# **Thread**

# **1. Các cách tạo thread?**

Có hai cách chính để tạo thread trong Java:

* Cách 1: Kế thừa từ lớp Thread và ghi đè phương thức run()
* Cách 2: Triển khai interface Runnable

# **2. Synchronous vs Asynchronous**

### **1. Synchronous (Đồng bộ)**

Trong lập trình đồng bộ, các tác vụ được thực hiện theo một thứ tự tuần tự. Khi một tác vụ được gọi, chương trình sẽ chờ tác vụ đó hoàn thành trước khi tiếp tục với tác vụ kế tiếp. Điều này có nghĩa là luồng thực thi sẽ bị chặn (blocked) cho đến khi tác vụ hoàn tất.

#### **Đặc điểm chính của Synchronous:**

* **Blocking (Chặn)**: Luồng thực thi bị chặn cho đến khi tác vụ hoàn thành.
* **Thứ tự tuần tự**: Các tác vụ phải được thực hiện theo thứ tự, nghĩa là không thể thực thi song song.
* **Đơn giản**: Dễ hiểu và dễ theo dõi trong các tình huống đơn luồng (single-threaded).
* **Hiệu năng thấp** trong các ứng dụng yêu cầu độ phản hồi cao hoặc có tác vụ tốn thời gian (như I/O hoặc mạng), vì luồng bị chặn trong suốt thời gian chờ.

### **2. Asynchronous (Bất đồng bộ)**

Ngược lại với lập trình đồng bộ, lập trình bất đồng bộ cho phép các tác vụ được thực hiện song song. Khi một tác vụ bất đồng bộ được gọi, chương trình không chờ nó hoàn thành mà tiếp tục thực thi tác vụ tiếp theo ngay lập tức. Tác vụ bất đồng bộ sẽ hoàn thành ở một thời điểm không xác định trong tương lai, thường thông qua callback, Future, hoặc CompletableFuture.

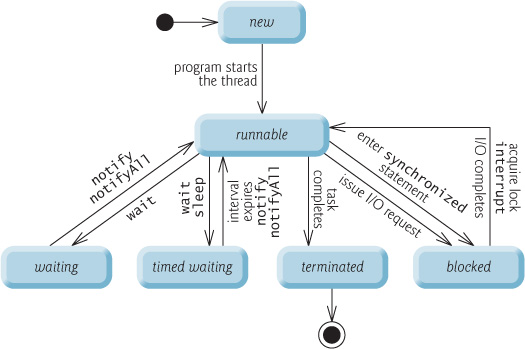
#### **Đặc điểm chính của Asynchronous:**

* **Non-blocking (Không chặn)**: Luồng thực thi không bị chặn, chương trình tiếp tục ngay lập tức.
* **Song song**: Các tác vụ có thể được thực thi song song với nhau mà không phải chờ đợi.
* **Hiệu năng cao** trong các ứng dụng yêu cầu nhiều tác vụ đồng thời (như ứng dụng mạng, xử lý I/O).
* **Phức tạp hơn** để xử lý do sự khó khăn trong việc quản lý trạng thái và luồng logic khi các tác vụ hoàn thành ở thời điểm khác nhau.

| Synchronous | Asynchronous |
| --- | --- |
| Luồng bị chặn cho đến khi tác vụ hoàn thành | Luồng không bị chặn, tiếp tục thực thi ngay lập tức |
| Thứ tự thực thi là tuần tự | Các tác vụ được thực hiện song song |
| Phù hợp cho các tác vụ ngắn và đơn giản | Phù hợp cho các tác vụ tốn thời gian hoặc nhiều tác vụ đồng thời |
| Dễ hiểu và triển khai | Khó quản lý hơn, yêu cầu xử lý đồng bộ hóa kết quả |
| Hiệu năng thấp trong các ứng dụng đòi hỏi phản hồi nhanh | Hiệu năng cao, tận dụng tài nguyên tốt hơn |

