**1.Synchronous vs Asynchronous:**

**a.Synchronous(Đồng bộ):**

* Là khi một tác vụ được thực hiện tuần tự. Quá trình thực hiện sẽ không chuyển sang tác vụ tiếp theo cho đến khi tác vụ hiện tại hoàn thành.
* Mô hình này dễ hiểu và đơn giản, nhưng khi tác vụ nào đó tốn thời gian (như truy vấn dữ liệu từ cơ sở dữ liệu hay đọc file lớn), chương trình sẽ bị "chặn" và không thể thực hiện bất kỳ tác vụ nào khác cho đến khi nó hoàn thành.

**b.Asynchronous (Bất đồng bộ)**:

* Là khi một tác vụ được thực hiện mà không cần đợi tác vụ trước đó hoàn thành. Khi một tác vụ bất đồng bộ được kích hoạt, chương trình có thể thực hiện các tác vụ khác trong khi chờ kết quả từ tác vụ bất đồng bộ.
* Điều này giúp cải thiện hiệu suất, đặc biệt khi bạn phải thực hiện các thao tác tốn thời gian như gọi API hay xử lý file lớn.
* Bất đồng bộ cho phép chương trình tiếp tục chạy mà không bị "chặn"

**2.Các trường hợp sử dụng:**

**a.Synchronous:**

**Trường hợp sử dụng:**

* Khi các tác vụ cần được thực hiện theo thứ tự, tức là kết quả của tác vụ trước phụ thuộc vào tác vụ sau.
* Khi không có các tác vụ tốn thời gian cần xử lý, hoặc môi trường không hỗ trợ tốt bất đồng bộ.

**Ưu điểm:**

* Dễ triển khai và dễ hiểu.
* Không cần phải quản lý trạng thái hoặc xử lý các callback phức tạp

**Nhược điểm:**

* Tốn thời gian chờ khi xử lý các tác vụ như truy vấn csdl,gọi API,…
* Gây ra blocking ảnh hưởng đến hiệu xuất tổng thể khi có các tác vụ lâu hoàn thành.

**b.Asynchronous:**

**Trường hợp sử dụng:**

* Khi bạn phải xử lý các tác vụ lâu hoặc có thể bị chặn như đọc file, gọi API, xử lý dữ liệu từ cơ sở dữ liệu.
* Khi cần thực hiện nhiều tác vụ đồng thời và cải thiện hiệu suất hệ thống.

**Ưu điểm:**

* Cải thiện hiệu suất tổng thể của hệ thống bằng cách không chặn các tác vụ khác trong khi chờ xử lý.
* Phù hợp cho hệ thống yêu cầu mở rộng cao và cần xử lý số lượng lớn tác vụ đồng thời

**Nhược điểm:**

* Phức tạp hơn khi quản lý trạng thái,callback,hoặc điều phối tác vụ
* Cần phải cẩn trọng trong việc xử lý lỗi và quản lý đồng bộ hóa dữ liệu khi nhiều tác vụ thực thi đồng thời.

**3.Từ khóa synchronized trong java?**

**synchronized** là từ khóa trong Java được sử dụng để đảm bảo rằng chỉ có một thread có thể truy cập vào một khối mã hoặc phương thức cụ thể tại một thời điểm, nhằm đảm bảo tính toàn vẹn của dữ liệu trong trường hợp nhiều thread truy cập cùng lúc.

**Cách sử dụng:**

Có thể sử dụng với phương thức hoặc mã khối:

* **Synchronized method:** Khi một thread gọi một phương thức được đánh dấu synchronized,nó sẽ khóa đối tượng mà phương thức này thuộc về hoặc khóa lớp.
* **Synchronized block:**Khối mã được đánh dấu synchronized sẽ khóa một đối tượng cụ thể.Điều này sẽ giúp thu hẹp phạm vi của phần mã được khóa,tối ưu hơn so với việc khóa cả phương thức

**Ưu điểm:**

* Giúp đảm bảo thread safety(an toàn cho thread) bằng cách ngăn chặn nhiều thread cùng truy cập vào tài nguyên chung đồng thời

**Nhược điểm:**

* Hiệu suất có thể bị giảm vì việc sử dụng synchronized có thể dẫn đến các tình huống chờ đợi (blocking) khi nhiều thread cùng cố gắng truy cập vào tài nguyên cùng lúc.
* Có thể gây ra deadlock nếu không sử dụng cẩn thận.

**Process là gì?**

**Process** là một chương trình đang được thực thi, bao gồm mã chương trình, dữ liệu, và các tài nguyên khác cần thiết để thực thi chương trình. Mỗi process được quản lý độc lập bởi hệ điều hành và có không gian bộ nhớ riêng. Khi bạn mở một ứng dụng, hệ điều hành sẽ tạo một process cho ứng dụng đó.

**Thread là gì?**

**Thread** là đơn vị thực thi nhỏ nhất trong một process. Mỗi process có thể có nhiều thread chạy song song, chia sẻ cùng không gian bộ nhớ của process đó. Thread giúp tăng hiệu suất và tính đồng thời của ứng dụng. Một thread có thể được mô tả như một dòng lệnh thực thi, trong khi một process có thể chứa nhiều dòng lệnh này.

