một đoạn mã cụ thể trong phương thức.

public void increment() {

synchronized(this) {

count++;

}

}

**Ưu điểm của synchronized:**

* Đảm bảo tính toàn vẹn của dữ liệu khi có nhiều thread cùng thao tác trên các đối tượng chia sẻ.
* Tránh "race condition" và các vấn đề liên quan đến đồng bộ hóa dữ liệu.

**Nhược điểm của synchronized:**

* Giảm hiệu suất, vì khi một thread giữ khóa, các thread khác phải đợi, điều này có thể dẫn đến hiện tượng **"blocking"** hoặc **"deadlock"** nếu không được xử lý tốt.
* Độ phức tạp tăng khi có nhiều thread cùng sử dụng synchronized trên nhiều đối tượng.