1. Câu 1
   1. Synchronous (Đồng bộ)

* Thực hiện tuần tự trên cùng một luồng:
  + Mỗi lệnh được thực hiện xong mới đến lệnh tiếp theo
  + Nếu một lệnh chậm(VD: đọc file, request mạng), luồng hiện tại bị chặn, toàn bộ chương trình phải chờ
* Cơ chế:
  + CPU đọc lệnh từ thread chính
  + Thực thi lệnh
  + Nếu lệnh là I/O, thread chờ I/O hoàn tất (block) trước khi chạy lệnh tiếp theo
* Hình dung:
  + Mỗi task phải xong hoàn toàn mới đến task tiếp theo
  + Ưu điểm: đơn giản, dễ debug, logic tuần tự rõ ràng
  + Nhược điểm: không hiệu quả với các tác vụ chậm hoặc I/O nhiều, CPU bị “nhà rỗi” chờ
  1. Asynchronous(bất đồng bộ)
* Thực hiện không chặn luồng chính:
  + Khi một lệnh “bất đồng bộ” được gọi (VD: đọc file, request HTTP), chương trình tiếp tục chạy lệnh tiếp theo mà không chờ kết quả
  + Kết quả sẽ được trả về sau thông qua:
    - Callback(gọi hàm khi xong)
    - Promise / Future(lấy kết quả sau)
    - Event loop
* Cơ chế trong java:
  + Main thread submit task vào thread pool
  + Task được chạy trên thread khác
  + Main thread tiếp tục thực hiện lệnh tiếp theo mà không bị block
  + Khi worker thread hoàn tất, kết quả được lưu trong Future/Callback
  + Nếu main thread cần kết quả, gọi get() hoặc callbck sẽ được trigger
  1. Tóm lại
* **Synchronous (Đồng bộ)**
  + Tác vụ **phải hoàn thành xong** rồi chương trình mới chuyển sang tác vụ tiếp theo.
  + Luồng hiện tại **bị chặn (blocked)** khi đợi kết quả.
  + Đặc điểm: tuần tự, dễ hiểu, nhưng kém hiệu quả nếu có nhiều tác vụ I/O chậm.
* **Asynchronous (Bất đồng bộ)**
  + Khi gọi một tác vụ, chương trình **không cần chờ nó xong** mà có thể tiếp tục chạy tác vụ khác.
  + Kết quả của tác vụ sẽ được trả về sau qua **callback, Future/Promise, event loop**.
  + Đặc điểm: không chặn luồng chính, hiệu quả hơn cho I/O, nhưng phức tạp hơn để quản lý luồng dữ liệu.

1. Câu 2
   1. **Synchronous (Đồng bộ)**

* **Khi nào dùng?**
  + Các tác vụ **ngắn, nhanh, chắc chắn hoàn tất ngay** (VD: tính toán trong RAM, xử lý logic đơn giản).
  + Khi cần **giữ thứ tự thực thi chặt chẽ** (VD: phải có kết quả bước 1 rồi mới làm bước 2).
  + Ứng dụng nhỏ, không có yêu cầu hiệu năng cao, không nhiều tác vụ I/O.
* **Ưu điểm**
  + Dễ viết, dễ đọc, dễ debug.
  + Luồng chạy theo thứ tự → logic rõ ràng, ít nhầm lẫn.
  + Không lo vấn đề race condition (nhiều luồng tranh chấp tài nguyên).
* **Nhược điểm**
  + **Block luồng chính** → kém hiệu quả nếu có tác vụ chậm (I/O, gọi API, đọc file lớn).
  + Gây **treo ứng dụng** (VD: UI bị đơ khi xử lý lâu).
  + Không tận dụng được sức mạnh đa lõi CPU và concurrency.
  1. **Asynchronous (Bất đồng bộ)**
* **Khi nào dùng?**
  + Các tác vụ **I/O chậm**: đọc/ghi file lớn, gọi API, truy vấn DB, network request.
  + Ứng dụng cần **đa nhiệm**: server web phải phục vụ nhiều request cùng lúc.
  + Ứng dụng **giao diện (GUI)**: tránh đơ màn hình khi chạy tác vụ nặng.
  + Khi muốn **tận dụng CPU đa lõi**, cho nhiều task chạy song song.
* **Ưu điểm**
  + Không block luồng chính → hệ thống vẫn phản hồi nhanh.
  + Xử lý nhiều tác vụ song song → hiệu năng cao hơn.
  + Tận dụng tài nguyên tốt (CPU, I/O).
* **Nhược điểm**
  + Code phức tạp hơn (callback hell, xử lý lỗi khó).
  + Debug khó hơn vì kết quả không về theo thứ tự.
  + Dễ gặp vấn đề concurrency (nhiều thread truy cập chung dữ liệu).
  1. **So sánh nhanh**