**Các cách để tạo luồng trong Java:**

**1. Kế thừa lớp Thread:**

-Ta có thể tạo một lớp mới kế thừa từ lớp Thread và ghi đè phương thức **run().**

**2.Triển khai giao diện Runnable:**

-Ta có thể tạo một lớp mới triển khai giao diện Runnable và định nghĩa phương thức **run()**.nhưng có kiểu là void

**3.Triển khai giao diện Callable:**

-Ta cũng có thể tạo một triển khai giao diện khác là Callable và triển khai phương thức Call(),có thể có các kiểu trả về khác và có thể xử lý ngoại lệ checked và unchecked

**Multithreading là gì?**

-Đây là một kỹ thuật trong lập trình cho phép một ứng dụng thực thi nhiều luồng (thread) cùng một lúc. Mỗi thread là một đơn vị nhỏ hơn của quá trình (process) và có thể thực hiện các tác vụ khác nhau mà không ảnh hưởng đến các thread khác trong cùng một ứng dụng

**Ưu điểm của Multithreading:**

1. **Hiệu suất cao**:
   * Multithreading cho phép tận dụng tối đa tài nguyên CPU, đặc biệt là trong các hệ thống đa nhân. Nhiều thread có thể thực hiện song song, dẫn đến cải thiện hiệu suất và tốc độ xử lý.
2. **Phản hồi nhanh hơn**:
   * Trong ứng dụng GUI, multithreading giúp giao diện người dùng luôn phản hồi mượt mà, cho phép người dùng tiếp tục tương tác trong khi một hoặc nhiều tác vụ khác đang được xử lý ở phía sau.
3. **Tăng tính hiệu quả**:
   * Khi một thread bị chặn (blocked) do các tác vụ I/O (như đọc hoặc ghi dữ liệu), các thread khác có thể tiếp tục thực hiện, giúp ứng dụng hoạt động hiệu quả hơn.
4. **Quản lý dễ dàng hơn**:
   * Với multithreading, việc chia nhỏ một tác vụ lớn thành nhiều phần nhỏ và thực hiện chúng song song có thể làm cho mã dễ đọc và bảo trì hơn.

**Nhược điểm của Multithreading:**

1. **Phức tạp trong lập trình**:
   * Việc quản lý nhiều thread đồng thời có thể phức tạp hơn, dẫn đến khó khăn trong việc kiểm soát và duy trì mã. Tình trạng race condition, deadlock và livelock có thể xảy ra nếu không quản lý thread một cách chính xác.
2. **Tốn kém tài nguyên**:
   * Mặc dù thread nhẹ hơn process, nhưng việc tạo quá nhiều thread vẫn có thể tốn tài nguyên, bao gồm bộ nhớ và thời gian CPU cho việc chuyển đổi giữa các thread.
3. **Khó khăn trong gỡ lỗi**:
   * Các lỗi liên quan đến multithreading thường khó tìm và gỡ bỏ hơn vì chúng có thể xảy ra không thường xuyên và phụ thuộc vào trạng thái đồng thời của các thread.
4. **Chia sẻ dữ liệu**:
   * Khi nhiều thread chia sẻ dữ liệu, cần phải có cơ chế đồng bộ hóa (synchronization) để tránh xung đột và bảo vệ tính toàn vẹn của dữ liệu. Việc đồng bộ hóa không đúng cách có thể dẫn đến các vấn đề về hiệu suất hoặc lỗi.

**Để kiểm tra xem một thread hoặc một nhóm thread (multithreading) đã hoàn thành hay chưa trong Java, bạn có thể sử dụng một số phương pháp khác nhau. Dưới đây là các cách phổ biến:**

### 1. ****Sử dụng phương thức**** isAlive() ****của Thread****

Phương thức isAlive() cho phép bạn kiểm tra xem một thread còn đang hoạt động hay không. Nó trả về true nếu thread còn sống (đang thực thi) và false nếu không.

2. **Sử dụng** join()

Phương thức join() của thread cho phép một thread (ví dụ: thread chính) chờ cho đến khi thread khác hoàn thành. Sau khi gọi join(), thread chính sẽ bị chặn cho đến khi thread đang chờ kết thúc.

### 3. ****Sử dụng Future với Callable****

Nếu bạn sử dụng Callable và ExecutorService, bạn có thể sử dụng đối tượng Future để kiểm tra xem một tác vụ đã hoàn thành hay chưa. Phương thức isDone() sẽ trả về true nếu tác vụ đã hoàn thành.

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

Trên lý thuyết là không có giới hạn,có thể tạo bao nhiêu luồng tùy thích nhưng trên thực tế cũng có những giới hạn nhất định,một vài yếu tố có thể kể đến:

-Giới hạn tài nguyên hệ thống như Memory,CPU

-Giới hạn JVM có thể có các cấu hình liên quan đến kích thước stack cho mỗi thread. Bạn có thể thay đổi kích thước stack bằng cách sử dụng tùy chọn -Xss khi khởi động JVM. Ví dụ: -Xss512k sẽ đặt kích thước stack cho mỗi thread là 512 KB.

-Hệ điều hành cũng có thể giới hạn số lượng thread mà một ứng dụng có thể tạo ra. Ví dụ, một số hệ điều hành có giới hạn về số lượng thread tối đa mà một tiến trình có thể tạo ra, thường là do giới hạn tài nguyên hệ thống.