**Các câu hỏi phỏng vấn**

1. **Spring là gì? Các thành phần của Spring?**

* Spring là một framework phát triển ứng dụng phổ biến trong ngôn ngữ Java. Nó cung cấp một cách tiếp cận linh hoạt và mạnh mẽ để xây dựng các ứng dụng Java, giúp giảm độ phức tạp và tăng tính khả năng mở rộng.
* Các thành phần của Spring:
* **Spring Core**: Cung cấp Dependency Injection (DI) và IoC (Inversion of Control) để quản lý các thành phần và sự phụ thuộc trong ứng dụng
* Giảm sự ràng buộc giữa các thành phần và tạo ra 1 cách linh hoạt để quản lý các bean trong ứng dụng
* **Spring MVC**: Giúp phát triển các ứng dụng web dựa theo mô hình MVC (Model – View – Controller)
* **Spring Boot**: Tạo ra các ứng dụng độc lập mà không cần cấu hình phức tạp.
* **Spring Security**: Cung cấp các tính năng liên quan đến xác thực và ủy quyền trong các ứng dụng. Cho phép kiểm soát quyền truy cập vào các tài nguyên và bảo mật người dùng bằng cách xác thực người dùng và thực hiện các phương thức bảo mật như xác thực dựa trên vai trò (Role-based authentication) và xác thực dựa trên mã JWT (JSON Web Tokens)
* **Spring Data**: Cung cấp tiện ích cho phép làm việc với CSDL. Spring Data cung cấp những tính năng như tạo truy vấn linh hoạt, map các đối tượng Java vào CSDL, và xử lý các thao tác CRUD 1 cách dễ dàng

1. **DI là gì? IoC là gì? Cơ chế load bean của IoC Container?**

* DI (Dependency Injection) là 1 kỹ thuật làm giảm sự phụ thuộc giữa các đối tượng với nhau, giúp cho code trở nên dễ dàng hơn trong việc bảo trì và nâng cấp
* Có 3 cách thực hiện Dependency Injection:
* Thông qua constructor
* Thông qua phương thức setter
* Thông qua interface injection
* IoC (Inversion of Control) là 1 nguyên tắc thiết kế đảo ngược quyền điều khiển đối tượng so với lập trình hướng thủ tục truyền thống. IoC giúp giảm sự ràng buộc giữa các thành phần trong ứng dụng và tạo ra 1 cách linh hoạt để quản lý các đối tượng
* Cơ chế load bean của IoC Container được thực hiện bằng cách sử dụng metadata (XML, Java Annotation hoặc Java Configuration) để định nghĩa và cấu hình các bean trong ứng dụng
* **Quét và đọc metadata**: IoC đọc các tệp metadata để biết được các đối tượng bean nào cần được tạo và quản lý
* **Tạo và cấu hình bean**: Dựa vào metadata, IoC Container tạo các đối tượng bean cung cấp các giá trị cần thiết để cấu hình (DI) các thành phần phụ thuộc vào bean đó
* **Quản lý vòng đời (lifecycle management):** IoC Container quản lý vòng đời của các bean , gồm việc khởi tạo , cấu hình và giải phóng tài nguyên khi không cần thiết
* **Đưa bean vào sử dụng**: Các bean được tạo và cấu hình sẽ được đưa vào sử dụng trong ứng dụng, và các thành phần khác có thể truy cập và sử dụng các bean thông qua DI

1. **Có bao nhiêu cách inject bean?**

Có 3 cách phổ biến để inject bean theo mô hình IoC:

* **Constructor injection**: Các dependency được truyền thông qua tham số của constructor
* **Setter injection**: Các dependency được thiết lập thông qua các phương thức setter trên bean
* **Filed injection**: Các dependency được chú thích trực tiếp trên field của bean. IoC Container sẽ khởi tạo bean bằng constructor mặc định và sau đó, sử dụng reflection để gán giá trị của dependency trực tiếp vào trường đó

1. **Bean scope là gì? Có bao nhiêu loại bean scope?**

* Bean scope là 1 thuộc tính của Spring Framework, xác định phạm vi tồn tại và sử dụng của bean đó.
* Có 5 loại bean scope :
* **Singleton**: Đây là scope mặc định của 1 bean trong Spring. Khi bean có scope là singleton, chỉ có 1 instance duy nhất của bean được tạo và sử dụng trong toàn bộ ứng dụng
* **Prototype**: Mỗi lần yêu cầu bean, Spring sẽ tạo ra 1 instance mới của bean đó. Điều này đảm bảo rằng mỗi yêu cầu bean sẽ nhận được 1 instance độc lập
* **Request**: Scope request chỉ có hiệu lực trong 1 request HTTP. Mỗi request sẽ có 1 instance riêng của bean
* **Session**: Scope session chỉ có hiệu lực trong một phiên làm việc (session) của người dùng. Mỗi session sẽ có 1 instance riêng của bean
* **Global session**: Tương tự như session , nhưng áp dụng cho các ứng dụng web sử dụng cơ chế global session
* Singleton thường là lựa chọn phổ biến vì hiệu năng và khả năng tái sử dụng, trong khi prototype hữu ích khi bạn cần một instance mới mỗi lần yêu cầu bean. Scope request và session thường được sử dụng trong ứng dụng web để quản lý dữ liệu của từng request hoặc session riêng biệt.