**3. Phân biệt trường hợp sử dụng, ưu nhược điểm của async và sync**

# **4. Vòng đời của thread**



* **NEW** : Đây là trạng thái khi luồng vừa được khởi tạo bằng phương thức khởi tạo của lớp Thread nhưng chưa được start(). Ở trạng thái này, luồng được tạo ra nhưng chưa được cấp phát tài nguyên và cũng chưa chạy. Nếu luồng đang ở trạng thái này mà ta gọi các phương thức ép buộc stop,resume,suspend … sẽ là nguyên nhân sảy ra ngoại lệ IllegalThreadStateException .
* **RUNNABLE** : Sau khi gọi phương thức start() thì luồng test đã được cấp phát tài nguyên và các lịch điều phối CPU cho luồng test cũng bắt đầu có hiệu lực. Ở đây, chúng ta dùng trạng thái là Runnable chứ không phải Running, vì luồng không thực sự luôn chạy mà tùy vào hệ thống mà có sự điều phối CPU khác nhau.
* **WAITING** : Thread chờ không giới hạn cho đến khi một luồng khác đánh thức nó.
* **TIMED\_WAITING** : Thread chờ trong một thời gian nhất định, hoặc là có một luồng khác đánh thức nó.
* **BLOCKED**: Đây là 1 dạng của trạng thái “Not Runnable”, là trạng thái khi Thread vẫn còn sống, nhưng hiện tại không được chọn để chạy. Thread chờ một monitor để unlock một đối tượng mà nó cần.
* **TERMINATED** : Một thread ở trong trạng thái terminated hoặc dead khi phương thức run() của nó bị thoát.

# **5. Một số thông tin liên quan đến luồng**

### **Định danh của luồng (ThreadId)**

ThreadId là định danh của luồng, nó dùng để phân biệt với các luồng khác cùng tiến trình hoặc cùng tập luồng. Đây là thông số mà máy ảo java tự tạo ra khi ta tạo luồng nên ta không thể sửa đổi cũng như áp đặt thông số này khi tạo luồng. Nhưng ta có thể lấy được nó thông qua phương thức getId() của lớp Thread

### **Tên của luồng (ThreadName)**

ThreadName là tên của luồng, đây là thuộc tính mà ta có thể đặt hoặc không đặt cho luồng. Nếu ta không đặt cho luồng thì máy ảo java sẽ tự đặt với quy tắc sau: “Thread-” + Thứ tự luồng được tạo ra, bắt đầu từ 0.

### **Độ ưu tiên của luồng (Priority)**

Như đã nói ở phần trước, mỗi luồng có 1 độ ưu tiên nhất định. Đây sẽ là thông số quyết định mức ưu tiên khi cấp phát CPU cho các luồng.

Trong java, đế đặt độ ưu tiên cho 1 luồng ta dùng phương thức: **void setPriority(int newPriority)**

* int newPriority : Là giá trị từ 1 đến 10.

Java có định nghĩa sẵn 3 mức ưu tiên chuẩn như sau:

* Thread.**MIN\_PRIORITY** (giá trị **01**)
* hread.**NORM\_PRIORITY** (giá trị **05**)
* Thread.**MAX\_PRIORITY** (giá trị **10**)

Để lấy độ ưu tiên của 1 luồng, ta dùng phương thức: **int getPriority()**

# 6. synchronized trong java

Từ khóa **synchronized** trong Java được sử dụng để đồng bộ hóa truy cập vào một khối mã hoặc phương thức, nhằm đảm bảo rằng chỉ có một **thread** được phép truy cập vào phần tài nguyên được đồng bộ hóa tại một thời điểm. Điều này giúp tránh các vấn đề liên quan đến **điều kiện đua** (race condition) trong các ứng dụng đa luồng (multi-threading), khi nhiều thread cùng lúc cố gắng truy cập và thay đổi dữ liệu dùng chung.

### **1. Cách hoạt động của synchronized**

Khi một thread muốn thực thi một đoạn mã được đồng bộ hóa, nó phải **giành được khóa (lock)** của đối tượng (hoặc lớp, trong trường hợp đồng bộ hóa tĩnh). Nếu một thread khác đang nắm giữ khóa, thread đó phải đợi đến khi khóa được giải phóng. Điều này giúp bảo vệ các tài nguyên dùng chung khỏi bị truy cập và thay đổi đồng thời từ nhiều thread.

### **2. Sử dụng synchronized trong Java**

Có hai cách để sử dụng synchronized:

1. **Đồng bộ hóa một phương thức (Method Synchronization)**.
2. **Đồng bộ hóa một khối mã (Block Synchronization)**.

#### **2.1. Đồng bộ hóa một phương thức**

Phương thức synchronized đảm bảo rằng chỉ có một thread được phép thực thi phương thức này tại một thời điểm, nếu nhiều thread cùng truy cập vào cùng một đối tượng.

