**Giới thiệu**

Các ứng dụng hiện nay yêu cầu một tốc độ phản hồi cao để nâng cao trải nghiệm người dùng, giúp hệ thống mượt mà, linh hoạt, không bị đóng băng luồng. Các yêu cầu này cũng là kết quả hướng tới khi chúng ta sử dụng mô hình lập trình theo **Reactive Programming**.

Trong bài viết này, chúng ta sẽ cố gắng làm sáng tỏ mô hình lập trình này thông qua một số khái niệm Synchronous và Asynchronous , Blocking và Non-Blocking trước.

**Synchronous và Asynchronous**

Synchronous (Xử lý đồng bộ): là xử lý mà chương trình sẽ chạy theo từng bước, nghĩa là thực hiện xong đoạn code trên mới tới đoạn code kế tiếp và sẽ theo thứ tự từ trên xuống dưới, từ trái qua phải. Đây cũng là nguyên tắc cơ bản mà các bạn đã được học.

Asynchronous (Xử lý bất đồng bộ): Ngược lại với xử lý đồng bộ, nghĩa là chương trình có thể hoạt động nhảy cóc, function phía dưới có thể hoạt động mà không cần phải chờ function hay một đoạn code nào đó phía trên thực hiện xong. Dưới đây là minh họa cho việc làm việc với dữ liệu đồng bộ và bất đồng bộ :



Như ta thấy nếu các công việc không liên quan đến nhau thì bất đồng bộ giúp ta tiết kiệm thời gian xử lý hơn và mang lại cho người dùng trải nghiệm tốt hơn.

**Blocking và Non-Blocking**

Chúng ta có thể hiểu một cách đơn giản khi chúng ta muốn dấy một danh sách Student.

Lập trình theo mô hình Blocking thì phải chờ đợi chương trình thực hiện lấy tất cả Student rồi mới thực hiện các thao tác tiếp theo, hay được gọi là bị đóng băng luồng chờ quá trình đóng gói tất cả Student hoàn tất. Do đó sẽ dẫn tốn thời gian chờ đợi nếu số lượng danh sách rất lớn.

Lập trình theo mô hình Non-Blocking thì hoạt động ngược lại, không cần phải chờ đợi hoàn thiện cả danh sách Student mà với mỗi Student nào được đưa ra thì thực hiện thao tác luôn với nó. Điều này dẫn tới không bị đóng băng luồng, kể cả số lượng danh sách lớn.

**Reactive Programming**

Nói một cách ngắn gọn, **Reactive Programming** là mô hình lập trình mà ở đó dữ liệu được truyền tải dưới dạng luồng ( stream). Mô hình này dưa trên nguyên tắc Asynchronous và Non-Blocking để làm việc với dữ liệu.

Dưới đây là một số khái niệm mà bạn cần phải biết khi làm việc với mô hình này:

**Publisher:** Là nhà cung cấp dữ liệu, hoặc là nơi phát ra nguồn dữ liệu.

**Subscriber:** Lắng nghe **Publisher**, yêu cầu dữ liệu mới. Hay được gọi Là người tiêu thụ dữ liệu.

**Backpressure:** Là khả năng mà **Subscriber** cho phép **Publisher** có thể xử lý bao nhiêu yêu cầu tại thời điểm đó. Bởi vì **Subscriber** chịu trách nhiệm về luồng dữ liệu, không phải **Publisher** vì nó chỉ cung cấp dữ liệu.

**Stream:** Luồng dữ liệu bao gồm các dữ liệu trả về , các lỗi xảy ra và luồng này phải là luồng bất đồng bộ.

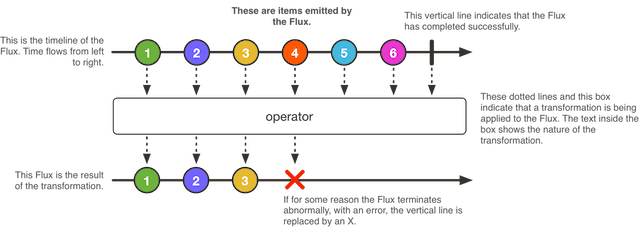
Như vậy dữ liệu của chúng ra sẽ được chuyển thành một dòng (data stream) do đó tránh được việc bị blocking và các dữ liệu phát ra thì đều được subcriber lắng nghe dẫn đến quá trình xử lý và báo lỗi diễn ra một cách đơn giản hơn.

**Reactor**

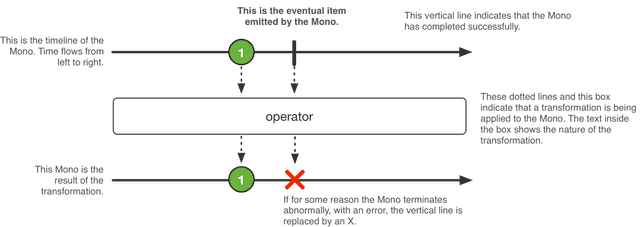
**Reactor** là một nền tảng để ta triển khai việc lập trình theo phong cách **reactive programming**. Nó được tích hợp trực tiếp với Java 8 funcion APIs như CompletableFuture, Stream, Duration.

**Reactor** cung cấp 2 loại về **Publisher** :

Flux: là một steam phát ra từ 0...n phần tử.



Mono: là một steam phát ra từ 0...1 phần tử.



Vậy là các bạn có thể hiểu được **Reactive Programming** phải không nào :D. Các bài viết tới chúng ta sẽ đi sâu hơn về các thực thi cũng như các function mà **Reactor** hỗ trợ. Hãy chú ý theo dõi và đừng quên nhận xét để chúng tôi có thể cải thiện các bài viết tốt hơn.