1. **Làm thế nào để ưu tiên 1 bean khi Spring tìm thấy quá nhiều bean phù hợp?**

* Giải pháp là sử dụng các annotation sau:
* **@Primary annotation**: Khi Spring Boot gặp nhiều bean cùng loại, nó sẽ ưu tiên các bean được đánh dấu là @Primary
* **@Qualifier annotation**: Sử dụng @Qualifier để chỉ định tên hoặc giá trị đặc biệt để xác định bean mà bạn muốn ưu tiên
* **@Order annotation**: Sử dụng @Order để xác định thứ tự ưu tiên của các bean. Bean có giá trị @Order nhỏ hơn sẽ được ưu tiên hơn

1. **Các loại Fetch Type trong Hibernate?**

* Có 2 loại FetchType chính:
* **FetchType.EAGER**: Hibernate sẽ tự động tải tất cả dữ liệu liên quan ngay lập tức khi truy vấn đối tượng gốc
* FetchType.EAGER thường dùng khi bạn muốn truy xuất lập tức các đối tượng liên quan và không muốn gặp vấn đề về lazy loading
* **FetchType.LAZY**: Hibernate sẽ không tự động tải dữ liệu liên quan ngay lập tức. Thay vào đó, dữ liệu sẽ được tải chỉ khi nào bạn truy cập trực tiếp vào các thuộc tính liên quan
* FetchType.LAZY thường được sử dụng khi bạn muốn trì hoãn việc tải dữ liệu không cần thiết và tối ưu hóa hiệu năng
* @ManyToOne và @OneToOne có Fetch Type mặc định là FetchType.EAGER
* @ManyToMany và @OneToMany có Fetch Type mặc định là FetchType.LAZY

1. **Spring AOP là gì?**

* Spring AOP (Aspect-Oriented Programming) là 1 phần quan trọng của Spring Framework cho phép chúng ta tách rời logic phần mềm chính (core logic) và các khía cạnh (aspects) khau nhau như ghi log, xử lý ngoại lệ, …
* Trong Spring AOP có 4 loại advice hỗ trợ:
* **Before advice**: Chạy trước khi method được thực thi
* **After returning advice**: Chạy sau khi method trả về 1 kết quả
* **After throwing advice**: Chạy khi method ném ra 1 exception
* **Around advice**: Chạy khi method được thực thi (Bao gồm cả 3 loại device trên

1. **So sánh Spring MVC và Spring Boot**

|  |  |
| --- | --- |
| **Spring MVC** | **Spring Boot** |
| Là 1 phần của Spring Framework, cho phép chúng ta xây dựng ứng dụng web dựa theo mô hình MVC | Là 1 framework của Spring, Spring Boot cung cấp những tính năng mạnh mẽ, triển khai ứng dụng web dễ dàng hơn và ít cần cấu hình hơn |
| Spring MVC yêu cầu cấu hình phức tạp để khởi tạo và cấu hình các thành phần của ứng dụng (các file XML,…) | Spring Boot tự động cấu hình và quản lý các thành phần trên ứng dụng , giảm thiểu sự phức tạp |
| Phải quản lý các dependencies 1 cách chi tiết và cụ thể | Tích hợp sẵn công nghệ và các framework phổ biến như Tomcat, Hibernate, Thymeleaf,… |

1. **So sánh các cache level của JPA Hibernate**

Trong Hibernate có 3 mức độ cache:

* **Cấp độ Cache Thực thể (Entity-level Cache):** Cache này lưu trữ các thực thể được truy vấn từ CSDL, giúp giảm số lần truy cập truy cập CSDL khi truy vấn lại cùng 1 thực thể
* Nên sử dụng cache này với các thực thể thường xuyên được cập nhật
* **Cấp độ Cache Mối quan hệ (Relationship-level Cache):** Cache này lưu trữ thông tin về các mối quan hệ giữa các thực thể, giúp giảm số lượng câu truy vấn liên quan đến mối quan hệ
* Nên sử dụng cache này cho các mối quan hệ mà có ít thay đổi
* **Cấp độ Cache Truy Vấn (Query-level Cache):** Cache này lưu trữ kết quả của các câu truy vấn, giúp giảm số lượng đến CSDL khi cùng 1 câu truy vấn được thực hiện
* Nên sử dụng cache này cho các câu truy vấn mà kết quả ít thay đổi

1. **Sự khác biệt giữa TRUNCATE, DELETE và DROP?**

* **TRUNCATE**: Câu lệnh TRUNCATE dùng để xóa toàn bộ dữ liệu trong 1 bảng cụ thể. Dữ liệu trong bảng sẽ bị xóa hoàn toàn và không thể khôi phục được. Câu lệnh TRUNCATE cũng reset lại các giá trị tự tăng (auto-increment) trong bảng
* **DELETE**: Câu lệnh DELETE dùng để xóa các bản ghi cụ thể từ 1 bảng. Dữ liệu bị xóa có thể được khôi phục lại bằng cách sử dụng câu lệnh ROLLBACK hoặc từ bản sao dự phòng (backup)
* **DROP**: Câu lệnh DROP dùng để xóa 1 đối tượng (table, view, index, procedure, …) hoàn toàn khỏi cơ sở dữ liệu. Đối tượng sẽ bị xóa vĩnh viễn và không thể khôi phục

1. **UNIQUE và PRIMARY KEY khác nhau như thế nào?**

|  |  |
| --- | --- |
| **PRIMARY KEY** | **UNIQUE** |
| Trong 1 bảng chỉ có duy nhất 1 PRIMARY KEY, không trùng lặp trên 1 cột hoặc 1 nhóm cột | Trong 1 bảng có thể có nhiều ràng buộc UNIQUE, không trùng lặp trên 1 cột hoặc 1 nhóm cột |
| PRIMARY KEY không cho phép giá trị NULL | Ràng buộc unique cho phép giá trị NULL |

