# Thread

1. Một khi thread đã stop rồi thì không thể chạy lại được(Cần phải khởi tạo lại)
2. Synchronized

* Cho phép chỉ một thread được **thực thi đoạn mã đó tại một thời điểm** trên cùng một đối tượng khóa. Để tránh race condition
  + Synchronized Instance Method (Không ảnh hưởng đến các đối tượng khác của cùng class.)
  + Synchronized Static Method (ảnh hưởng từ tất cả đối tượng cùng class)
* Synchronized block
  + synchronized(this) (Không ảnh hưởng đến các đối tượng khác của cùng class.)
  + synchronized(DemoSync.class) Đồng bộ toàn cục trên class, dù gọi từ đối tượng khác nhau.

1. Deadlock

hai hay nhiều luồng (thread) bị chặn vô thời hạn khi mỗi luồng đang giữ một tài nguyên và chờ tài nguyên mà luồng khác đang giữ, dẫn đến không có luồng nào tiếp tục thực thi được.

1. Timer, Timer Task, shedule

Timer là một lớp trong Java dùng để lập lịch thực thi một hoặc nhiều tác vụ (task) sau một khoảng thời gian nhất định hoặc lặp lại theo chu kỳ.

Timer: Thư viện trong java, tạo ra một thread riêng để chạy các task đã lập lịch.

Time task: kế thừa TimerTask và ghi đè phương thức run() để định nghĩa công việc cụ thể.

Shedule: cung cấp các phương thức schedule() để thực hiện việc này.

# MultiThread

thay vì tạo ra một luồng mới cho mỗi tác vụ và hủy bỏ luồng khi tác vụ hoàn thành. điều này có thể tốn kém tài nguyên và gây lãng phí. **Thread Pool** sẽ tạo ra một nhóm luồng (thread pool) và tái sử dụng chúng để xử lý nhiều tác vụ. Khi một tác vụ cần thực thi, một luồng có sẵn trong nhóm sẽ được lấy ra và thực thi tác vụ đó. Sau khi hoàn thành, luồng sẽ quay lại nhóm để sẵn sàng cho tác vụ tiếp theo.

**Thread Pool có chứa Blocking Queue và threads**

**Tạo thread pool**: Khi ứng dụng cần thực hiện tác vụ song song, nó sẽ tạo ra một pool chứa một số luồng ban đầu. Số lượng luồng trong pool có thể thay đổi tùy theo nhu cầu của ứng dụng.

**Gửi tác vụ vào pool**: các nhiệm vụ sẽ được chèn vào trong một **Blocking Queue**. chỉ một thread rảnh lấy ra thực thi, các thread khác chờ đến lượt.

**Tái sử dụng luồng**: Sau khi hoàn thành tác vụ, luồng không bị huỷ mà quay lại thread pool đểsẵn sàng cho tác vụ tiếp theo.

**Quản lý số lượng thread**: Thread pool có thể tự động điều chỉnh số lượng luồng dựa trên tải công việc (ví dụ, số lượng tác vụ trong queue).

|  |  |  |
| --- | --- | --- |
| **Khái niệm** | **Đồng bộ (Synchronous)** | **Bất đồng bộ (Asynchronous)** |
| Thực thi | Theo thứ tự, tuần tự | Độc lập, không chờ nhau |
| Ưu điểm | Đơn giản, dễ kiểm soát | Tăng hiệu suất, không chặn luồng |
| Nhược điểm | Có thể gây chậm do phải chờ | Phức tạp trong xử lý lỗi và luồng |

## **Bài tập cơ bản**

1. **Tạo và chạy Thread đơn giản**
   * Viết chương trình tạo một thread in ra các số từ 1 đến 10.
   * Tạo nhiều thread cùng chạy song song, mỗi thread in ra tên của nó.
2. **Sử dụng Runnable Interface**
   * Viết một class triển khai Runnable để in chuỗi ký tự. Tạo nhiều thread từ class này và chạy đồng thời.
3. **Thread Sleep và Join**
   * Viết chương trình sử dụng Thread.sleep() để tạm dừng một thread trong 2 giây.
   * Sử dụng join() để đợi thread hoàn thành trước khi main thread kết thúc.
4. **Thread Synchronization - Đồng bộ hóa đơn giản**
   * Viết chương trình có biến đếm chung. Tạo nhiều thread cùng tăng biến đếm này 1000 lần. Quan sát lỗi race condition.
   * Sử dụng từ khóa synchronized để đảm bảo thread an toàn khi cập nhật biến chung.

## **Bài tập trung cấp**

1. **Synchronized với Methods và Blocks**
   * Viết chương trình mô phỏng việc rút tiền từ tài khoản ngân hàng chung, dùng synchronized method/block để tránh lỗi cập nhật dữ liệu.
2. **Deadlock**
   * Viết ví dụ gây deadlock giữa hai thread khi mỗi thread giữ một khóa và chờ khóa của thread kia.
   * Viết lại chương trình để tránh deadlock bằng cách thay đổi thứ tự lấy khóa.
3. **Producer-Consumer với wait() và notify()**
   * Viết chương trình Producer-Consumer sử dụng wait(), notify() và synchronized để đồng bộ hóa truy cập hàng đợi.
4. **Sử dụng CountDownLatch hoặc CyclicBarrier**
   * Viết chương trình cho phép nhiều thread chờ nhau hoàn thành trước khi tiếp tục công việc.

## **Bài tập nâng cao**

1. **Thread Pool**
   * Tạo một ThreadPool đơn giản sử dụng ExecutorService để thực thi nhiều nhiệm vụ song song.
   * Mô phỏng các nhiệm vụ có thời gian xử lý khác nhau.
2. **Callable và Future**
   * Viết chương trình sử dụng Callable để trả về kết quả tính toán từ một thread.
   * Sử dụng Future để lấy kết quả và kiểm tra xem thread đã hoàn thành chưa.
3. **ReadWriteLock**
   * Viết chương trình mô phỏng đọc-ghi dữ liệu dùng ReentrantReadWriteLock để tăng hiệu suất đọc nhiều hơn ghi.
4. **Phân tích và xử lý lỗi trong đa luồng**
   * Viết chương trình mô phỏng lỗi race condition, deadlock và sử dụng công cụ debug hoặc logging để phát hiện lỗi.
5. **Đồng bộ hóa nâng cao: Phaser**
   * Viết chương trình mô phỏng nhiều thread phối hợp nhiều pha công việc bằng Phaser.
6. **Thiết kế ứng dụng đa luồng xử lý dữ liệu lớn**
   * Xây dựng chương trình đa luồng đọc file lớn, xử lý dữ liệu, và ghi kết quả với độ đồng bộ và hiệu suất tối ưu.