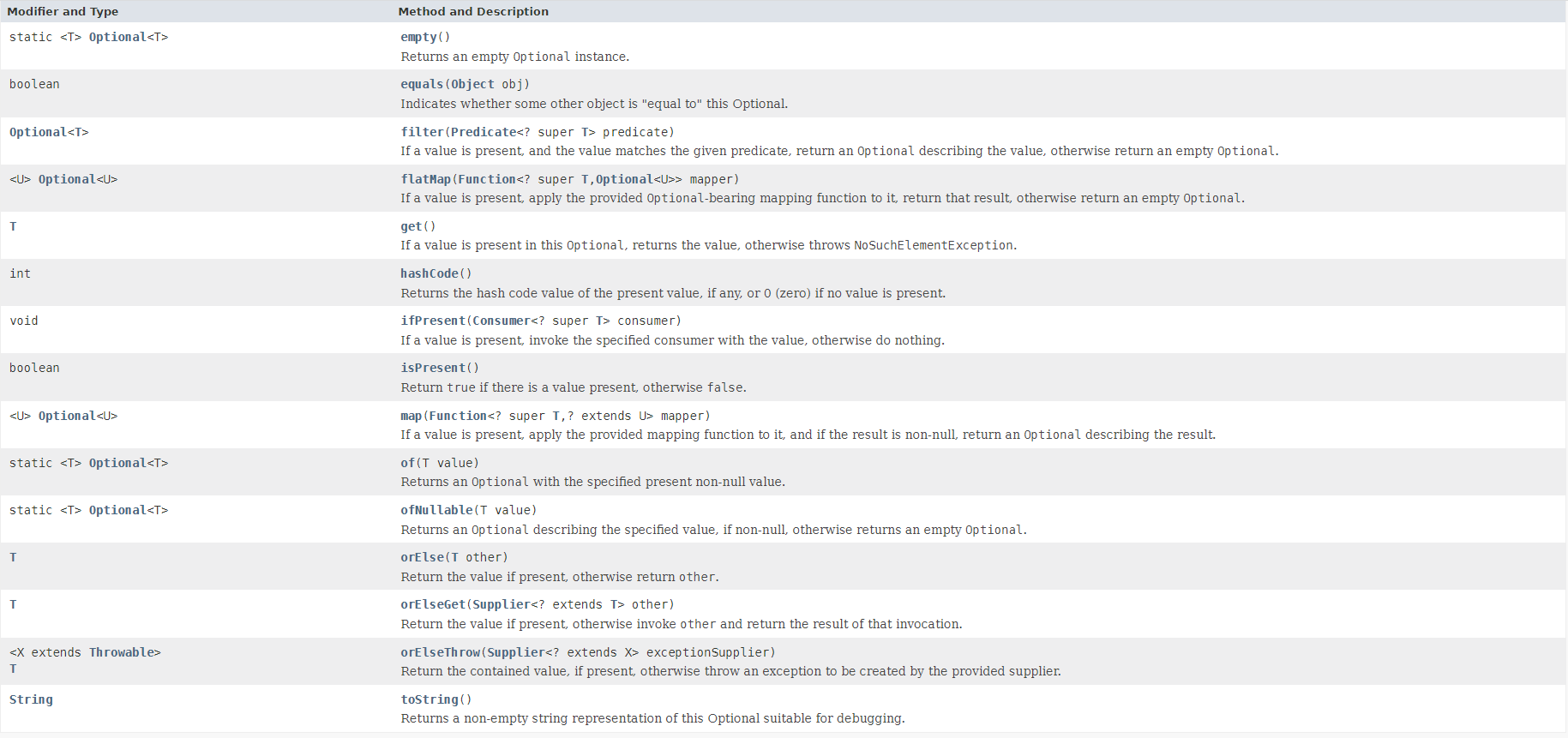
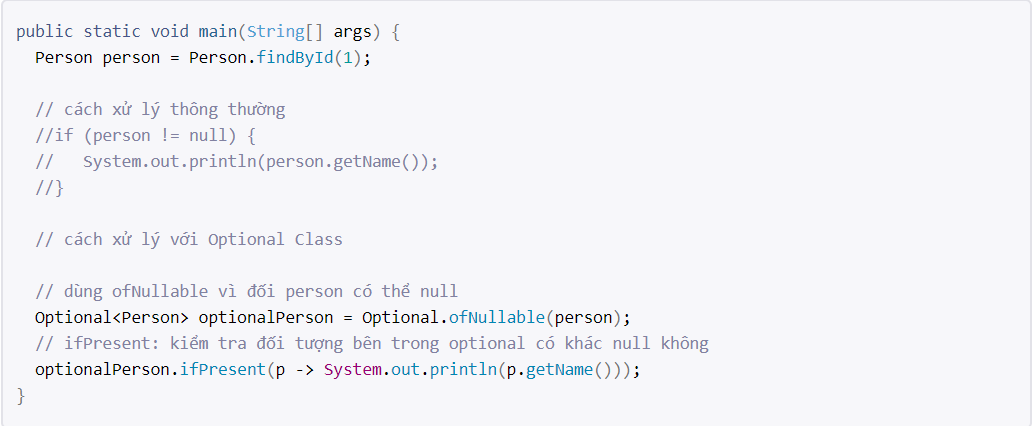
**1 Optional class?**