1. **Các cách để tối ưu hóa truy vấn trong cơ sở dữ liệu**

* **Chỉ chọn các cột cần thiết**: Chỉ chọn các cột cần thiết thay vì lấy toàn bộ các cột trong bảng
* **Tạo chỉ mục (indexing):** Đánh chỉ mục thường xuyên trong câu truy vấn để tăng tốc độ tìm kiếm và sắp xếp dữ liệu
* **Sử dụng JOINs hiệu quả**: Sử dụng JOINs để kết hợp dữ liệu từ nhiều bảng thay vì sử dụng nhiều câu truy vấn riêng lẻ
* **Sử dụng câu lệnh WHERE chính xác**: Sử dụng câu lệnh WHERE để lọc dữ liệu ngay từ câu truy vấn, tránh việc phải truy vấn toàn bộ dữ liệu và sau đó lọc ở ứng dụng
* **Sử dụng GROUP BY và HAVING**: Sử dụng GROUP BY để nhóm dữ liệu, HAVING để áp dụng điều kiện cho câu truy vấn
* **Tối ưu hóa cấu trúc câu truy vấn**: Sử dụng các cấu trúc UNION, UNION ALL, EXISTS, IN, NOT IN để thực hiện truy vấn 1 cách hiệu quả
* **Sử dụng cơ chế caching**: Để lưu trữ kết quả truy vấn đã được thực hiện trước đó, giúp giảm thiểu thời gian truy cập CSDL
* **Đánh giá và tối ưu cấu trúc dữ liệu**: Xem xét lại cấu trúc dữ liệu của bảng, kiểm tra việc sử dụng các kiểu dữ liệu phù hợp và tối ưu, cân nhắc việc tạo các bảng phụ, phân chia dữ liệu để giảm thiểu thời gian truy vấn.
* **Sử dụng kỹ thuật nhân bản dữ liệu**: Đôi khi nhân bản dữ liệu giúp tăng tốc độ truy vấn, tuy nhiên cũng làm tăng kích thước dữ liệu và đồng bộ hóa dữ liệu
* **Sử dụng truy vấn parametrized**: Sử dụng truy vấn parametrized (truy vấn có tham số) để tránh các lỗ hổng bảo mật như SQL injection và tăng tốc độ truy vấn

1. **Cấu trúc UNION, UNION ALL, EXISTS, IN, NOT IN là gì?**

* UNION và UNION ALL là phép toán kết hợp hai hay nhiều tập hợp dữ liệu có cấu trúc tương tự thành 1 tập hợp duy nhất. UNION là 1 tập hợp chứa tất cả các bản ghi duy nhất từ các tập hợp con. UNION ALL thì chưa tất cả các bản ghi từ các tập hợp con, bao gồm cả bản ghi trùng lặp
* EXISTS là 1 phép toán kiểm tra sự tồn tại của các bản ghi trong 1 hoặc nhiều bảng dựa trên 1 điều kiện. Sẽ trả về TRUE hoặc FALSE
* IN là phép toán so sánh 1 giá trị với 1 tập hợp có giá trị được chỉ định. NOT IN ngược lại với IN

1. **SQL Injection là gì? Làm thế nào để tránh được SQL Injection?**

* SQL Injection là 1 kỹ thuật tấn công trong lập trình web, kẻ tấn công sẽ chèn các đoạn mã SQL nguy hiểm vào câu truy vấn SQL được thực thi trên hệ thống, từ đó khai thác dữ liệu từ database
* Để ngăn chặn SQL Injection cần thực hiện các biện pháp như sử dụng câu truy vấn tham số hóa, câu truy vấn động cẩn thận , kiểm tra và xác thực dữ liệu đầu vào, hạn chế quyền truy cập đến CSDL và sử dụng các công cụ bảo mật phù hợp

1. **Deadlock là gì? Cách xử lý deadlock?**

* Deadlock là tình trạng mà 2 hoặc nhiều luồng chờ đợi tài nguyên mà lẫn nhau đang nắm giữ, dẫn đến tình trạng không luồng nào được thực thi và ở trạng thái chờ mãi mãi
* Cách xử lý deadlock:
* **Tránh việc các khóa lồng nhau**: Khi 1 tài nguyên đã được giao cho 1 luồng thì đừng cố giao tài nguyên đó cho 1 luồng khác
* **Tránh khóa các tài nguyên không cần thiết để hạn chế deadlock**, phải xem xét kỹ xem tài nguyên đó có được sử dụng cho nhiều luồng hoặc giá trị của nó có ảnh hưởng gì đến kết quả tính toán của các luồng khác hay không
* **Sử dụng join() method** để các thread khác chờ cho đến khi 1 thread được hoàn thành rồi mới bắt đầu khởi chạy

1. **Race condition là gì? Data race là gì? Cách xử lý race condition và data race?**

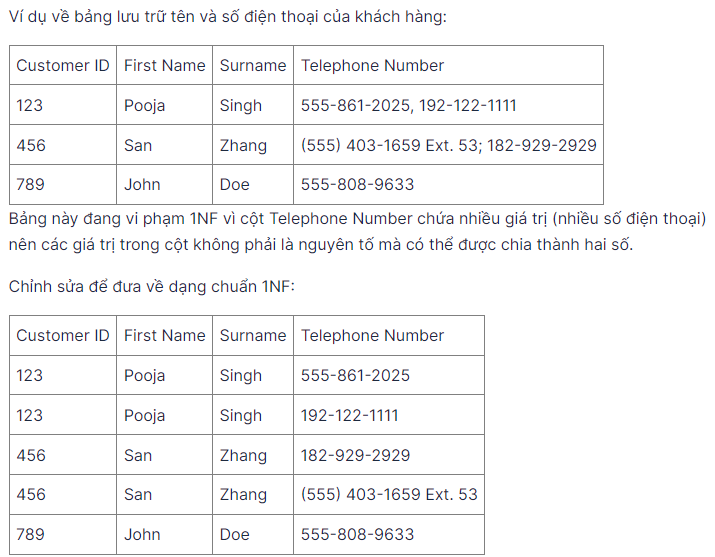
* Race condition là tình trạng xảy ra khi 2 hoặc nhiều luồng cùng truy cập vào 1 tài nguyên chung 1 cách không đồng bộ, đẫn dến kết quả bị sai
* Data race là tình trạng xảy ra khi 2 hoặc nhiều luồng cùng truy cập vào cùng 1 biến hoặc tài nguyên chia sẻ mà ít nhất 1 luồng thực hiện hoạt động ghi (write) lên biến đó
* Race condition là tình trạng xảy ra khi có sự cạnh tranh không đồng bộ giữa các hoạt động của các luồng trên dữ liệu chung, data race là 1 trường hợp cụ thể của race condition khi có sự cạnh tranh giữa các hoạt động ghi trên biến chia sẻ

