**Java Interview Questions**

**I. Java Core:**

**1.1. Các kiểu dữ liệu(nguyên thuỷ và không nguyên thuỷ(Object))**

* Kiểu DL nguyên thuỷ: là kiểu dữ liệu được lưu trực tiếp trong bộ nhớ   
  với kích thước và giá trị cụ thể (int, float, double, boolean, long, byte... )
* Kiểu DL k nguyên thuỷ: là kiểu dữ liệu Object được lưu trong bộ nhớ Heap (String, Integer, Boolean, Float ....)

**1.2. String**

1. **Định nghĩa**

* Trong java, string(chuỗi) là một đối tượng biểu diễn một chuỗi các giá trị char.
* Lớp String trong java cung cấp rất nhiều các phương thức để thực hiện các thao tác với chuỗi như: compare(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring(), v.v
* Lớp java.lang.String được implements từ các interface **Serializable, Comparable and CharSequence.**
* CharSequence Interface được sử dụng để biểu diễn chuỗi các ký tự. Nó được cài đặt bởi các lớp String, **StringBuffer và StringBuilder**. Điều này có nghĩa là chúng ta có thể tạo chuỗi ký tự trong java bằng cách sử dụng 3 lớp này.
* **String là bất biến (immutable)** tức là không thể thay đổi. Có nghĩa là khi nào bạn thay đổi giá trị của bất kỳ chuỗi nào thì một instance mới được tạo ra. Đối với chuỗi có thể thay đổi, bạn có thể sử dụng các lớp **StringBuffer và StringBuilder.**

1. **So sánh String (== và equals())**

* **==** là so sánh địa chỉ  
  **equals()** so sánh value

1. **Cách khởi tạo**

* Có 2 cách để khai báo String là sử dụng phép gán (=) or new String()  
  **+ (=):** sẽ tìm kiếm trong String pool xem đã tồn tại giá trị cần khởi tạo chưa, nếu chưa có sẽ tạo mới đối tượng và tham chiếu đến biến. Nếu tồn tại rồi thì sẽ trả về tham chiếu rồi gán cho biến.  
  **+ (new String()):** sẽ tạo mới 1 đối tượng String được lưu trong bộ nhớ Heap mà không liên quan gì đến String pool.

1. **Sử dụng StringBuilder**

* Phương thức phổ biến: **append("string"), insert(index, "string"), delete(index from, index to), reverse(), toString()**
* Khi sử dụng nó sẽ thay đổi trực tiếp đối tượng String và sẽ **không tạo ra đối tượng trung gian -> tăng hiệu suốt khi làm việc**, đặc biệt khi sử dụng trong loop.

**II. Spring boot:**

**2.1. SpringBootApplication annotation**

- @SpringBootApplication đóng gói các chú thích @Configuration, @EnableAutoConfiguration, @ComponentScan với các thuộc tính mặc định của của chúng.

+ @Configuration: cho phép đăng ký các bean bổ sung trong ngữ cảnh hoặc nhập các lớp cấu hình bổ sung

+ @ComponentScan: cho phép @Component quét trên tất cả các @bean được khởi tạo trong phạm vi project

+ @EnableAutoConfiguration: kích hoạt cơ chế tự động cấu hình của Spring Boot

**2.2. ComponentScan**

**-** Được gắn trong annotation @SpringBootApplication.

- Trong các ứng dụng Spring Boot, chúng ta thường định nghĩa các bean bằng cách sử dụng các Annotation như @Component, @Service, @Repository, hoặc @Controller, ... Các bean này cần được quản lý và tạo ra bởi Spring Boot để chúng ta có thể sử dụng chúng trong ứng dụng.

- Component Scan cho phép Spring Boot tự động tìm kiếm và quản lý các bean trong ứng dụng của bạn. Mặc định, Spring Boot sẽ quét toàn bộ các package và các package con của package main.

**2.3. Lifecycle Bean**

1. Khi IoC Container (ApplicationContext) tìm thấy một Bean cần quản lý, nó sẽ khởi tạo bằng Constructor
2. inject dependencies vào Bean bằng Setter, và thực hiện các quá trình cài đặt khác vào Bean như setBeanName, setBeanClassLoader, v.v..
3. Hàm đánh dấu @PostConstruct được gọi
4. Tiền xử lý sau khi @PostConstruct được gọi.
5. Bean sẵn sàng để hoạt động
6. Nếu IoC Container không quản lý bean nữa hoặc bị shutdown nó sẽ gọi hàm @PreDestroy trong Bean
7. Xóa Bean.

**2.4. Dependency injection và IoC (Đọc thêm** [*link*](https://quochung.cyou/spring-ioc-dependency-injection-component-va-bean/)**)**

1. **IoC(Inversion of Control - đảo ngược sự điều khiển)**

* IoC nhằm mục đích đơn giản hóa quá trình tạo đối tượng và liên kết giữa chúng, bằng cách tuân theo nguyên tắc: Không tạo đối tượng, chỉ mô tả cách chúng sẽ được tạo ra.
* Inversion of control có nhiều kiểu để thực hiện, như dùng ServiceLocator, Delegate,… nhưng phổ biến nhất là Dependency injection

1. **Dependency injection**

* DI là một dạng thực hiện của IoC, bằng cách tiêm (inject) module vào một module khác cần nó.
* 2 loại injection chính:

**+ Constructor-based injection:** Dùng inject các module bắt buộc. Các module được inject nằm trong constructor, và được gán lần lượt vào các field.