**Optional là 1 đối tượng generic, bản chất của Optional là một container (bao chứa) chứa đối tượng mà nó reference tới.**

**Nó có thể rỗng hoặc chứa giá trị NULL (trường hợp đối tượng mà nó reference tới bị null). Cấu trúc này đảm bảo các biến được sử dụng thông qua Optional Class sẽ tránh được lỗi NullPointerException.**



Cách dùng



**DI là gì? (Dependence Inject)**

- là 1 kỹ thuật, 1 design pattern cho phép xóa bỏ sự phụ thuộc hard-code và làm cho ứng dụng của bạn dễ mở rộng và maintain hơn.

Các phương pháp thực hiện Dependency Injection

- Constructor Injection: Các dependency sẽ được truyền vào (inject vào) 1 class thông qua constructor của class đó. Đây là cách thông dụng nhất. (ví dụ trên mình dùng theo cách này)

- Setter Injection: Các dependency sẽ được truyền vào 1 class thông qua các hàm Setter/Getter

**IoC là gì?** (**Inversion of Control)**

Inversion of Control (Đảo chiều sự điều khiển) là engine nạp bean từ bên ngoài vào hệ thống nhằm cho phép điều khiển theo mã tuỳ biến của bean bên ngoài.

Những IoC container sẽ chích những dependencies khi khởi tạo bean.

**Spring Container**

- Spring IOC Container là trái tim của Spring

- Nó có nhiệm vụ quản lý vòng đời của bean (các đối tượng trong dự án spring), khởi tạo, cấu hình, và tương tác giữa các bean trong ứng dụng Spring