1. **Bất đồng bộ là gì? Cách xử lý bất đồng bộ trong Java**

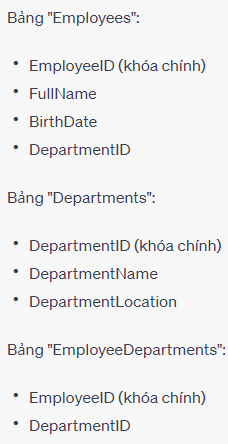
* Bất đồng bộ trong Java là tình trạng hai hay nhiều luồng cùng truy cập vào và thay đổi dữ liệu chia sẻ mà không được đồng bộ hóa đúng cách
* Gây ra race condition, data inconsitancy, deadlock
* Cách xử lý bất đồng bộ trong Java:
* **Sử dụng đồng bộ hóa (synchronized):** Sử dụng từ khóa synchronized hoặc lock để đảm bảo 1 luồng được thực thi vào 1 khối mã cùng 1 thời điểm
* **Sử dụng các lớp đồng bộ hóa:** ConcurrentHashMap hoặc Collections.synchronizedList là các lớp đồng bộ hóa
* **Sử dụng cấu trúc dữ liệu thread-safe:** ConcurrentHashMap, CopyOnWriteArrayList, AtomicInteger, AtomicLong là các cấu trúc dữ liệu có sẵn trong Java
* **Sử dụng các phương pháp không đồng bộ:** Sử dụng các phương pháp không đồng bộ như CompletableFuture, ExecutorService hoặc các công nghệ Reactive như Reactor hoặc RxJava
* Đảm bảo tính nhất quán ,an toàn truy cập và thay đổi dữ liệu chia sẻ

1. **Chuẩn 1NF, 2NF, 3NF, BCNF là gì?**

* **Chuẩn 1NF** : Mỗi ô dữ liệu trong bảng phải chứa giá trị duy nhất, mỗi cột chỉ chứa 1 giá trị từ miền



* **Chuẩn 2NF** : Bao gồm chuẩn 1NF và yêu cầu mỗi thuộc tính trong bảng phải phụ thuộc hoàn toàn vào khóa chính và không phụ thuộc vào bất kỳ thuộc tính (cột) nào khác trong bảng
* **Chuẩn 3NF** : Bao gồm chuẩn 2NF , yêu cầu mỗi thuộc tính (cột) trong bảng phải phụ thuộc hoàn toàn vào khóa chính và không phụ thuộc vào bất kỳ thuộc tính (cột) phi khóa nào trong bảng

-> 

* **Chuẩn BCNF** : Bao gồm chuẩn 3NF và mọi phụ thuộc phi chính phải phụ thuộc vào khóa chính

1. **Mô tả các thuật toán sắp xếp?**

* **Bubble Sort**: Là thuật toán hoán đổi lần lượt các phần tử cạnh nhau đến khi các phần tử được sắp xếp theo thứ tự tăng dần hoặc giảm dần
* **Insertion Sort**: Chia danh sách thành 2 phần đã sắp xếp và chưa sắp xếp, duyệt các phần tử của phần chưa sắp xếp và chèn vào danh sách đã sắp xếp theo đúng thứ tự tăng dần hoặc giảm dần
* **Selection Sort**: Tìm phần tử nhỏ nhất trong mảng sau đó hoán đổi với phần tử đầu tiên trong mảng, lặp lại quá trình này để có được danh sách sắp xếp mong muốn
* **Merge Sort**: Là thuật toán chia danh sách thành 2 nửa, sắp xếp từng nửa độc lập rồi trộn 2 nửa này lại với nhau theo thứ tự
* **Quick Sort**: Chọn 1 phần tử làm pivot và chia danh sách thành 2 phần, 1 phần có giá trị nhỏ hơn pivot, 1 phần có giá trị lớn hơn pivot. Đệ quy sắp xếp 2 phần này

1. **Mô hình 3 layer?**

* **Lớp Presentation (Presentation Layer)** : Lớp này xử lý giao diện người dùng và tương tác với người dùng. Nó hiển thị thông tin và thu thập dữ liệu từ người dùng, sau đó gửi yêu cầu tới lớp Logic
* **Lớp Logic (Business Layer)** : Lớp này chứa logic nghiệp vụ của ứng dụng. Nó xử lý yêu cầu từ lớp Presentation, thực hiện các phép tính, kiểm tra và xử lý dữ liệu theo quy tắc kinh doanh
* **Lớp Data Access (Data Access Layer)** : Lớp này tương tác với cơ sở dữ liệu. Nó cung cấp các phương thức để truy xuất, sửa đổi và xóa dữ liệu từ cơ sở dữ liệu. Lớp Logic sử dụng lớp này để lấy và lưu trữ dữ liệu

1. **Ưu điểm của Spring MVC so với JSP Servlet?**

* **Modularity**: Spring MVC sử dụng kiến trúc loosely coupled, cho phép tách thành các tầng xử lý logic nghiệp vụ (controllers), xử lý logic giao diện (views) và xử lý logic truy cập dữ liệu (models)
* Ứng dụng dễ dàng sử dụng, bảo trì và testing
* **Dependency Injection**: Spring MVC cung cấp cơ chế Dependency Injection (DI) cho phép đưa các dependencies vào controllers, views và các thành phần khác
* Thúc đẩy tái sử dụng code, khả năng testing, dễ dàng quản lý dependencies
* **Testability**: Spring MVC hỗ trợ Unit test và Integration test, với DI thì có thể dễ dàng mock các dependencies và viết test case cụ thể
* **Flexibility :** Spring MVC được hỗ trợ nhiều công nghệ khác nhau như JSP, Thymeleaf, Velocity, FreeMarker.
* **Request Mapping:** Spring MVC cung cấp cơ chế request mapping linh hoạt dựa trên các tiêu chí như URL patterns, HTTP methods, request parameters, headers and more
* **Interceptors:** Spring MVC cung cấp các bộ chặn, cho phép chặn các yêu cầu và thực hiện logic xử lý trước và sau. Ví dụ như authentication, logging, caching, …
* **Tích hợp được với các Spring features khác:** Spring MVC có thể tích hợp liền mạch với các Spring features phổ biến như Spring Security, Spring Data, Spring AOP, …