Trong ví dụ trên, khi một thread gọi phương thức synchronizedMethod(), các thread khác muốn gọi phương thức này phải chờ đến khi thread hiện tại hoàn tất.

* Khi từ khóa synchronized được sử dụng với một **phương thức không tĩnh** (non-static), thread cần phải nắm giữ khóa của **đối tượng hiện tại** (instance of the class).
* Khi từ khóa synchronized được sử dụng với một **phương thức tĩnh** (static), thread cần phải nắm giữ khóa của **class object** (khóa ở cấp lớp).

#### **2.2. Đồng bộ hóa một khối mã**

Đôi khi, bạn chỉ cần đồng bộ hóa một phần nhỏ trong phương thức, thay vì toàn bộ phương thức. Bạn có thể đồng bộ hóa một **khối mã** cụ thể trong phương thức. Điều này giúp cải thiện hiệu suất vì chỉ có phần mã quan trọng được đồng bộ hóa.

Trong ví dụ trên, chỉ khối mã bên trong synchronized (this) mới được đồng bộ hóa. Điều này giúp các phần mã khác trong phương thức vẫn có thể được thực thi đồng thời bởi các thread khác.

* Bạn có thể đồng bộ hóa trên **bất kỳ đối tượng nào**, không nhất thiết phải là this. Ví dụ: synchronized (someObject), khi đó thread phải giành được khóa của đối tượng someObject.

### **3. synchronized với các phương thức tĩnh**

Phương thức tĩnh (static method) đồng bộ hóa sử dụng khóa của **class object**, không phải của một đối tượng cụ thể. Điều này đảm bảo rằng chỉ một thread có thể truy cập phương thức tĩnh này trên bất kỳ đối tượng nào của lớp tại một thời điểm.

### **4. Tại sao cần synchronized?**

Trong các ứng dụng đa luồng, khi nhiều thread cùng truy cập và thay đổi dữ liệu dùng chung, sẽ có thể xảy ra các vấn đề liên quan đến **điều kiện đua (race condition)**. synchronized giúp đảm bảo rằng các thao tác trên dữ liệu dùng chung được thực thi một cách **nguyên tử** (atomic) và **có thứ tự**.

### **5. Ưu và nhược điểm của từ khóa synchronized**

#### **Ưu điểm:**

* **Đảm bảo an toàn dữ liệu**: Giúp tránh các lỗi cạnh tranh tài nguyên (race condition) bằng cách đảm bảo rằng chỉ có một thread truy cập vào tài nguyên dùng chung tại một thời điểm.
* **Dễ triển khai**: synchronized là một cách đơn giản để đảm bảo đồng bộ hóa mà không cần phải quản lý thủ công việc cấp phát khóa.

#### **Nhược điểm:**

* **Giảm hiệu suất**: Đồng bộ hóa làm chậm hệ thống vì các thread phải chờ đợi cho đến khi khóa được giải phóng, dẫn đến **trạng thái chờ** (waiting/blocking) và có thể gây ra hiện tượng **tắc nghẽn** (bottleneck).
* **Deadlock (Bế tắc)**: Nếu đồng bộ hóa không được thiết kế đúng cách, có thể xảy ra deadlock, khi mà hai hoặc nhiều thread chờ nhau giải phóng khóa mà không thread nào có thể tiếp tục thực thi.
* **Khó kiểm soát khi phức tạp**: Khi có quá nhiều điểm đồng bộ trong ứng dụng lớn, việc kiểm soát các vấn đề về hiệu suất và deadlock trở nên khó khăn.

### **6. Một số chú ý khi sử dụng synchronized**

* **Đồng bộ hóa cục bộ** (block synchronization) nên được ưu tiên hơn phương thức đồng bộ hóa để giảm phạm vi khóa và cải thiện hiệu suất.
* Tránh việc đồng bộ hóa trên các đối tượng dễ bị chia sẻ ngoài ý muốn (như String hoặc Integer), vì chúng có thể được chia sẻ giữa nhiều thread khác nhau trong JVM.
* Xem xét việc sử dụng các **API đồng bộ nâng cao** trong thư viện java.util.concurrent như ReentrantLock, AtomicInteger, hoặc ConcurrentHashMap để có thêm sự linh hoạt và kiểm soát khi đồng bộ hóa.

### **7. Các lựa chọn thay thế cho synchronized**

* **ReentrantLock**: Cung cấp các tính năng đồng bộ hóa nâng cao như khóa thời gian chờ (timed lock) hoặc khóa không bị chặn (non-blocking).