Danh sách câu hỏi phỏng vấn Java (tài liệu tự soạn)

1. **Java Core:**
2. **Thế nào là lập trình hướng đối tượng?**

* Lập trình hướng đối tượng là một kỹ thuật lập trình cho phép lập trình viên tạo ra các đối tượng trong code và trừu tượng hóa các đối tượng đó.
* Đối tượng: một đối tượng bao gồm hai thông tin:
  + Thuộc tính: là những thông tin, đặc điểm của đối tượng. VD: con người có tên, tuổi, mắt, mũi, tay, chân,...
  + Hành vi (phương thức): là những thao tác, hành động mà đối tượng đó có thể thực hiện được. VD: một người có thể thực hiện hành động nói, ăn, đi lại,...
  + Đối tượng là một thể hiện của class.
* Lớp: là một template, một khuôn mẫu mô tả các đối tượng sẽ được tạo ra.

1. **Các tính chất đặc thù của lập trình hướng đối tượng?**

* **Tính kế thừa:** Khi 1 class được thừa hưởng lại những thuộc tính và phương thức của 1 class khác.
  + Tái sử dụng code
  + Dễ dàng nâng cấp và bảo trì
* **Tính bao đóng:** Là 1 kỹ thuật ẩn dấu những thông tin bên trong đối tượng.
  + VD: Các class liên quan đến nhau thì gom vào 1 package, các thuộc tính để private và các phương thức để public.
* **Tính đa hình:** Là khả năng một đối tượng có thể thực hiện một tác vụ theo nhiều cách khác nhau (hiểu đơn giản là các đối tượng, các phương thức giống nhau có thể có các hành vi khác nhau tùy vào từng tình huống khác nhau), là khả năng muôn hình vạn trạng của một đối tượng.
  + Tính đa hình được thể hiện trong code dưới 3 hình thức: Overload, Override, đa hình thông qua các đối tượng đa hình.
  + VD: Ghế: ghế sofa, ghế tựa,...
* **Tính trừu tượng:** Là tiến trình ẩn các chi tiết trình triển khai và chỉ hiện tính năng tới người dung. ***Trừu tượng*** trong thực tế còn có thể hiểu là cái gì đó không có thực. Vậy tính ***Trừu tượng*** trong OOP ý muốn nói đến một lớp nào đó mang một đặc tính trừu tượng, không có thực. Và như vậy bạn có thể hiểu là lớp Trừu tượng sẽ là lớp không tồn tại đúng không nào?

Thực ra thì lớp Trừu tượng vẫn có tồn tại, vẫn là một lớp thôi. Nhưng nó trừu tượng ở chỗ, nó ***không thể được dùng để tạo ra các đối tượng như những lớp bình thường khác***. Lớp Trừu tượng khi này chỉ là cái “xác không hồn”, hay bạn có thể hiểu ***nó chỉ là một cái sườn, để mà bạn có thể tạo ra các lớp con của nó dựa vào sự ràng buộc từ cái sườn này***. (**hay nó là tính chất chỉ nêu tên vấn đề mà không thể hiện cụ thể**, **Đó là một quá trình che giấu các hoạt động bên trong và chỉ hiển thị tính năng thiết yếu đối với người dùng mà không nói quy trình hoạt động**).

* + VD: Khi mà user gửi tin nhắn thì người đó chỉ soạn tin nhắn và ấn gửi nhưng user đó không biết các hoạt động bên trong khi quá trình tin nhắn được gửi.

1. **Sự khác nhau giữa while và doWhile?**

|  |  |
| --- | --- |
| While | doWhile |
| + Vòng lặp while kiểm tra điều kiện ở đầu vòng lặp và nếu điều kiện thỏa mãn thì khối lệnh bên trong vòng lặp được thực thi.  + Nếu điều kiện trong while là sai thì không một câu lệnh nào trong vòng lặp được thực thi. | + Vòng lặp doWhile thì điều kiện được thực thi sau khi thực hiện khối lệnh trong phần thân của vòng lặp.  + Nếu điều kiện trong doWhile là sai thì các câu lệnh trong vòng lặp được thực hiện ít nhất 1 lần. |

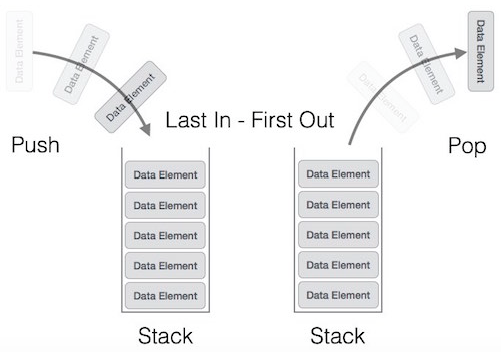
1. **Java là gì?**

* Java là một ngôn ngữ lập trình bậc cao, hướng đối tượng và bảo mật cao. Nó được tạo ra năm 1995 bởi James Gosling. Nó được sử dụng trong phát triển phần mềm, web, game hay ứng dụng trên các thiết bị di động.
* Slogan: “Write once run anywhere”.
* Các tính năng nổi bật của java:
  + Đơn giản
  + Bảo mật
  + Hướng đối tượng
  + Đa luồng
  + Thông dịch

1. **JDK? JVM? JRE?**

* JDK (Java Development Kit):
  + Dùng để phát triển ứng dụng Java.
  + Compile, run, debug ứng dụng Java.
* JVM (Java Virtual Machine):
  + Trình thông dịch và thực thi mã bytecode ra mã máy chạy.
  + File .java -> File .class (bytecode) -> Mac, windows, linux.
* JRE (Java Runtime Environment):
  + **Tạo ra môi trường** để chạy ứng dụng Java.
  + Chỉ chạy các ứng dụng Java.