1. **Thread safety là gì? Cách để đạt được thread safety?**

* Thread safety là 1 chương trình, class hay method có thể hoạt động chính xác và đáng tin cậy trong môi trường đa luồng , mà không gây ra các lỗi race condition, data corruption, deadlock hoặc sai kết quả
* Cách để đạt được thread safety có thể là sử dụng khóa (lock), các khối hay phương thức synchronized, atomic operations và cấu trúc dữ liệu đa luồng như ConcurrentHashMap

1. **Garbage Collector trong Java là gì?**

* Garbage Collector trong Java là 1 thành phần của máy ảo JVM có trách nhiệm thu dọn bộ nhớ không sử dụng (garbage) để giải phóng tài nguyên và tái sử dụng bộ nhớ
* Garbage Collector thực hiện giải phóng bộ nhớ theo các bước sau đây:

Xác định đối tượng không còn tham chiếu -> Xóa các đối tượng không còn được tham chiếu -> Cấp phát bộ nhớ mới

1. **Thuật toán đệ quy được lưu trong bộ nhớ như thế nào?**

* Thuật toán đệ quy trong Java được lưu trữ trong bộ nhớ thông qua việc tạo ra các stack frame hoặc activation record
* Stack frame chứa các địa chỉ trở về (return address), giá trị các biến local của hàm, giá trị các tham số được truyền vào hàm và các giá trị tạm thời
* Mỗi lời gọi đệ quy tạo ra 1 stack frame mới , sau đó các frame này được xếp chồng lên Execution Stack và sẽ bị gỡ bở khi đệ quy hoàn thành

1. **Phân biệt Singleton và Prototype**

* **Singleton**:
* Singleton là mẫu thiết kế nhằm đảm bảo 1 lớp chỉ có 1 đối tượng duy nhất trong suốt quá trình chạy của ứng dụng
* Đối tượng Singleton được tạo ra 1 lần và sau đó được sử dụng lại ở các lần gọi sau
* Truy cập đến đối tượng Singleton thông qua 1 phương thức tĩnh (static) để đảm bảo rằng chỉ có 1 thể hiện duy nhất được trả về
* **Prototye**:
* Prototype là 1 mẫu thiết kế nhằm tạo ra các đối tượng mới từ 1 đối tượng mẫu đã tồn tại
* Mỗi lần cần tạo đối tượng mới, 1 bản sau của đối tượng mẫu được tạo ra
* Các đối tượng mới không liên quan hay phụ thuộc vào nhau

1. **Phân biệt Stack và Heap**

* **Heap**
* Heap là bộ nhớ được sử dụng để lưu trữ các biến instance, các đối tượng
* Các biến instance và đối tượng trong Heap được quản lý bởi Garbage Collector của JVM
* Các đối tượng trong Heap được truy cập thông qua con trỏ từ biến tham chiếu
* **Stack**
* Stack là bộ nhớ được sử dụng để lưu trữ các biến local, các biến tham số và các biến tham chiếu trong phương thức
* Khi phương thức được gọi, 1 frame mới được tạo trên stack để lưu trữ các biến local của phương thức đó, khung sẽ được loại bỏ khi phương thức kết thúc
* Các biến trong Stack được truy cập trực tiếp thông qua tên biến

1. **Mô hình MVC là gì?**

Mô hình MVC là mô hình được xây dựng theo kiến trúc Model, View, Controller

* **Model**: Là nơi lưu trữ và logic xử lý dữ liệu trong ứng dụng. Model không phụ thuộc vào View hoặc Controller
* **View**: Đại diện cho giao diện người dùng trong ứng dụng. View hiển thị dữ liệu từ Model và nhận tương tác người dùng
* **Controller**: Điều phối tương tác View và Controller. Controller chịu trách nhiệm xử lý logic của ứng dụng và đảm bảo sự tương tác giữa Model và View

1. **So sánh String và StringBuffer (hoặc StringBuilder)**

* **String**
* String là 1 lớp bất biến, khi chuỗi đã được khởi tạo thì không thể thay đổi giá trị của nó
* Khi thực hiện thao tác thay đổi chuỗi, thì Java tạo 1 đối tượng chuỗi mới thay vì đối tượng chuỗi ban đầu
* Có thể gây lãng phí tài nguyên và giảm hiệu suất
* **StringBuffer (hoặc StringBuilder)**
* StringBuffer (hoặc StringBuilder) là 1 lớp có thể thay đổi, cho phép thay đổi nội dung của chuỗi
* Khi thực hiện thao tác thay đổi chuỗi, thì StringBuffer (hoặc StringBuilder) sẽ thay đổi trực tiếp trên đối tượng chuỗi hiện có mà k tạo bản sao mới
* Tiết kiệm bộ nhớ, tài nguyên và tăng hiệu suất

1. **So sánh Session và Cookie**

**Session**

* Session không được lưu trữ ở trình duyệt của người dùng
* Dữ liệu session được lưu trữ ở phía server
* Dữ liệu session không dễ dàng sửa được vì nó được lưu ở server
* Sau khi đóng trình duyệt sẽ hết phiên làm việc (session)

**Cookie**

* Cookie được lưu trữ trên trình duyệt của người dùng
* Dữ liệu cookie được lưu trữ ở phía client
* Dữ liệu cookie dễ dàng sửa đổi khi chúng được lưu trữ trên client
* Dữ liệu cookie có sẵn trong trình duyệt đến khi hết hạn
* Session được lưu trữ trên máy chủ và sử dụng ID phiên để xác định người dùng, Cookie được lưu trữ trên máy tính của người dùng và được gửi đến máy chủ trong các yêu cầu trang web

1. **Từ khóa final, static và this trong lập trình hướng đối tượng**

**Từ khóa final:**

* Biến được khai báo là final, nó phải được khởi tạo giá trị và không thể thay đổi
* Khi 1 phương thức được khai báo là final, nó không thể được ghi đè bởi lớp con
* Khi 1 lớp được khai báo là final, nó không thể được kế thừa bởi lớp con

**Từ khóa static:**

* Từ khóa static được sử dụng để tạo ra 1 thành viên của lớp mà không cần tạo ra 1 phiên bản đối tượng
* Biến static là biến chung cho tất cả các đối tượng của lớp và được lưu trữ trong bộ nhớ static
* Phương thức static có thể được gọi trực tiếp thông qua tên lớp mà không cần khởi tạo đối tượng
* Không thể truy cập các thành viên non-static từ 1 phương thức static

