# Phân biệt synchronous (đồng bộ) và asynchronous (bất đồng bộ)

1. **Synchronous (Đồng bộ)**

* Các tác vụ trong chương trình được thực hiện tuần tự, tức là mỗi tác vụ sẽ phải chờ tác vụ trước đó hoàn thành trước khi tiếp tục.
* Ví dụ: Nếu gọi một API trong chương trình đồng bộ, chương trình sẽ đợi API phản hồi trước khi tiếp tục công việc khác.

1. **Asynchronous (Bất đồng bộ)**

* Các tác vụ có thể được thực hiện song song, không cần phải chờ tác vụ khác hoàn thành. Thay vì đợi, chương trình có thể thực hiện công việc khác trong khi chờ phản hồi từ tác vụ.
* Ví dụ: Khi gọi một API trong chương trình bất đồng bộ, chương trình sẽ tiếp tục thực hiện các công việc khác trong khi đợi API phản hồi.

# sync và async

1. **sync**

* **Ưu điểm:**
* Dễ hiểu và dễ triển khai hơn do mọi thứ được thực hiện theo thứ tự tuần tự.
* Phù hợp với các ứng dụng yêu cầu thực hiện các tác vụ tuần tự, không cần thực hiện nhiều tác vụ song song.
* **Nhược điểm:**
* Hiệu suất không cao nếu có nhiều tác vụ mất thời gian vì chương trình phải đợi từng tác vụ hoàn thành.
* Dễ dẫn đến **bottleneck** (nghẽn cổ chai) trong các tác vụ I/O (đọc ghi tệp, gọi API).
* **Trường hợp sử dụng:**
* Khi cần sự đồng bộ chặt chẽ giữa các bước, chẳng hạn như khi các bước phụ thuộc vào kết quả của nhau.
* Trong các ứng dụng đơn giản, không có yêu cầu xử lý nhiều tác vụ song song.

1. **async**

* **Ưu điểm:**
* Tăng hiệu suất của hệ thống do có thể thực hiện nhiều tác vụ đồng thời mà không cần chờ nhau.
* Phù hợp cho các tác vụ I/O (ví dụ: đọc ghi dữ liệu từ cơ sở dữ liệu, gọi API, xử lý tệp tin lớn).
* **Nhược điểm:**
* Phức tạp hơn để triển khai và quản lý, vì phải xử lý các callback, promise hoặc async/await.
* Việc kiểm soát lỗi và duy trì tính nhất quán khó hơn so với sync.
* **Trường hợp sử dụng:**
* Khi cần tối ưu hiệu suất của các tác vụ có thể thực hiện đồng thời (chẳng hạn khi xử lý nhiều request đồng thời trong ứng dụng web).
* Trong các ứng dụng real-time, nơi yêu cầu phản hồi nhanh và không muốn bị chặn bởi các tác vụ mất thời gian.

# synchronized trong Java

* synchronized được sử dụng để đảm bảo rằng một đoạn mã cụ thể chỉ được thực thi bởi một luồng (thread) tại một thời điểm. Điều này ngăn chặn hai hoặc nhiều luồng truy cập cùng một tài nguyên (biến, phương thức) cùng lúc, từ đó tránh được các vấn đề về data race (xung đột dữ liệu) và inconsistent state (trạng thái không nhất quán).
* Khi một phương thức hoặc một khối mã được đánh dấu là synchronized, nếu một luồng (thread) đang thực thi khối mã đó, thì các luồng khác phải chờ đến khi luồng đầu tiên hoàn thành công việc của nó trước khi truy cập vào đoạn mã đó.
* **Ưu điểm:** Đảm bảo tính nhất quán của dữ liệu khi có nhiều luồng cùng truy cập và thay đổi dữ liệu.
* **Nhược điểm:** Giảm hiệu suất nếu có quá nhiều luồng bị khóa, chờ đợi để truy cập tài nguyên. Có thể xảy ra deadlock (khoá chết) nếu không được xử lý đúng cách.