1. **Cách tổ chức hoạt động của các Collection Framework như List, Map, Set, Stack, Queue?**
   1. **List:**
   * List Interface là một tập hợp theo thứ tự cho phép chúng ta thêm và xóa các phần tử như một mảng.
   1. **Map:**
   * Map Interface cho phép các phần tử được lưu trữ trong các cặp key- value.
   1. **Set:**
   * Set Interface cho phép chúng ta lưu trữ các phần tử trong các tập hợp khác nhau tương tự như tập hợp trong toán học.
   1. **Stack:**
   * Có thể hiểu nó là ngăn xếp thẳng đứng. Đặc điểm của nó là LIFO (Last In First Out).

****

* 1. **Queue:**
  + Có thể hiểu nó là cấu trúc dữ liệu hàng đợi. Đặc điểm của nó là FIFO (First In First Out).

****

1. **Phân biệt ArrayList, LinkedList?**

|  |  |
| --- | --- |
| ArrayList | LinkedList |
| + Lưu trữ các phần tử theo dạng mảng (mảng động).  + ArrayList tốt hơn trong việc lưu trữ và truy xuất dữ liệu. | + Lưu trữ các phần tử thành các ô nhớ khác nhau(node) trong bộ nhớ và liên kết bởi con trỏ next (danh sách liên kết).  + LinkedList tốt hơn trong việc thao tác dữ liệu (thêm/xóa). |

1. **Phân biệt ArrayList - Array, Set – List, Override – Overload?**

|  |  |
| --- | --- |
| Array | ArrayList |
| + Kích thước cố định.  + Có thể lưu trữ kiểu nguyên thủy và đối tượng.  + Tốc độ thao tác và lưu trữ nhanh hơn.  + Chỉ có thuộc tính length(). | + Kích thước có thể thay đổi được.  + Chỉ có thể lưu trữ được kiểu đối tượng.  + Tốc độ thao tác và xử lý chậm hơn.  + Có nhiều phương thức hỗ trợ. |

|  |  |
| --- | --- |
| List | Set |
| + List có thể chứa các phần  tử trùng lặp | + Set chỉ chứa các phần tử duy nhất. |

|  |  |
| --- | --- |
| Set | Map |
| + Set chỉ chứa giá trị. | + Map chứa cặp key-value. |

|  |  |
| --- | --- |
| Overload | Override |
| + Được sử dụng để giúp code dễ đọc hơn.  + Được thực hiện bên trong0 class.  + Tham số truyền vào khác nhau.  + Nạp chồng là ví dụ về đa hình lúc biên dịch. | + Được sử dụng để cung cấp cài đặt cụ thể cho phương thức được khai báo ở lớp cha.  + Xảy ra trong hai class có quan hệ kế thừa.  + Tham số truyền vào giống nhau.  + Là ví dụ về đa hình lúc runtime. |

1. **Khái niệm về Generic, VD và lý do sử dụng?**

* Generic được ra đời từ java 5. Nó là đại diện một kiểu dữ liệu nào đó mà ta chưa biết tại thời điểm code.
* VD: List<Integer> integerBox = new ArrayList<>();

1. **Sự khác nhau giữa abstract class và interface?**

|  |  |
| --- | --- |
| Abstract class | Interface |
| + Có phương thức abstract và non-abstract.  + Abstract class không hỗ trợ đa kế thừa.  + Abstract class có các biến final, non-final, static, non-static.  + Sử dụng từ khóa abstract để khai báo.  + Là một class.  + Abstract Class không khởi tạo được đối tượng. | + Chỉ có phương thức abstract (từ java 8: default và static).  + Interface có hỗ trợ đa kế thừa.  + Chỉ có các biến static final (hằng số).  + Sử dụng từ khóa interface để khai báo.  + Là một bản thiết kế của 1 class  + Interface không khởi tạo được đối tượng. |

Khi khai báo 1 biến trong interface thì mặc định là final static mà không cần phải khai báo “final static”.

* **Khi nào dùng abstract:**

Khi bạn muốn định nghĩa một lớp cơ sở để chia sẻ phương thức và thuộc tính cho các lớp con của nó, nhưng không muốn khởi tạo đối tượng từ lớp cơ sở đó.

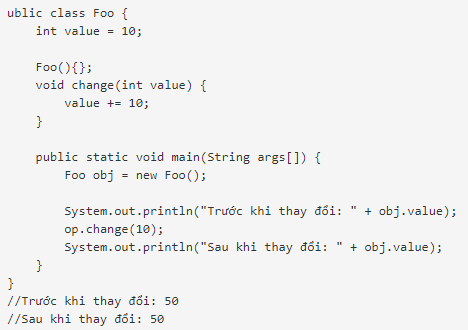
Khi bạn muốn có các phương thức trừu tượng (abstract methods) trong lớp, tức là phương thức không có thân hàm, và các lớp con sẽ phải định nghĩa lại (override) các phương thức này.