**Từ khóa this:**

* Từ khóa this được sử dụng để đề cập đến đối tượng hiện tại của lớp
* Từ khóa this được sử dụng để truy cập các biến và phương thức của đối tượng hiện tại
* Từ khóa this cũng được sử dụng để khởi tạo 1 đối tượng hoặc gọi 1 constructor từ constructor khác trong cùng 1 lớp
* Nếu 1 biến local cùng tên với biến instance, từ khóa this được sử dụng để xác định rõ đâu là biến instance, đâu là biến local

1. **So sánh HTTP và HTTPS**

* **Bảo mật**: HTTP không mã hóa dữ liệu truyền tải, HTTPS sử dụng SSL/TLS để mã hóa dữ liệu
* HTTPS có tính bảo mật cao hơn HTTP, HTTPS giúp bảo vệ dữ liệu khỏi việc bị đánh cắp hoặc thay đổi trong quá trình truyền tải
* **Port sử dụng:** HTTP sử dụng port 80, HTTPS sử dụng port 443
* **Tốc độ truyền tải:** HTTP sẽ nhanh hơn HTTPS 1 chút vì không mã hóa và giải mã dữ liệu truyền tải

1. **So sánh List và ArrayList**

**List**

* List là 1 interface trong Java
* List không có giới hạn về kích thước
* List không định nghĩa cách triển khai cụ thể
* List bao gồm nhiều lớp triển khai khác nhau như ArrayList, LinkedList và Vector

**ArrayList**

* ArrayList là 1 lớp cụ thể triển khai interface List
* ArrayList là 1 mảng động, có thể thay đổi kích thước
* ArrayList được triển khai bằng cách sử dụng mảng
* ArrayList cho phép chứa các phần tử trùng lặp và phần tử null

1. **So sánh ArrayList và LinkedList**

**ArrayList**

* ArrayList là 1 mảng động , cho phép lưu trữ các phần tử theo chỉ mục index
* ArrayList sẽ tốt hơn LinkedList ở việc tìm kiếm và truy xuất dữ liệu vì chỉ cần truy cập vào index của phần tử trong mảng
* ArrayList sẽ sử dụng ít bộ nhớ hơn so với LinkedList

**LinkedList**

* LinkedList lưu trữ các phần tử theo danh sách liên kết doubly
* LinkedList sẽ tốt hơn ArrayList ở việc thao tác với dữ liệu như thêm, sửa, xóa vì chỉ cần thay đổi liên kết giữa các node
* LinkedList sẽ sử dụng bộ nhớ nhiều hơn so với ArrayList

1. **So sánh ArrayList và Vector**

**ArrayList**

* ArrayList không được đồng bộ hóa nên ArrayList sẽ không đảm bảo tính nhất quán và an toàn trong môi trường đa luồng
* ArrayList sẽ có hiệu suất nhanh hơn Vector trong môi trường đơn luồng
* Không thể chủ động thay đổi kích thước của ArrayList

**Vector**

* Vector được đồng bộ hóa nên Vector sẽ đảm bảo được tính nhất quán và an toàn trong môi trường đa luồng
* Vector sẽ có hiệu suất chậm hơn ArrayList trong môi trường đơn luồng
* Có thể chủ động thay đổi kích thước của ArrayList

1. **So sánh Array và ArrayList**

**Array**

* Array là 1 mảng có kích thước cố định
* Array có thể lưu trữ được kiểu dữ liệu nguyên thủy và đối tượng
* Array không được cung cấp những phương thức thao tác với dữ liệu như là ArrayList
* Array sẽ sử dụng ít bộ nhớ hơn so với ArrayList

**ArrayList**

* ArrayList là 1 mảng động, có khả năng thay đổi kích thước
* ArrayList không thể lưu trữ được kiểu dữ liệu nguyên thủy
* ArrayList có khả năng sử dụng những phương thức để thao tác với dữ liệu
* ArrayList sẽ sử dụng nhiều bộ nhớ hơn so với Array

1. **So sánh HashSet và TreeSet**

HashSet và TreeSet đều cho phép lưu trữ các phần tử null

**HashSet**

* HashSet sử dụng cơ chế mã băm hash để lưu trữ các phần tử
* HashSet không đảm bảo thứ tự của các phần tử, có thời gian truy cập nhanh với các thao tác dữ liệu phần tử

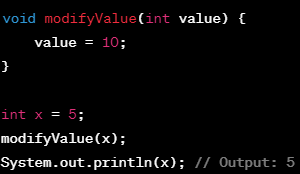
**TreeSet**

* TreeSet sử dụng cây nhị phân để lưu trữ các phần tử
* TreeSet sắp xếp các phần tử theo thứ tự tăng dần hoặc theo tiêu chí sắp xếp chỉ định, có thời gian truy cập lâu với các thao tác dữ liệu phần tử

1. **So sánh biến tham chiếu và biến tham trị**

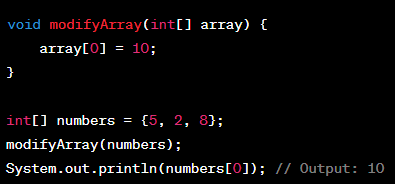
**Biến tham trị:**

* Khi truyền biến tham trị vào 1 hàm hay phương thức, 1 bản sao giá trị của biến được tạo ra và được sử dụng trong phạm vi của hàm hoặc phương thức đó
* Thay đổi giá trị của biến trong hàm hay phương thức -> không ảnh hưởng đến giá trị của biến ban đầu bên ngoài phạm vi đó
* Biến tham trị được sử dụng cho kiểu dữ liệu nguyên thủy



**Biến tham chiếu:**

* Khi truyển biến tham chiếu vào 1 hàm hay phương thức, tham chiếu đến đối tượng được truyền vào được tạo ra, cả biến gốc và tham chiếu đều trỏ đến cùng 1 đối tượng trong bộ nhớ
* Thay đổi đối tượng thông qua tham chiếu -> ảnh hưởng đến đối tượng ban đầu bên ngoài phạm vi đó
* Biến tham chiếu được sử dụng cho các kiểu dữ liệu đối tượng như các lớp và các mảng



