Backend with Java Spring

ManhDX - EngineerPro

Lecture 1 - Concurrency

ManhDX - EngineerPro

Concurrency

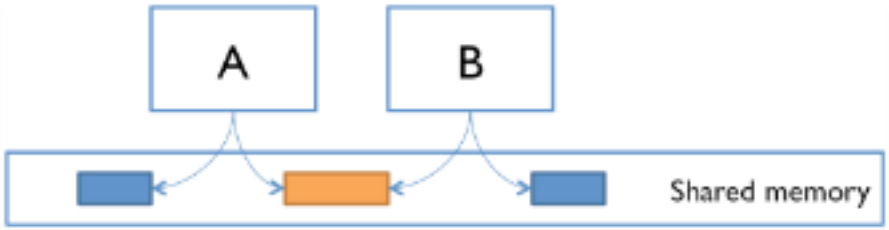
- Concurrency (Đồng thời) tức là nhiều phép tính toán cùng xảy ra tại cùng một thời điểm. Ví dụ:
  - Nhiều máy tính trong một mạng máy tính
  - Nhiều ứng dung cùng chạy trong một máy tính
  - Nhiều bộ xử lý (processor) trong một máy tính (nhiều nhân trong một chip)
  - Một website cần xử lý nhiều lượng truy cập cùng lúc

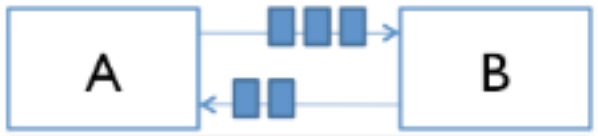
Concurrency (tt)

- Hai mô hình phổ biến của concurrent programming
  - Shared Memory: các concurrent mô-đun tương tác với nhau thông qua việc đọc và ghi các shared objects trong bộ nhớ.

    Ví dụ:
    - A và B chia sẻ chung các objects màu cam, trong khi các objects màu xanh là private.
    - A và B
  - Message Passing: các concurrent mô-đun tương tác với nhau bằng cách gửi các message tới bên kia thông qua một kênh giao tiếp.

    Ví dụ:
    A và B là 2 máy tính ở trong cùng một mạng, giao tiếp thông qua kết nối mạng