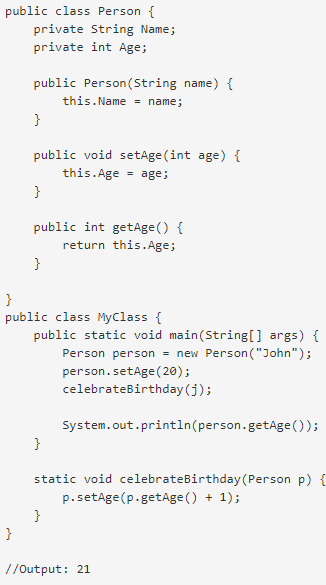
* **Khi nào dùng interface:**

Khi bạn muốn định nghĩa một bộ quy tắc (contract) mà các lớp khác nhau phải tuân theo. Các lớp triển khai (implement) một interface phải định nghĩa lại (override) toàn bộ các phương thức trừu tượng trong interface đó.

Khi bạn muốn đạt được tính đa kế thừa trong Java, vì một lớp có thể triển khai (implement) nhiều interface cùng lúc.

1. **Khái niệm tham trị và tham chiếu?**

* Tham trị tức là truyền giá trị.
  + Kiểu này dành cho các biến, các tham số khai báo kiểu dữ liệu nguyên thủy gồm: byte, short, int, long, float, double, char.
  + Nếu chúng ta gọi một phương thức và truyền một giá trị cho phương thức đó được gọi là truyền giá trị. Việc thay đổi giá trị chỉ có hiệu lực trong phương thức được gọi, không có hiệu lực bên ngoài phương thức. Vì nó chỉ tác động đến bản sao chứ không phải là địa chỉ nó trong bộ nhớ.
  + VD: ****
* Tham chiếu tức là chiếu đến 1 địa chỉ.
  + Khi chúng ta gọi một phương thức và truyền một tham chiếu cho phương thức đó được gọi là truyền tham chiếu. Việc thay đổi giá trị của biến tham chiếu bên trong phương thức làm thay đổi giá trị gốc của nó.
  + VD:

****

1. **Khái niệm Exception? Phân biệt Check và Uncheck Exception?**

* Exception trong java là một vấn đề phát sinh trong quá trình thực thi chương trình. Khi xảy ra ngoại lệ, luồng xử lý bị gián đoạn, chương trình bị dừng bất thường.
* CheckedException: là một ngoại lệ được kiểm tra và thông báo bởi trình biên dịch tại thời điểm biên dịch (compile time exception).
* UncheckedException: là một ngoại lệ được gọi tại thời gian chạy chương trình (RuntimeException).

1. **Phân biệt String Builder và String Buffer.**

|  |  |
| --- | --- |
| StringBuilder | StringBuffer |
| **+** Là không đồng bộ.  + Thích hợp làm việc với ứng dụng đơn luồng.  + Xử lý chuỗi nhanh hơn. | **+** Là đồng bộ.  + Thích hợp làm việc với ứng dụng đa luồng.  + Xử lý chuỗi chậm hơn. |

1. **Tại sao String là immutable?**

* Lí do đầu tiên là về hiệu năng (performance). Team JVM đã nhận thấy các chuỗi trong các ứng dụng thực tế hầu như là lặp lại. Do vậy thay vì tạo ra 20 instance khác nhau cho cùng 1 chuỗi thì JVM team đã thiết kế để sử dụng string pool (hay còn gọi là string constants pool) , chỉ tạo ra 1 instance duy nhất.
* Lí do thứ hai là về bảo mật (security). String là kiểu dữ liệu được sử dụng nhiều nhất trong các ứng dụng. Khi sử dụng driver hay tạo một kết nối đến URL nào đó, bạn thường phải truyền đối dưới dạng String. Nếu String không phải là immutable thì có thể gây ra một vấn đề bảo mật gọi là Pandora box.
* Để quản lý bộ nhớ hiệu quả hơn (vì không có đối tượng mới được tạo ra nếu nó đã tồn tại trong string constant pool).
* Bởi vì java sử dụng string literal. Giả sử có 5 biến tham chiếu, tất cả đều trỏ đến object string “shachin”. Nếu một tham chiếu nào đó thay đổi được giá trị của object thì nó sẽ ảnh hưởng đến các tham chiếu còn lại cùng trỏ đến object string đó.

1. **Lớp final là gì?**

* Lớp final là lớp mà trong đó các phương thức được tạo ra không thể ghi đè (Override) và lớp final không thể bị kế thừa (tức là không lớp nào kế thừa được nó nữa).

1. **Biến final (Biến hằng)**

* Biến hằng là biến không thể thay đổi giá trị (tức là từ đầu đến cuối chỉ nhận giá trị ban đầu)

1. **Phương thức final (final method)**

* Final method là phương thức mà lớp con không thể ghi đè.