1. **Connection Pool là gì?**

* Connection Pool là cơ chế quản lý và tái sử dụng các kết nối đến cơ sở dữ liệu trong ứng dụng web
* Việc sử dụng Connection Pool giúp giảm thiểu thời gian tạo và đóng kết nối, giảm tải cho cơ sở dữ liệu và tăng hiệu suất cho ứng dụng
* Quản lý tài nguyên tốt hơn và đảm bảo tính nhất quán và an toàn cho các thao tác đến cơ sở dữ liệu
* Các thư viện và framework phổ biến trong Java như là Apache Tomcat, Spring Framework, Hibernate đều cung cấp các cơ chế Connection Pool để hỗ trợ trong quản lý kết nối đến cơ sở dữ liệu

1. **Cách chạy ứng dụng Spring Boot**

Chuẩn bị môi trường (JDK) -> Tạo ứng dụng Spring Boot (Spring Initializr hoặc Spring Boot CLI) -> Import dự án vào môi trường phát triển (IntelliJ IDEA, Eclipse,…) -> Cấu hình ứng dụng (applications.properties hoặc applications.yml) -> Chạy ứng dụng

1. **Phân biệt Hibernate và MyBatis**

**Hibernate**

* Hibernate là 1 framework ORM toàn diện, cung cấp các tính năng mạnh mẽ để ánh xạ đối tượng trong ứng dụng Java với cấu trúc CSDL quan hệ
* Hibernate sử dụng các annotation hoặc file cấu hình XML để ánh xạ các đối tượng Java với các bảng trong cơ sở dữ liệu
* Hibernate tự động quản lý sự tương tác với cơ sở dữ liệu gồm CRUD dữ liệu

**MyBatis**

* MyBatis là 1 framework ORM , cho phép truy vấn SQL trực tiếp để tương tác với CSDL
* MyBatis sử dụng file cấu hình XML để ánh xạ các câu truy vấn SQL với các phương thức trong mã Java
* MyBatis không cung cấp tính năng tự động như Hibernate, phải viết câu lệnh CRUD dữ liệu và quản lý transaction

1. **Biến môi trường là gì? Nó có tác dụng gì?**

* Biến môi trường là 1 giá trị được đặt trong hệ thống môi trường của 1 hệ điều hành. Nó cung cấp thông tin và cấu hình cho các ứng dụng và quá trình hoạt động trên hệ thống
* Tác dụng của biến môi trường là:
* Cấu hình ứng dụng
* Quản lý session
* Điều chỉnh hành vi hệ thống
* Tích hợp và triển khai ứng dụng

1. **API là gì?**

API (Application Programming Interface) là cơ chế cho phép 2 thành phần phần mềm giao tiếp với nhau bằng 1 tập hợp các định nghĩa và giao thức

1. **Enum là gì?**

Enum là 1 kiểu dữ liệu đặc biệt được sử dụng để định nghĩa 1 tập hợp các hằng số có giá trị rõ ràng

1. **Websocket là gì?**

Websocket là 1 giao thức truyền thông 2 chiều để thiết lập và duy trì kết nối liên tục giữa client và server

1. **Các loại index trong csdl?**

* Primary index
* Unique index
* Clustered Index
* Non-Clustered Index
* Full-text Index
* Spatial Index

1. **SOLID Principle là gì?**

SOLID Principle là 1 nguyên tắc thiết kế phần mềm được sử dụng để xây dựng các hệ thống linh hoạt, dễ bảo trì và mở rộng

* **Single Responsibility Principle (SRP)** : 1 lớp nên chỉ có 1 trách nhiệm duy nhất
* **Open/Closed Principle (OCP):** Mã nguồn nên được mở để mở rộng nhưng đóng để sửa đổi
* **Liskov Substitution Principle (LSP):** Đối tượng của 1 lớp con phải có thể thay thế được đối tượng của lớp cha mà không làm thay đổi tính năng của chương trình
* **Interface Segregation Principle (ISP):** Khách hàng không nên phụ thuộc vào giao diện mà họ không sử dụng
* **Dependency Inversion Principle (DIP):** Các module cao cấp không nên phụ thuộc vào các module cấp thấp, cả hai nên phụ thuộc vào 1 abstraction

1. **Phân biệt GET,POST,PUT,DELETE**

GET, POST, PUT, DELETE là phương thức trong giao thức HTTP (Hypertext Transfer Protocol) được sử dụng đẻ tương tác với các nguồn tài nguyên trên web

* **Phương thức** **GET**: Phương thức GET được sử dụng để yêu cầu lấy thông tin từ 1 nguồn tài nguyên trên server. Phương thức GET không làm thay đổi trạng thái của server hoặc nguồn tài nguyên, chỉ trả về dữ liệu yêu cầu
* **Phương thức** **POST**: Phương thức POST được sử dụng để gửi dữ liệu từ client lên server để tạo mới 1 nguồn tài nguyên hoặc thực hiện 1 hành động. Phương thức POST có thể thay đổi trạng thái của server hoặc nguồn tài nguyên
* **Phương thức** **PUT**: Phương thức PUT được sử dụng để cập nhật và thay đổi 1 nguồn tài nguyên trên server
* **Phương thức** **DELETE**: Phương thức DELETE được sử dụng để xóa 1 nguồn tài nguyên trên server

1. **Immutable class là gì?**

Immutable class là 1 loại lớp trong lập trình mà các đối tượng của nó không thể thay đối sau khi khởi tạo

**Đặc điểm của immutable class:**

* Giá trị của các thuộc tính không thay đổi sau khi đối tượng được khởi tạo.
* Không cung cấp các phương thức setter để thay đổi các thuộc tính.
* Các thuộc tính được khai báo là private và chỉ được truy cập thông qua các phương thức getter.
* Không có phương thức nào thay đổi trạng thái của đối tượng.