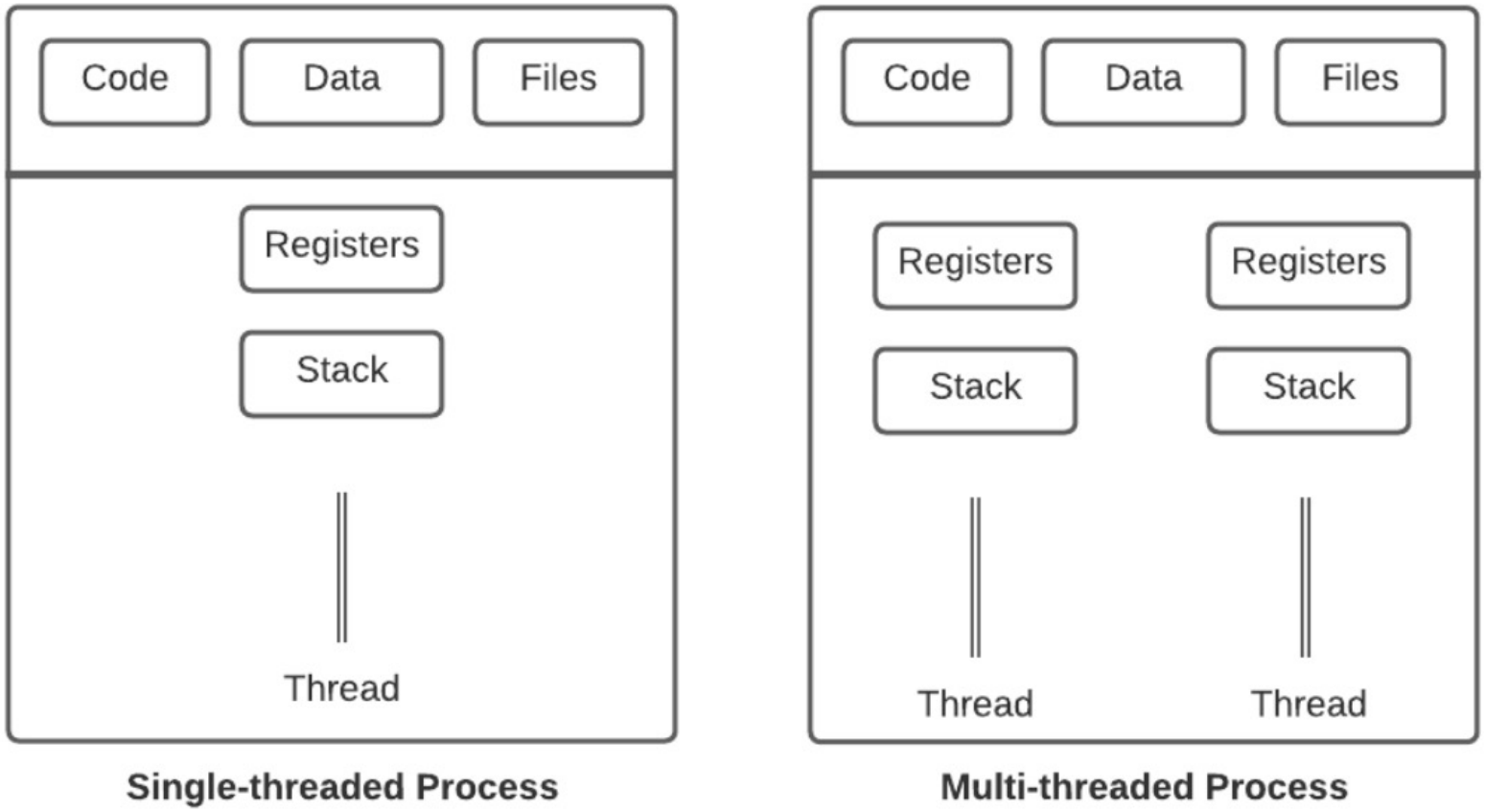




Process vs Thread

	Process	Thread
Khái niệm	Tiến trình là một chương trình đang được thực thi (đang chạy)	Luồng là một thành phần con của tiến trình. Một tiến trình có thể chạy đồng thời nhiều luồng.
Bộ nhớ	Các process là độc lập, không chia sẻ không gian nhớ với nhau	Các threads chia sẻ không gian nhớ
Khởi tạo	Cần nhiều system call để tạo nhiều process, tốn nhiều thời gian hơn	Có thể tạo nhiều threads với một system call, tốn ít thời gian hơn
Chấm dứt	Tốn thời gian hơn so với thread	Tốn ít thời gian hơn so với process
Giao tiếp	Cần sử dụng IPC, tốn kém hơn	Ít tốn kém hơn so với process
Context switching	Chậm hơn threads	Nhanh hơn nhiều so với process

Process vs Thread (tt)

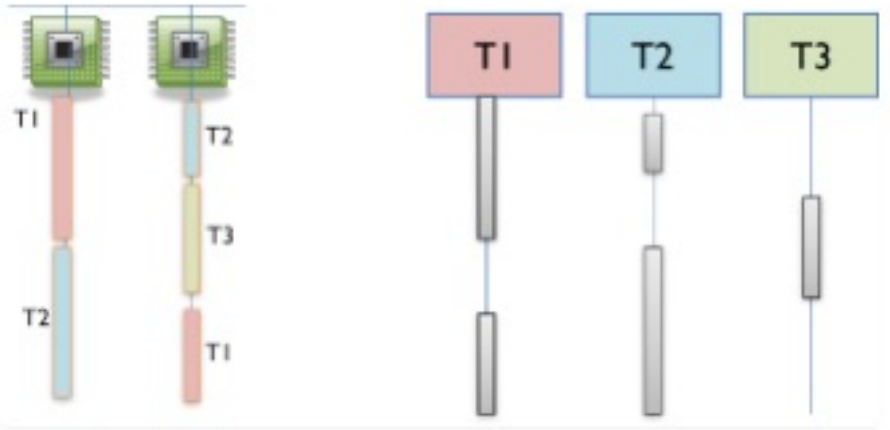


Single-threaded Process

Multi-threaded Process

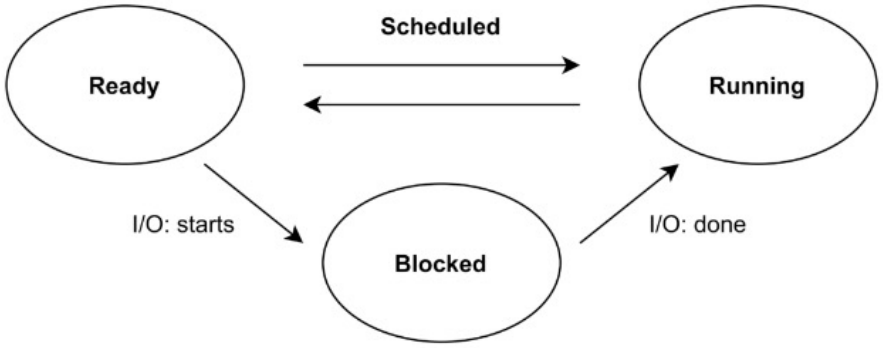
Process vs Thread (tt)