**+ Setter-based injection:** Dùng inject các module tùy chọn. Mỗi module sẽ được inject thông qua setter, nằm ở tham số và cũng gán cho field nào đó.

**2.5. Spring container (IoC Container)**

* IoC Container trong Spring chính là lõi của Spring Framework. IoC Container sẽ tạo ra các đối tượng, nối chúng lại với nhau, cấu hình chúng, và quản lý vòng đời của chúng từ khi tạo ra đến khi bị hủy. IoC Container sử dụng DI để quản lý các thành phần tạo nên một ứng dụng. Những đối tượng này được gọi là Spring Bean. IoC Container được cung cấp thông tin từ các tập tin XML.
* Có 2 loại IoC Container, đó là:

**+ BeanFactory**

**+ ApplicationContext**

**2.6. Ba cách thực hiện Dependency Injection trong Spring Framework**

xem ví dụ tại đây [*link*](https://quochung.cyou/3-cach-thuc-hien-dependency-injection-di-va-van-de-voi-autowired-trong-spring/)

1. **Field Injection**

* Field Injection chính là tên gọi của cách inject dependency khi ta khai báo bằng @Autowired
* Về cơ bản khi khai báo annotation này trên 1 field, method, constructor. Spring sẽ tự tìm dependency phù hợp và kết nối chúng với nhau.
* Nhược điểm: do nó tự động -> thiếu chủ động trong quá trình xử dụng

1. **Setter Injection**

* Cách làm trên như tên của nó, ta sẽ set một dependency của class lớn hơn ở thời điểm runtime
* Cách làm này cho phép ta dễ dàng thay đổi dependency của một class trong runtime, ví dụ như đổi 1 repo khác 1 service nào đó, …
* Tuy nhiên, ngoài ưu điểm là sự flexible khi set ở runtime, thì nó cũng đi cùng vài nhược điểm khác
* Do được set trong runtime, ta đôi khi có thể gặp phải NullPointer khi dependency chưa được khởi tạo, là null, … (vì lúc này chúng là optional, không bắt buộc với class nữa)
* Do nó lỏng lẻo hơn nên lúc này với các method trong class gọi đến dependency, ta không được đảm bảo là dependency đã được set, hoặc ta lại phải thêm code để kiểm tra là có dependency chưa, 1 là thêm code, 2 là dẫn tới nullpointer
* Với cách làm như thế này, ta có thể không cần Spring quản lý các bean mà có thể test chay hơn, có nhiều quyền kiểm soát hơn vào việc quản lý các dependency

1. **Constructor Injection**

* Như tên của thủ pháp, ta sẽ thêm dependency cho 1 class ngay tại thời điểm khởi tạo class cha
* Việc này đảm bảo 1 class luôn có đủ dependency để khởi tạo (tuỳ theo cách custom constructor)
* (Việc sử dụng mock cho phép nhiều khả năng khác trong test hơn, ví dụ mình có thể kiểm tra xem một hàm trong class đó được gọi bao nhiêu lần, … , chúng được quản lý theo dạng Proxy Design Pattern, mình sẽ nói ở bài viết khác)

**2.7. Khác biệt nổi bật giữa các phiên bản Java JDK (**[*link*](https://viblo.asia/p/tat-tan-tat-tu-jdk-8-den-jdk-21-MG24B63B4z3)**)**

**1. Java 8:**

1. **Lambda Expressions (Biểu thức lambda):** Cho phép lập trình hàm (functional programming) bằng cách cho phép sử dụng các hàm vô danh (anonymous functions).

* Cung cấp cú pháp ngắn gọn để tạo các đoạn mã xử lý sự kiện hoặc triển khai functional interface chỉ chứa một phương thức duy nhất
* **Ưu điểm:**

+ Cung cấp implement cho Functional Interface

+ Viết ít code hơn

+ Hiệu quả hơn nhờ việc hỗ trợ tuần tự (sequential) và song song (parallel) thông qua stream API

1. **Functional Interfaces:** Là interface chỉ chứa một phương thức trừu tượng duy nhất được gọi là "abstract method".

* Tính năng chính của Functional Interface là khả năng sử dụng biểu thức lambda để triển khai phương thức của nó.
* Chú thích @FunctionalInterface để đánh dấu những giao diện như vậy.

1. **Stream Api**

* Giới thiệu một trừu tượng mới gọi là Stream để xử lý chuỗi các phần tử.
* Hỗ trợ các thao tác hàm trên streams như filter, map, reduce, v.v.

1. **Default Methods (Phương Thức Mặc Định):**

* Default Methods được giới thiệu để thêm tính năng mở rộng cho các interface mà không làm phá vỡ các lớp implement từ interface này.
* Một interface có thể chứa phương thức mặc định, là phương thức có thể được triển khai bởi các lớp triển khai của interface đó. **Các lớp triển khai có thể sử dụng phương thức mặc định này mà không cần phải triển khai lại nó.**

1. **New Date and Time Api (Api ngày và giờ mới)**

* Gói java.time giới thiệu một API linh hoạt hơn để xử lý ngày và giờ.
* Giải quyết các vấn đề với các lớp java.util.Date và java.util.Calendar cũ.

**2. Java 17**

**III. OOP**