**Lợi ích của immutable class:**

* **Đảm bảo tính nguyên vẹn dữ liệu**: Các đối tượng immutable không thể bị thay đổi trạng thái sau khi khởi tạo, điều này đảm bảo tính toàn vẹn của dữ liệu.
* **An toàn đa luồng**: Đối tượng immutable có thể được truy cập và sử dụng đồng thời từ nhiều luồng mà không gây ra xung đột dữ liệu.
* **Tối ưu hóa hiệu suất**: Vì không cần xử lý việc thay đổi trạng thái, các đối tượng immutable có thể được sử dụng lại và chia sẻ một cách an toàn.

1. **Batch insert là gì?**

Batch insert trong cơ sở dữ liệu là một quá trình chèn dữ liệu từ nhiều bản ghi vào cơ sở dữ liệu cùng một lúc thay vì chèn từng bản ghi một. Thay vì gửi các truy vấn INSERT riêng lẻ, batch insert cho phép chèn nhiều bản ghi cùng lúc, giúp cải thiện hiệu suất và tăng tốc quá trình chèn dữ liệu.

1. **So sánh MySQL và PostgreSQL**

* **Hiệu suất:** MySQL sẽ nhanh hơn PostgreSQL trong khi thực hiện những tác vụ đơn giản , còn PostgreSQL có hiệu suất tốt hơn MySQL khi xử lý các tác vụ phức tạp
* **Độ tin cậy:** PostgreSQL có khả năng xử lý lỗi tốt hơn khi so với MySQL. PostgreSQL có khả năng phục hồi dữ liệu cao hơn, đảm bảo tính toàn vẹn của cơ sở dữ liệu trong trường hợp có lỗi
* **Khả năng mở rộng:** MySQL sẽ có khả năng mở rộng dễ dàng hơn so với PostgreSQL
* **Hỗ trợ:** MySQL đang được hỗ trợ rộng rãi bởi cộng đồng và có nhiều công cụ hỗ trợ phát triển ứng dụng. PostgreSQL cũng có sự hỗ trợ đáng kể từ cộng đồng, nhưng không phổ biến như MySQL

1. **So sánh Stack và Queue**

* **Sắp xếp phần tử:** Trong Stack các phần tử mới được thêm vào và xóa ở đỉnh danh sách, Queue các phần tử được thêm vào cuối danh sách và xóa phần tử ở đầu danh sách
* **Cấu trúc:** Stack có cấu trúc là LIFO, còn Queue có cấu trúc là FIFO
* **Số lượng phần tử:** Stack và Queue không bị giới hạn về số lượng phần tử được lưu trữ

1. **MicroService là gì? Ưu và nhược điểm của MicroService**

* MicroService là kiểu kiến trúc phần mềm mà module được chia thành các service nhỏ (microservice) đảm nhiệm nghiệp vụ riêng
* **Ưu điểm của MicroService:**
* Các service được triển khai riêng biệt nên dễ dàng nâng cấp và maintain ứng dụngg
* Do tách biệt nên nếu 1 service bị lỗi, toàn bộ hệ thống vẫn hoạt động bình thường
* Các service tách biệt nên có thể tùy biến công nghệ (ngôn ngữ lập trình, database,…) riêng
* **Nhược điểm của MicroService:**
* Các module phải giao tiếp với nhau qua network (Webservices/Message Queue) nên tốc độ có thể bị ảnh hưởng
* Các module phải tự giải quyết các vấn đề về bảo mật, transaction,…
* Phức tạp trong việc đồng nhất dữ liệu (data consistency)
* Phức tạp trong việc deploy/monitoring các service

1. **Message Queue là gì? Kiến trúc gồm những gì? Message Queue có tác dụng gì?**

* Message Queue là kiến trúc để thực hiện việc giao tiếp không đồng bộ. Nó là hàng đợi chứa các message chờ để được xử lý tuần tự theo cơ chế FIFO
* **Kiến trúc của Message Queue bao gồm:**
* Message: Thông tin để gửi (Text, Binary, JSON,…)
* Producer: Tạo ra thông tin và đưa thông tin vào Message Queue
* Message Queue: Chứa các message, cho phép producer và consumer tương tác với nhau
* Consumer: Nhận message từ Message Queue và xử lý
* Message Queue cần được sử dụng khi các microservice tương tác bất đồng bộ với nhau. Các message sẽ được đưa vào Message Queue được xử lý với 1 microservice khác, và sẽ được chỉ xóa đi khi nó đã được xử lý , điều này đảm bảo tác vụ chắc chắn được thực hiện và 1 lần duy nhất

1. **Độ phức tạp của thuật toán là gì?**

Độ phức tạp của thuật toán là khái niệm trong khoa học máy tính, được sử dụng để đánh giá độ khó và thời gian thực hiện của 1 thuật toán. Nó được đo bằng số lần các thao tác cơ bản (so sánh, gán, truy cập bộ nhớ, …) được thực hiện khi thuật toán được thực hiện trên 1 tập dữ liệu đầu vào cụ thể

1. **CI/CD là gì?**

CI/CD là khái niệm liên quan đến quy trình phát triển phần mềm kéo dài và tự động hóa quá trình triển khai sản phẩm

* **CI (Continuous Integration):** Là phương pháp xây dựng và kiểm tra mã nguồn thường xuyên để phát hiện lỗi càng sớm càng tốt. CI sẽ tự động kiểm tra tính hợp lệ của mã nguồn và xây dựng các bài kiểm tra khi có sự thay đổi bất kỳ vào dự án của nhà phát triển
* **CD (Continuous Delivery/ Continuous Deployment):** Sau quá trình CI, CD (Continuous Delivery) tự động kiểm tra xây dựng và đóng gói thành 1 sản phẩm có thể triển khai. Còn CD (Continuous Deployment) tự động đưa sản phẩm mới thành công vào môi trường sản xuất
* Quá trình CD cung cấp giải pháp tự động đưa sản phẩm mới vào sản xuất nhanh chóng, mang lại lợi ích cho doanh nghiệp như ít lỗi hơn, phản hồi nhanh và khả năng đáp ứng nhanh hơn

CI/CD giúp đẩy nhanh tốc độ, tính linh hoạt của phát triển phần mềm, tăng cường chất lượng sản phẩm và đáp ứng nhanh hơn với các yêu cầu của khác hàng