- Làm sao để OS chạy 100 tiến trình đồng thời nếu chỉ có 1 CPU?
  - OS xử lý bằng cách ảo hóa CPU. Ví dụ: Thực thi 1 process, tạm dừng nó và chạy một process khác…
  - Khái niệm này gọi là CPU time-sharing (time-slicing), giúp cho người dùng chạy nhiều process.
  - Khả năng dừng một process và chạy process khác gọi là context switching.



Process vs Thread (tt)

- Các trạng thái của process:
  - Running: process đang được chạy trên một bộ xử lý và đang thực thi các câu lệnh
  - Ready: process đã sẵn sàng để thực thi, nhưng hệ điều hành (OS) chưa quyết định chạy nó tại thời điểm này
  - Blocked: process hoàn thành một số hành động mà khiến nó không thể chạy cho đến khi một sự kiện khác xảy ra. Ví dụ: process yêu cầu I/O tới một đĩa, nó sẽ bị block. Việc này sẽ cho phép một process khác được sử dung CPU.



Process vs Thread (tt)

- Demo
  - Running Java Thread
  - Shared memory với Race Condition

Interleaving

- Hàm deposit khi chuyển xuống low-level sẽ thực thi như sau
  - Lấy giá trị hiện tại của balance (blance = 0)
  - Thêm 1
  - Ghi kết quả (balance = 1)
- Khi có 2 thread A và B chạy đồng thời, các lệnh low-level có thể xen lẫn nhau (interleave). Do đó, thứ tự thực hiện các lệnh của thread A và B là tùy tiện. Ví dụ: bên trái với kết quả đúng, bên phải thiếu 1. Đây là hiện tượng race-condition.

```
private static void deposit() {
    balance = balance + 1;
}
```

A	B
A get balance (balance=0)	
A add 1	
A write back the result (balance=1)	
	B get balance (balance=1)
	B add 1
	B write back the result (balance=2)

A	B
A get balance (balance=0)	
	B get balance (balance=0)
A add 1	
	B add 1
A write back the result (balance=1)	
	B write back the result (balance=1)

Thread Safety

- Một kiểu dữ liệu hay một hàm là thread-safe nếu như nó luôn thực hiện đúng khi được sử dụng từ nhiều threads, bất kể các threads được thực thi như thế nào.
- Trong Java có các lớp hỗ trợ thread-safe bên cạnh các lớp thông thường.
  - Ví dụ: StringBuffer là thread-safe, trong khi StringBuilder không hỗ trợ thread-safe.
  - Các thread-safe collections: BlockingQueue, ConcurrentMap, CopyOnWriteArrayList
  - Các lớp thread-safe thường có performance thấp hơn các lớp unsafe thông thường.

Lecture 1 – Networking

ManhDX - EngineerPro

Client/Server Pattern

- Client/server pattern giao tiếp thông qua message passing
- Gồm 2 loại process chính: client và server.
- Client khởi tạo liên lạc và tạo kết nối đến server. Client gửi các yêu cầu (request) đến server và server sẽ gửi lại các phản hồi (response). Client đóng kết nối.
- Một server có thể xử lý kết nối đến nhiều client đồng thời, và một client cũng có thể kết nối nhiều server đồng thời
- Ví dụ: web browser
- Ở trên internet, tiến trình của client và server chạy trên các máy khác nhau, và kết nối thông qua mạng (network)

Network concepts

- Một số khái niệm quan trọng của mạng:
  - IP Address dùng để nhân diện một network interface. IP v4 có 32 bit. Ví dụ: 1.2.3.4 hay 127.0.0.1 (localhost)
  - Hostname: là tên có thể được chuyển đổi thành địa chỉ IP. Một hostname có thể trỏ đến nhiều IP và nhiều hostname có thể trỏ đến cùng một IP. Ví dụ: google.com, localhost
  - Port: Dùng để kết nối trực tiếp đến một process đang chạy. Network interface có nhiều port, từ 1 – 65535. Một port chỉ có thể được listen bởi 1 process. Một số port phổ biến:
    - 22: SSH; 25: email server; 80: web; 443: HTTPS
  - Socket: là một điểm nối của kết nối giữa client và server
  - Protocol: là một tập các messages được trao đổi giữa 2 bên. Một số protocol phổ biến: HTTP, SMTP