(Vẫn có thể kế thừa)

1. **Biến static**

* Biến static chỉ phụ thuộc vào lớp mà không phụ thuộc vào đối tượng.
* Biến static chỉ khởi tạo một lần khi chương trình bắt đầu thực thi
* Biến static có thể truy cập trực tiếp bằng tên class mà không cần bất kỳ đối tượng nào
* Cú pháp: Tên\_class.Tên\_biến\_static

1. **Hàm static**

* Chỉ phụ thuộc vào lớp mà không phụ thuộc vào đối tượng
* Phương thức static chỉ có thể truy cập vào các thành phần static mà không thể truy cập vào các thành phần không static.
* Phương thức static có thể truy cập trực tiếp bằng tên class mà không cần bất kỳ đối tượng nào
* Cú pháp: Tên\_class.Tên\_phương\_thức\_static

1. **Lớp static**

* Một class được đặt là static chỉ khi nó là một nested class (tức là nằm trong một class khác)
* Nested static class có thể được truy cập mà không cần một object của outer class (lớp bên ngoài)

1. **Spring Framework:**
2. **Mô hình MVC là gì? Mô tả luồng đi của một ứng dụng MVC?**

* MVC là viết tắt của 3 từ Modal-View-Controller.
* Đây là mô hình thiết kế sử dụng trong kỹ thuật phần mềm.
* Luồng đi của 1 ứng dụng MVC:
  + Khi một user gửi request đến server, controller sẽ thực hiện nhiệm vụ của mình đó là tiếp nhận xử lý yêu cầu. Trong trường hợp cần thiết, nó có thể liên hệ Modal – bộ phận làm việc với database để hỗ trợ.
  + Khi xử lý xong yêu cầu, kết quả sẽ được trả về View. Tại View sẽ tạo mã HTML thành giao diện và trả về hiển thị trên trình duyệt.

1. **Các khái niệm về Dependency Injection, JPA, ORM Mapping?**

* **Dependency Injection:** là một design pattern, là một phương pháp lập trình, là một thiết kế để bạn có được hiệu quả cao khi code (giảm sự phụ thuộc giữa các class).
* **JPA:** 
  + JPA là viết tắt của java persistence API, là một đặc các tiêu chuẩn của Java để làm việc với cơ sở dữ liệu quan hệ.
  + JPA có thể xem như cầu nối giữa Java object và cơ sở dữ liệu quan hệ. Là một tập hợp các đặc tả (các interface), JPA không chứa bất kỳ phương thức thực thi nào, nó cần một JPA implementation triển khai tất cả các đặc tả mà nó định nghĩa. Những ORM tools như Hibernate, TopLink đều cung cấp trình triển khai cho JPA.
* **ORM:**
  + ORM là viết tắt của Object Relational Mapping, là công nghệ cho phép chuyển đổi dữ liệu giữa các object trong ngôn ngữ hướng đối tượng sang database quan hệ và ngược lại.
* **Hibernate:**
  + Hibernate là môt framework cho phép chúng ta kết nối với cơ sở dữ liệu và thao tác với cơ sở dữ liệu một cách nhanh chóng.
  + Hibernate giúp chúng ta có thể map các object với cở sở dữ liệu quan hệ.
  + Hibernate sẽ là một layer đứng trung gian giữa ứng dụng và database, và chúng ta sẽ giao tiếp với hibernate thay vì giao tiếp với DB.

1. **Giải thích các annotation @Controller, @Service, @Repository, @Autowire?**

* **@Controller:** là annotation được khai báo ở đầu class để đánh dấu class đó là tầng Controller (tầng giao tiếp giữa View và Model).
* **@Service:** là annotation được khai báo ở đầu class để đánh dấu class đó là tầng Service (tầng xử lý logic nghiệp vụ).
* **@Repository:** là annotation được khai báo ở đầu class để đánh dấu class đó là tầng Repository (tầng giao tiếp với cơ sở dữ liệu, phục vụ truy vấn dữ liệu).
* **@Autowire:** là annotation đánh dầu cho Spring biết rằng sẽ tự động Inject Bean tương ứng vào vị trí được đánh dấu.
* **@Component:** là một annotation đánh dấu trên các class để giúp Spring biết nó là một Bean.