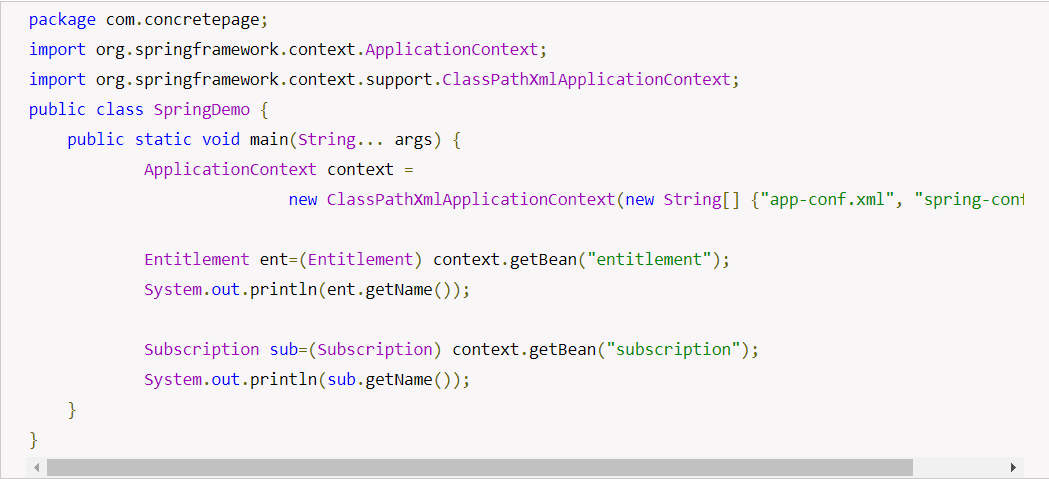
Khởi tạo Spring Container

Có 3 cách khởi tạo Spring Container

- AnnotationConfigApplicationContext: Sử dụng khi chúng ta viết 1 chương trình Java độc lập. Không phải là Java web mà chỉ là ứng dụng java thông thường.



- ClassPathXmlApplicationContext: Nếu như ta sử dụng XML để cấu hình cho Spring thì ta dùng ClassPathXmlApplicationContext để nạp các cấu hình đó thông qua file XML.



- FileSystemXmlApplicationContext: Cũng tương tự như ClassPathXmlApplicationContext nhưng file cấu hình chúng ta không phải là XML và chúng ta chỉ định đường dẫn để nạp file đó là ở đâu.

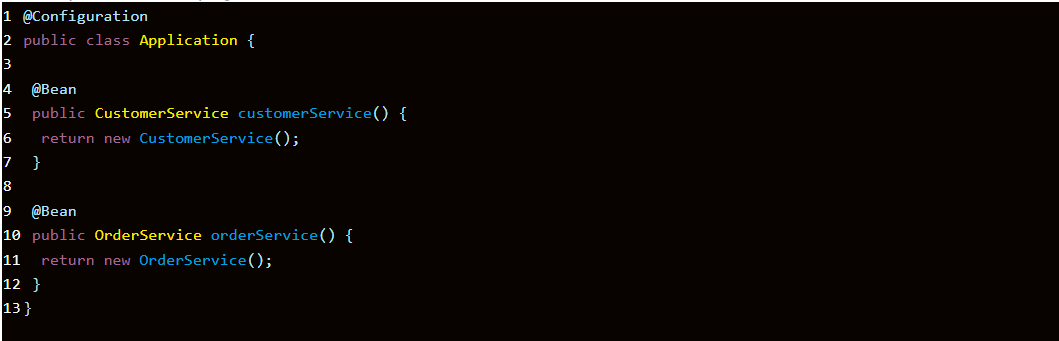


**Có bao nhiêu cách tạo bean?**

*Cách 1:* Dùng annotation trên class

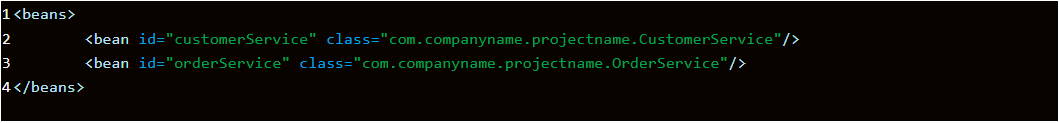
@Component, …

Cách 2: Dùng @Configuration và @Bean



Cách 3: Khai báo trong file XML

Tạo Bean bằng XML



**Java Bean là gì?**

JavaBean là các lớp Java thuần, dùng để thể hiện các thực thể trong các chương trình Java.

Cụ thể, các lớp:

có các hàm tạo public mặc định (không có đối số)

cho phép truy cập vào các thuộc tính của họ bằng cách sử dụng phương thức truy cập (getter và setter)

triển khai thực hiện java.io.Serializable

Serialization trong Java là cơ chế chuyển đổi trạng thái của một đối tượng (giá trị các thuộc tính trong object) thành một chuỗi byte sao cho chuỗi byte này có thể chuyển đổi ngược lại thành một đối tượng.