| **Tiêu chí** | **Synchronous (Đồng bộ)** | **Asynchronous (Bất đồng bộ)** |
| --- | --- | --- |
| Cách chạy | Tuần tự, chờ xong mới chạy tiếp | Không chờ, tiếp tục chạy lệnh khác |
| Hiệu năng | Thấp nếu nhiều I/O | Cao, tận dụng đa luồng, không block |
| Dễ lập trình | Dễ, trực quan | Khó hơn, cần callback/Future/Promise |
| Ứng dụng phù hợp | Tác vụ ngắn, xử lý tuần tự | I/O chậm, server nhiều request, GUI |
| Rủi ro | Treo ứng dụng khi tác vụ chậm | Race condition, khó debug |

* + - * **Tóm lại**:
* **Sync** = đơn giản, dùng cho tác vụ nhỏ, cần thứ tự chặt chẽ.
* **Async** = phức tạp hơn, nhưng hiệu quả, phù hợp cho hệ thống lớn, nhiều I/O hoặc yêu cầu responsiveness.

1. Câu 3
   1. Synchronized

* Là một từ khóa trong Java, dùng để đồng bộ hóa
* Nó đảm bảo rằng tại một thời điểm, chỉ có 1 thread được truy cập vào khối code/tài nguyên được đánh dấu synchronized
* Mục đích: tránh race condition khi nhiều thread cùng đọc/ghi vào tài nguyên chung
  1. Cách dùng
* Synchronized method
  + Khi 1 thread gọi …., các thread khác phải chờ cho đến khi thread hiện tại thoát khỏi method này
  + Lock ở đây chính là object hiện tại(this)
* Synchronized block
  + Thay vì khóa cả method, ta chỉ khóa một đoạn code cụ thể -> linh hoạt và hiệu quả hơn
  + Synchronized(this) nghĩa là dùng chính object đó làm lock
* Static synchronized method
  + Lock ở đây là class object, chứ không phải instance
  + Có nghĩa la tất cả các thread trên mọi object đều phải chờ nhau khi gọi increment()
  1. Cách hoạt đông
* Khi 1 thread đi vào vùng synchronized:
  + Nó phải giữ lock của object/class đó
  + Nếu lock đang bị thread khác giữ -> thread này phải chờ
  + Khi thread thoát khỏi synchronized block/method -> lock được giải phóng
  + Thread khác mới được vào
  1. Ưu nhược điểm
* **Ưu điểm**
  + Đơn giản, dễ dùng.
  + Giải quyết race condition hiệu quả.
* **Nhược điểm**
  + Có thể làm **giảm hiệu năng** (vì thread phải chờ lock).
  + Dễ gây **deadlock** nếu không cẩn thận khi nhiều lock lồng nhau.
* **Tóm lại:**
  + synchronized = cơ chế **lock** để tránh nhiều thread sửa chung một tài nguyên cùng lúc.
  + Dùng cho **method** hoặc **block code**.
  + Phải cẩn thận để không gây **giảm hiệu năng** hoặc **deadlock**.