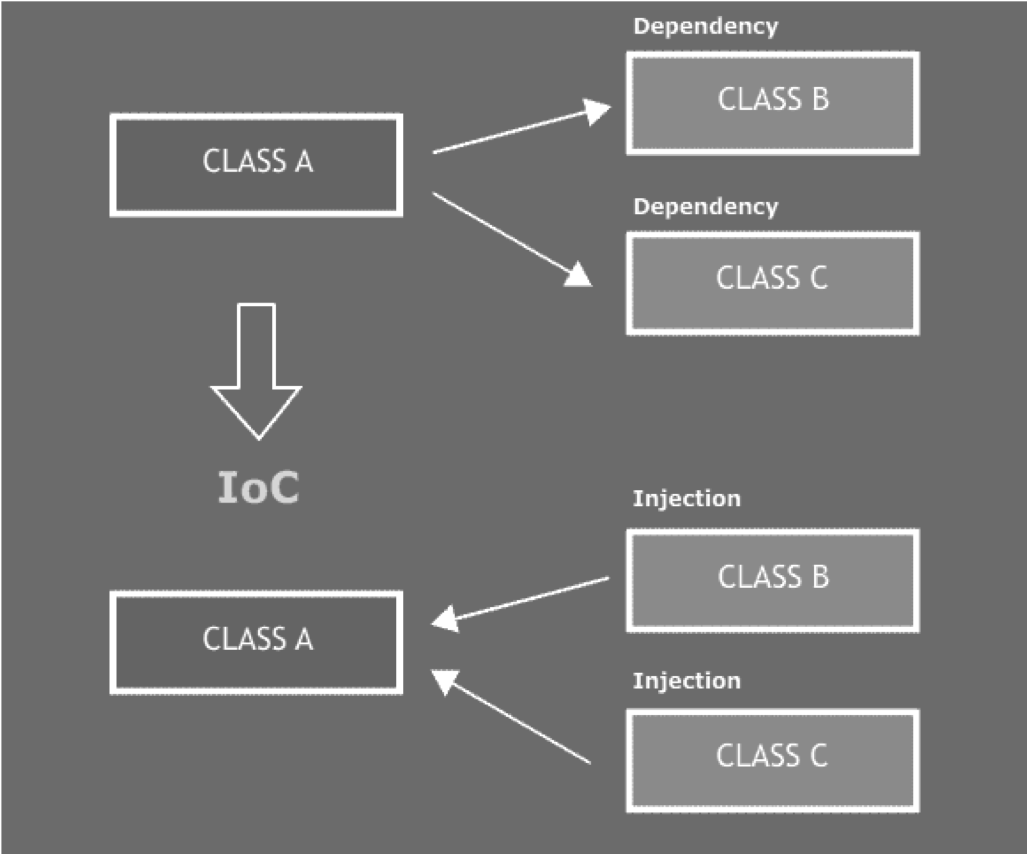
1. **Sự khác nhau giữa Session và Cookie?**

|  |  |
| --- | --- |
| **Cookie** | **Session** |
| **Cookie** được lưu trữ trên trình duyệt của người dùng. | **Session** không được lưu trữ trong trình duyệt của người dùng. |
| Dữ liệu **cookie** được lưu trữ ở phía máy khách. | Dữ liệu **session** được lưu trữ ở phía máy chủ. |
| Dữ liệu **cookie** dễ dàng sửa đổi khi chúng được lưu trữ ở phía khách hàng. | Dữ liệu **session** không dễ dàng sửa đổi vì chúng được lưu trữ ở phía máy chủ. |
| Dữ liệu **cookie** có sẵn trong trình duyệt của chúng ta đến khi hết hạn. | Dữ liệu **session** có sẵn cho trình duyệt chạy. Sau khi đóng trình duyệt sẽ mất thông tin **session**. |

1. Làm thế nào để bảo mật trong lập trình?
2. **API là gì?**

* API (Application Programming Interface): Giao diện lập trình ứng dụng. Nôm na hiểu đó là một ứng dụng thao tác với database và trả dữ liệu về cho FE.
* Các kiểu API: REST, SOAP, RESTful.
* REST API (bây giờ hay dùng):
  + Chỉ sử dụng với giao thức HTTP.
  + HTTP có các method:
    - **POST:** Thường dùng để thêm hoặc truyền dữ liệu để bảo mật **(Cho phép nhận vào một body, query string).**
    - **PUT:** Thường dùng trong trường hợp update Object **(update cả object, cho phép nhận vào một body, query string).**
    - **PATH:** Thường dùng trong trường hợp update Object **(update một số trường, cho phép nhận vào một body, query string).**
    - **GET:** Thường dùng để lấy dữ liệu **(chỉ cho phép nhận vào query string).**
    - **DELETE:** Thường dùng để xóa Object **(cho phép nhận vào một body, query string).**
* SOAP API:
  + Sử dụng với nhiều giao thức: HTTP, SMTP, FTP,…

1. **IoC là gì?**

****

* Ioc (Inversion of Control) là một nguyên lý thiết kế trong công nghệ phần mềm. Các kiến trúc phần mềm được áp dụng thiết kế này sẽ được đảo ngược quyền điều khiển so với kiểu lập trình hướng thủ tục. Trong lập trình hướng thủ tục thì các đoạn mã được thêm vào sẽ gọi các thư viện nhưng với IoC thì các IoC Container sẽ chích những dependencies vào khi nó khởi tạo bean.
* Nói 1 cách tóm tắt thì IoC là một nguyên lý thiết kế được tạo ra để các code tuân thủ theo nguyên lý Dependency Inversion
* Bình thường khi các class có quan hệ HAS-A thì trong một class sẽ chứa các thể hiện của 1 class khác. Nhưng khi 1 class nào (Class B hoặc Class C) đó thay đổi thì sẽ ảnh hưởng đến class chứa nó (Class A). Chính vì vậy IoC ra đời để làm giảm sự phụ thuộc giữa các class.
* Triển khai nguyên lý IoC thường thực hiện bởi các Framework theo từng loại ngôn ngữ lập trình, các mô hình lập trình (pattern) triển khai từ IoC như:
  + Service Locator
  + Events
  + DI (Dependency Inject)