Lecture 1 – Git

ManhDX - EngineerPro

Git

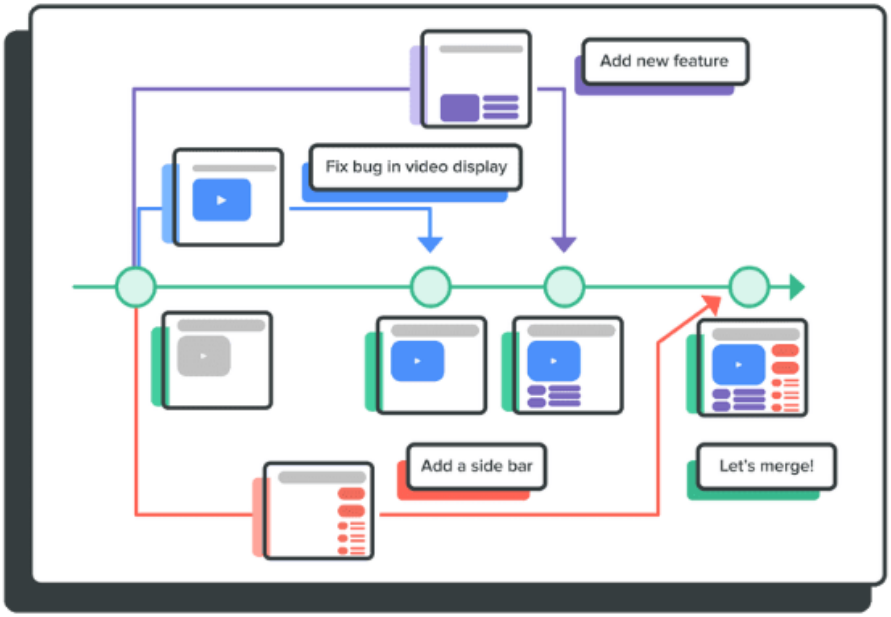
- Git là một hệ thống quản lý phiên bản (version control system).
- Hệ thống quản lý phiên bản giúp ghi lại những thay đổi ở các files theo thời gian. Nó có thể:
  - Đảo ngược thay đổi của các file về quá khứ
  - So sánh thay đổi giữa các phiên bản
  - Xem ai tạo ra những thay đổi …
- Một số khái niệm quan trọng:
  - Repository: một thư mục chứa tất cả files, và có 2 loại: local và remote repository
  - Commit (hoặc revision): một snapshot của các files tại một thời điểm
  - Add (hoặc stage): trạng thái của các files cần trước khi có thể commit.
  - Clone: tải repository về máy để tạo local repository
  - Push: Đẩy một local commit lên remote repository.
  - Pull: tải về những commit mới nhất từ remote repository

Git (tt)

- Khi push, commit có thể bị rejected do local repository của bạn không có những thay đổi mới nhất của remote repository. Khi đó, bạn sẽ cần phải pull từ remote, sau đó mới push ngược lại.
- Nếu không may mắn, những thay đổi code của bạn có thể bị conflict với những thay đổi code mới nhất trên remote. Ví dụ: bạn và đồng nghiệp sửa cùng một dòng code hoặc đoạn code. Khi đó, sẽ cần phải resolve conflict.
- Tips:
  - Để tránh conflict, hãy pull trước khi tạo ra những thay đổi

Git (tt)

- Git Branch (Nhánh Git) là một dòng phát triển độc lập, có thể được tạo ra từ một nhánh khác, có thể sử dụng khi phát triển tính năng mới (hoặc sửa lỗi) để tách biệt với nhánh chính.