1. **IoC Container là gì?**

* IoC Container chính là thành phần thực hiện IoC
* Trong Spring, công việc của Spring Container (IoC Container) là sẽ tạo các đối tượng rồi lắp ráp chúng lại với nhau, cấu hình các đối tượng và quản lý vòng đời của chúng từ lúc được tạo ra cho đến khi bị hủy.
* Spring Container sẽ sử dụng DI để quản lý các thành phần, đối tượng để tạo nên 1 ứng dụng. Các thành phần, đối tượng này được gọi là Spring Bean.
* Hay nó là một cái kho chứa các bean, nó sẽ chịu trách nhiệm khởi tạo, lắp ráp và quản lý vòng đời của các bean. Khi 1 class nào cần thì nó sẽ lấy bean từ trong container và inject các dependency (bean) vào thông qua setter, constructor,…

1. **Sự khác nhau giữa BeanFactory và ApplicationContext?**

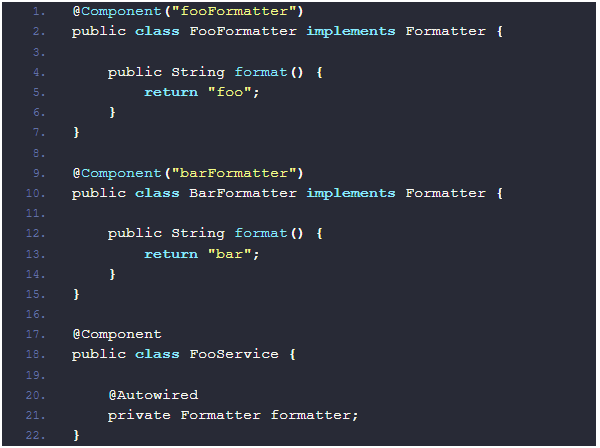
* IoC Container trong Spring có 2 kiểu là:
  + BeanFactory
  + ApplicationContext
* BeanFactory hay ApplicationContext đều là các interface thực hiện IoC Container. ApplicationContext được xây dựng như BeanFactory nhưng nó có thêm một vài chức năng mở rộng: tích hợp với Spring AOP, xử lý message và context cho web application

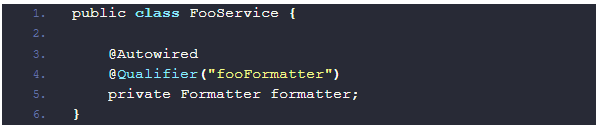
1. **Các loại scopes trong Spring bean:**

* Singleton: Một bean được định nghĩa với singleton scope sẽ khiến IoC container khởi tạo duy nhất một instance cho bean đó và nó được sử dụng trong tất cả các yêu cầu đến bean này. Đây là scope mặc định cho spring bean.
  + VD: @Bean @Scope(“singleton”) public Person personSingleton() {return new Person();}
* Prototype: Một bean với prototype scope sẽ trả về các instance khác nhau mỗi khi có một yêu cầu mới sử dụng chúng đến IoC Container
  + VD: @Bean @Scope(“prototype”) public Person personSingleton() {return new Person();}
* Request: Tương tự như prototype scope. Tuy nhiên nó được sử dụng cho các ứng dụng web. Một instance mới của bean sẽ được tạo cho mỗi yêu cầu HTTP Request.
  + VD: @Bean @RequestScope public HelloMessageGenerator requestScopedBean() {return new HelloMessageGenerator()}
* Session: Một bean mới sẽ được tạo cho mỗi phiên HTTP Session
  + VD: @SessionScope
* Global Session: Được sử dụng để tạo các global session bean cho các ứng dụng Portlet

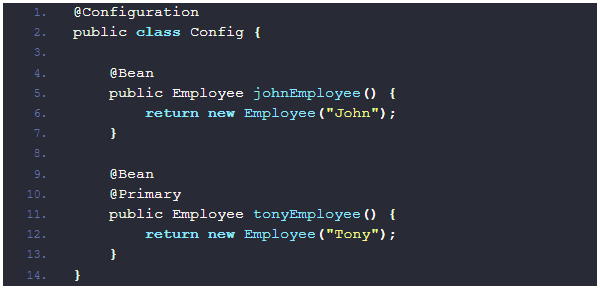
1. **Annotation @Primary và @Qualifier**

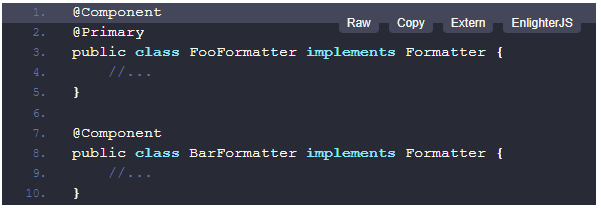
* Được sử dụng trong trường hợp có nhiều bean cùng 1 kiểu. Lúc này Spring sẽ không biết autowire bean nào dẫn đến lỗi.
* @Qualifier:





* @Primary:





* **Cần lưu ý rằng nếu cả @Primary và @Qualifier được sử dụng thì @Qualifier sẽ được ưu tiên sử dụng.** Về cơ bản, @Primary đơn giản chỉ là trường hợp mặc định, còn @Qualifier là chỉ định cụ thể.

1. **Database:**
2. **Khái niệm database?**

* **Cơ sở dữ liệu** (*Database*) là một tập hợp các [dữ liệu](https://vi.wikipedia.org/wiki/D%E1%BB%AF_li%E1%BB%87u_(m%C3%A1y_t%C3%ADnh)" \o "Dữ liệu (máy tính)) có tổ chức, thường được lưu trữ và truy cập điện tử từ hệ thống máy tính.
* VD: Thông tin của các sinh viên bao gồm: id, hoTen, soDienThoai, Email,...

1. **Các quan hệ trong database?**

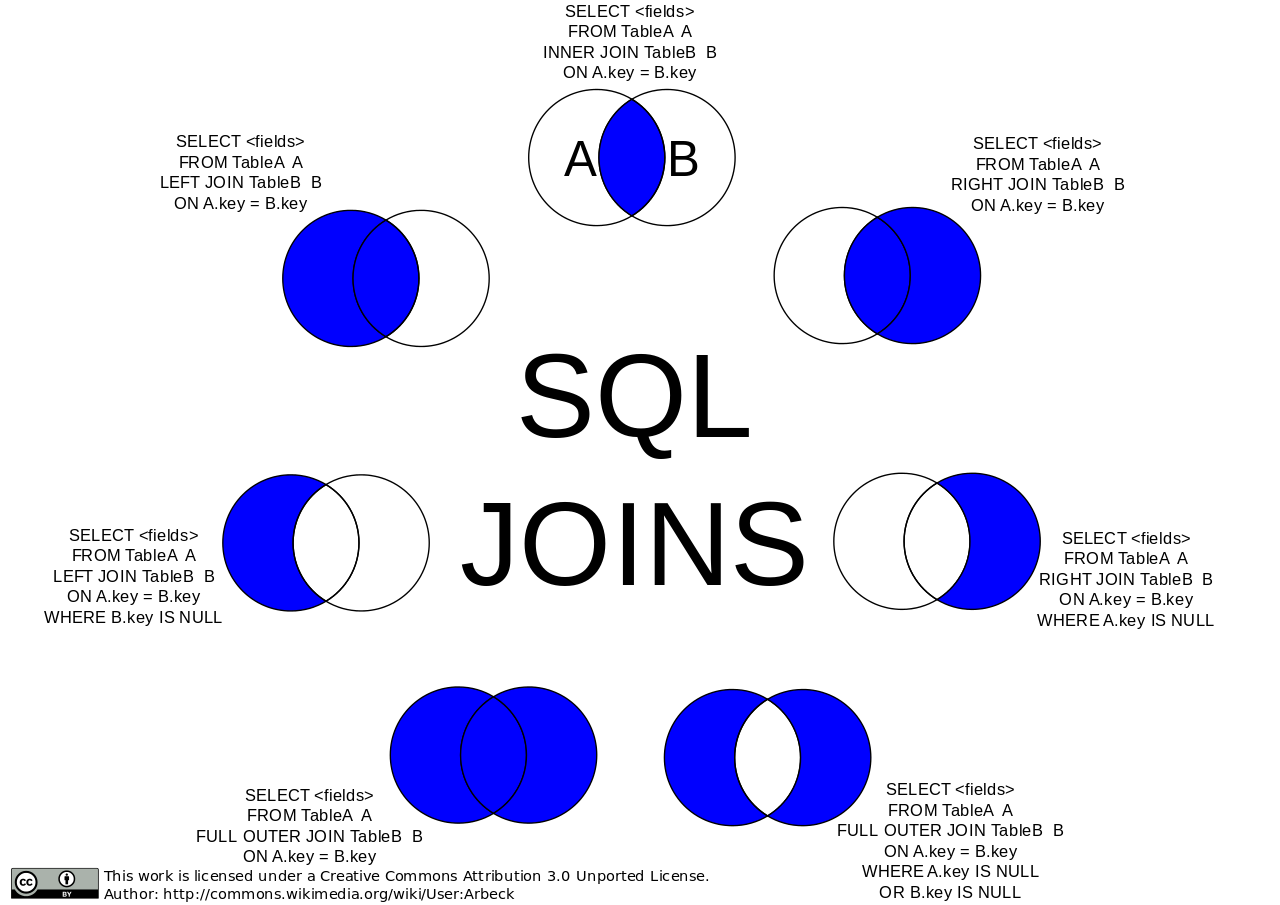
* 1-1
* 1-n
* n-1
* n-n

1. **Định nghĩa từ khóa JOIN?**

* JOIN là phép kết nối dữ liệu từ nhiều bảng lại với nhau.
* Khi bạn cần truy vấn các cột dữ liệu từ nhiều bảng khác nhau để trả về trong cùng một tập kết quả, bạn cần dùng từ khóa JOIN.