Git (tt)

- Demo git clone, add, commit, status, push, pull, branch với Gitlab

Lecture 1 – JUnit

ManhDX - EngineerPro

Test

- Test:
  - Mục đích là để xác định các vấn đề hay lỗi có thể xảy ra
  - Giúp cho phát hiện các vấn đề trước khi software được triển khai (release), khiến cho việc sửa lỗi dễ dàng hơn và ít tốn chi phí hơn.
  - Nâng cao chất lượng sản phẩm bằng cách kiểm tra yêu cầu của các chức năng đã được thỏa mãn.
  - Nâng cao trải nghiệm người dùng: ví dụ như UI/UX test
- Unit test:
  - dùng để test các mô-đun/component độc lập
  - thường được viết bởi developer
  - thời gian thực thi ngắn, dễ dàng xử lý kết quả
  - dễ dàng maintain và update khi cần thiết

JUnit

- JUnit là một framework unit testing của Java
- Junit hỗ trợ các annotations:
  - @Test, @BeforeEach @AfterEach
  - @BeforeAll @AfterAll
  - @Ignore ...
- ***Best practices***: Cách đặt tên test nên gồm 3 phần:
  - tên của hàm cần test
  - kịch bản cần test
  - kết quả kì vọng khi chạy kịch bản
- ***Best practices***: cài đặt test nên gồm 3 phần:
  - Arrange (Given): khởi tạo các objects
  - Act (When): các hành động (phương thức) với objects
  - Assert (Then): kiểm tra các điều kiện cần được thỏa mãn

JUnit

- Demo JUnit

Lecture 1 – Code Review

ManhDX - EngineerPro

Code Review

- Code Review được thực hiện khi có một thay đổi bất kì trên source code.
- Ví dụ: bạn làm một task A, để hoàn thành task A bạn sẽ thực hiện thay đổi (thêm, sửa, xóa code) trên máy bạn. Sau đó những thay đổi này sẽ được đẩy đưa lên hệ thống review (ví dụ như github, gitlab) để những reviewer có thể đọc, đưa ra ý kiến và xác nhận nó.
- Reviewer có thể là người làm cùng trong team bạn.
- Sau khi được xác nhận bởi reviewer, thì code mới được merge vào nhánh chính và đưa lên production.

Code Review (tt)

- Những vấn đề có thể xem xét khi review code:
  - Bugs hoặc có thể bug:
    - Lặp code (DRY – Don’t repeat yourself)
    - Không nhất quán giữa code và yêu cầu
    - Off-by-one error
    - Scope của biến quá lớn
    - Magic numbers...
  - Code không minh bạch
    - Tên biến, tên hàm đặt không tốt
    - Thụt đầu dòng không thống nhất
    - Biến/hàm được sử dụng cho nhiều mục đích.
    - …
  - …

Code Review (tt)

- Demo Code Review với Gitlab

Lecture 1 – Dependency Injection

ManhDX - EngineerPro

Dependency

```
// Without DI
public class UserService {
    private DatabaseConnection dbConnection = new DatabaseConnection();

    public User getUser(int userId) {
        return dbConnection.fetchUser(userId);
    }
}
```

- Vấn đề:
  - Mỗi instance của UserService lại khởi tạo mới một DatabaseConnection (expensive object – tốn time & tài nguyên)
  - UserService đã bị ràng buộc chặt (tight-coupling) với DatabaseConnection, nếu như thay đổi phương thức khởi tạo new() của DatabaseConnection thì UserService cũng phải thay đổi theo
  - Sẽ khó để test hàm getUser vì bị phụ thuộc vào dbConnection, tức là phụ thuộc vào môi trường test với kết nối đến DB

Dependency Injection (DI)

```java
// With DI
public class UserService {
    private DatabaseConnection dbConnection;

    // Constructor injection
    public UserService(DatabaseConnection dbConnection) {
        this.dbConnection = dbConnection;
    }

    public User getUser(int userId) {
        return dbConnection.fetchUser(userId);
    }
}
```

- Cải tiến với Dependency Injection :
  - Đưa dbConnection vào tham số của hàm khởi tạo của UserService
  - UserService không trực tiếp tạo ra dbConnection nữa và có thể sử dụng bất cứ implementation nào của DatabaseConnection
  - Có thể test bằng cách mock DatabaseConnection với mock object và cho kết quả tùy biến
- Công dụng DI:
  - Bỏ ràng buộc chặt: thông qua việc sử dụng dependency với setter hoặc constructor
  - Cải thiện khả năng test: bằng cách thay thế object phụ thuộc với mock object (ví dụ sử dụng framework Mockito)
  - Tăng tính linh hoạt: với khả năng thay thế bằng các implementation khác nhau