1. **Các loại join trong database?**

* INNER JOIN (JOIN)
* LEFT JOIN
* RIGHT JOIN
* FULL JOIN
* CROSS JOIN



1. **DML và DDL?**

* DML là viết tắt của Data Manipulation Language (Ngôn ngữ thao tác dữ liệu) cho phép thực thi các câu truy vấn, bao gồm cú pháp để cập nhật – sửa đổi, chèn thêm và xóa các mẩu tin. VD: INSERT, UPDATE, DELETE là các câu lệnh DML
* DDL là viết tắt của Data Definition Language (Ngôn ngữ định nghĩa dữ liệu). Dùng để định nghĩa cấu trúc của CSDL, bao gồm định nghĩa các hàng, các cột, các bảng dữ liệu, các chỉ số và một số thuộc tính khác liên quan đến CSDL như vị trí của file và là thành phần chính trong các hệ quản lý dữ liệu và có sự khác biệt rất nhiều giữa các ngôn ngữ truy vấn trên các hệ khác nhau. VD: CREATE, ALTER, DROP,…

1. **Index là gì?**

* Index (mục lục) được sử dụng để tăng tốc độ truy xuất và tìm kiếm dữ liệu.
* Nó giúp tăng tốc độ tìm kiếm nhưng sẽ làm chậm tốc độ thêm và sửa.
* Có 4 loại index:
  + Single column index.
  + Unique index. (giá trị không trùng lặp)
  + Composite index. (cú pháp mở rộng của single: đánh dấu index nhiều cột)
  + Implicit index.

1. **Các khái niệm về Composite key, Transaction, Unique?**

* Composite key là khi trong một bảng chứa hai hay nhiều hơn primary key.
* Unique là một loại ràng buộc trên column, nó xác định các dòng dữ liệu không được trùng nhau ở column đó.

1. **Khóa chính, khóa ngoại là gì?**

* **Khóa chính** là một trường hoặc nhiều trường và nó được sử dụng để xác định một bản ghi là duy nhất và nó không được null.
* **Khóa ngoại** là một trường trong bảng con và nó được tham chiếu tới khóa chính của bảng mẹ. Nó được sử dụng để liên kết các bảng dữ liệu trong CSDL.

1. **Sự khác nhau Truncate, Delete, Drop?**

* **Truncate** là xóa tất cả các records đang tồn tại trong bảng (dữ liệu không thể được phục hồi).
* **Delete** là xóa một hay tất cả các records trong một bảng theo một điều kiện nhất định (dữ liệu có thể được phục hồi).
* **Drop** được sử dụng xóa một bảng khỏi DB.

1. **Sự khác nhau giữa Having và Where?**

* **Where** filter điều kiên theo dòng.
* **Having** điều kiện theo group.

1. **Views là gì?**

* View là một bảng ảo trong CSDL có nội dụng được định nghĩa thông qua một câu lệnh SQL nào đó.

1. **Store Procudure là gì?**

* Store Procedre (thủ tục) là tập hợp các câu lệnh SQL được sử dụng để thực thi một nhiệm vụ cụ thể.
* Nó hoạt động giống như một hàm trong các ngôn ngữ lập trình khác

1. **User Define Function là gì?**

* User Define Function là hàm do người dùng tự định nghĩa bao gồm một hoặc nhiều câu lệnh SQL với mục đích sử dụng lại.
* Giống như Procedure nhưng có kết quả trả về
* Hai loại UDF:
  + **Scalar user-define function**: trả về giá trị đơn, giá trị trả về có thể thuộc một trong các dữ liêu kiểu int, char, varchar(),… Nhưng không hỗ trợ trả về kiểu text, ntext, image, timestamp.
  + **Table-valued function**:
    - **Inline table-valued functions**: hàm trả về bảng từ câu truy vấn đơn.
    - **Multi-statement table-valued functions**: hàm trả về bảng từ truy vấn nhiều câu lệnh.

1. **Trigger là gì?**

* Trigger là một Store Procedure đặc biệt, nó được thực thi một cách tự động từ phí máy chủ CSDL khi có một sự kiện thay đổi dữ liệu xảy ra như UPDATE, INSERT, DELETE.
* Hai loại trigger:
  + **Data Manipulation Language Triggers.**
  + **Data Definition Language Triggers.**

1. Giải thích các Rule chuẩn hóa dữ liệu?

Design Pattern

MVC (Model-View-Controller):

Dependency Injection:

Factory Method Pattern:

Singleton Pattern:

Template Method Pattern:

Java 8

Optional: ktra Null và tránh lỗi NullPointerEx

Các functional interface: chỉ chứa một phương thức trừu tượng (abstract method). Java 8 và các phiên bản sau đã bổ sung hỗ trợ cho "functional interface" thông qua tính năng lambda expression và functional programming.

Runnable, Callable, Comparator, Consumer, Function, Predicate, và nhiều giao diện khác trong gói java.util.function.

* Function <T,R>: Mục đích chính của Function là giúp chúng ta dễ dàng chuyển một đối tượng từ kiểu dữ liệu này sang kiểu dữ liệu khác.

Collectors: Stream.collect() là một trong các phương thức đầu cuối (terminal operation) của Stream API trong Java 8. Nó cho phép thực hiện các thao tác có thể thay đổi trên các phần tử được lưu giữ trong Stream. Chẳng hạn như: chuyển các phần tử sang một số cấu trúc dữ liệu khác, áp dụng một số logic bổ sung, tính toán, …

Sắp xếp trong Java 8:

* Arrays.sort().
* Collections.sort().
* Nếu muốn so sánh 2 đối tượng bất kỳ thì kết hợp implement Comparable và Comparator