Lecture 1 – Spring Framework

ManhDX - EngineerPro

Spring Framework

- Giới thiệu:
  - Spring giúp cho việc lập trình Java nhanh hơn, dễ dàng hơn và an toàn hơn.
  - Spring tập trung vào tốc độ, sự đơn giản và sự hiệu quả.
  - Spring là framework Java phổ biến nhất
  - Spring có contributors từ các big-names trong lĩnh vực công nghệ, bao gồm: Amazon, Google, Microsoft, Alibaba…

Dependency Injection

- Một ứng dụng gồm nhiều component (thành phần) khác nhau, mỗi phần có nhiệm vụ khác nhau và phối hợp với các thành phần khác để hoàn thành công việc. Khi ứng dụng chạy, mỗi thành phần cần được khởi tạo và truyền vào lẫn nhau.
- Spring hỗ trợ một container (thường gọi là Spring application context) để khởi tạo và quản lý các thành phần của ứng dụng. Những thành phần này (hoặc beans) được kết nối lẫn nhau để tạo thành ứng dụng. Quá trình này gọi là dependency injection.
- Spring hỗ trợ auto config bằng cách:
  - autowiring: tự động truyền (inject) các beans mà components cần sử dụng
  - component scanning: tự động tìm các component và tạo ra các beans tương ứng trong Spring application context

First Spring Boot Application

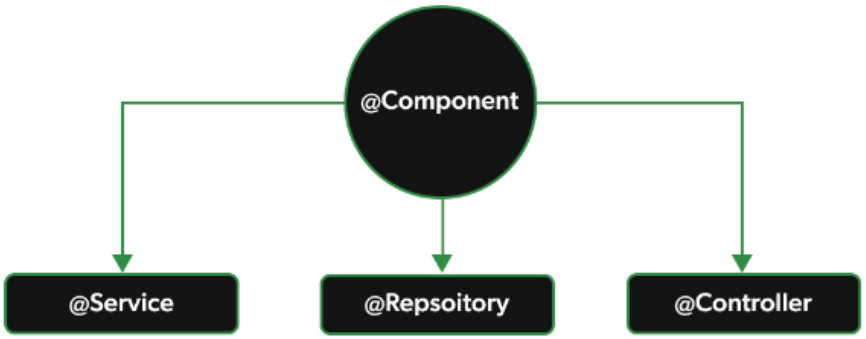
- Khởi tạo Spring Boot:
  - https://start.spring.io/
  - Sử dụng plugin trong VS Code
    - Spring Boot Extension Pack https://marketplace.visualstudio.com/items?itemName=vmware.vscode-boot-dev-pack
  - Một số plugin hữu ích khác:
    - Extension Pack for Java https://marketplace.visualstudio.com/items?itemName=vscjava.vscode-java-pack
    - Git Graph https://marketplace.visualstudio.com/items?itemName=mhutchie.git-graph
    - Gradle for Java: https://marketplace.visualstudio.com/items?itemName=vscjava.vscode-gradle

@SpringBootApplication

- @SpringBootApplication gồm 3 tính năng:
  - @EnableAutoConfiguration: sử dụng tính năng auto-config của Spring Boot, cho phép tự động đoán và cấu hình các bean dựa theo các dependency
  - @ComponentScan: giúp cho @Component scan ở package của application
  - @Configuration: giúp khởi tạo thêm các bean khác trong context hoặc thêm vào các lớp cấu tùy chỉnh thêm.

@Component và @Autowired

- @Component: là annotation giúp cho Spring nhận diện bean tự động. Tức là Spring sẽ:
  - Scan toàn bộ app để tìm các classs được đánh dấu bằng @Component
  - Khởi tạo chúng và truyền các phụ thuộc nếu có
- @Autowired: dùng để inject dependecy tự động
- Để làm rõ hơn một component thuộc tầng nào, Spring cung cấp thêm các annotation sau:
  - @Service
  - @Repository
  - @Controller



Bài tập

- Khởi tạo project với Spring Boot sử dụng build tool với Gradle
- Sử dụng Spring boot với Java (concurrency) collections, implement các chức năng sau cho người dùng:
  - Thích 1 bộ phim bất kì
  - Liệt kê danh sách các bộ phim ưu thích
  - Xóa bỏ 1 bộ phim khỏi danh sách yêu thích
  - ** Viết Unit Tests **
  - Có thể sử dụng Lombok để generate code cho các models.
- Nâng cao hơn:
  - Phim có thể là phim lẻ hoặc series
  - Thêm chức năng đánh giá số sao (star rating) cho người dùng