- Khả năng tuần tự hóa của một lớp được kích hoạt bởi lớp triển khai giao diện java.io.Serializable. Các lớp không triển khai giao diện này sẽ không có bất kỳ trạng thái nào của chúng được tuần tự hóa hoặc giải mã hóa. Tất cả các kiểu con của một lớp có thể tuần tự hóa đều có thể tự tuần tự hóa. Giao diện tuần tự hóa không có phương thức hoặc trường nào và chỉ dùng để xác định ngữ nghĩa của việc có thể tuần tự hóa.

**Spring Bean là gì?**

Spring Beans chính là những Java Object mà từ đó tạo nên khung sườn của một Spring Application. Chúng được cài đặt, lắp ráp và quản lý bởi Spring IoC container. Những bean này được tạo ra bởi configuration metadata được cung cấp từ container, ví dụ, trong tag nằm trong file XML.

Một Spring bean về cơ bản là một đối tượng do Spring quản lý. Cụ thể hơn, nó là một đối tượng được khởi tạo, cấu hình và quản lý bởi một vùng chứa Spring Framework. Spring bean được định nghĩa trong các tệp cấu hình Spring (hoặc gần đây là với các chú thích), được khởi tạo bởi các vùng chứa Spring, và sau đó được đưa vào các ứng dụng.

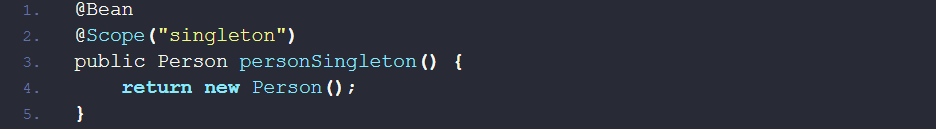
**5. spring bean có mấy scope?**

Có 5 scope được định nghĩa cho Spring Bean:

* Singleton: Chỉ duy nhất một thể hiện của bean sẽ được tạo cho mỗi container. Đây là scope mặc định cho spring bean. Khi sử dụng scope này cần chắc chắn rằng các bean không có các biến/thuộc tính được share.
* Prototype: Một thể hiện của bean sẽ được tạo cho mỗi lần được yêu cầu(request)
* Request: giống với prototype scope, tuy nhiên nó dùng cho ứng dụng web, một thể hiện của bean sẽ được tạo cho mỗi HTTP request.
* Session: Mỗi thể hiện của bean sẽ được tạo cho mỗi HTTP Session
* Global-Session: Được sử dụng để tạo global sesion bean cho các ứng dụng Portlet.

Trong đó có Singleton, Prototype là hay được sử dụng

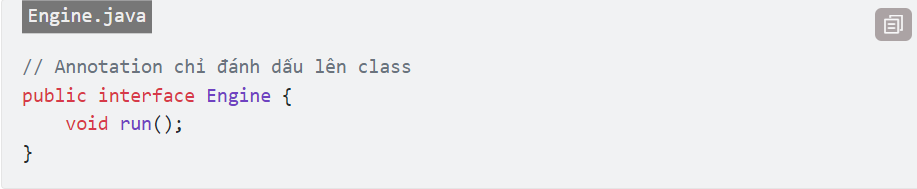
Cách dùng: @Scope (“...”)

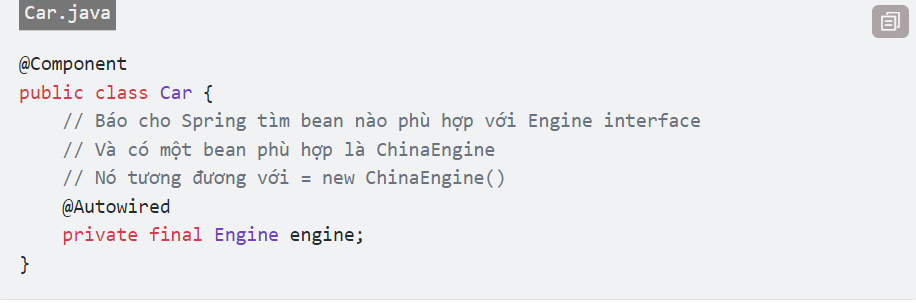


6. Trường hợp nhiều class implement 1 interface thì phải xử lý thế nào

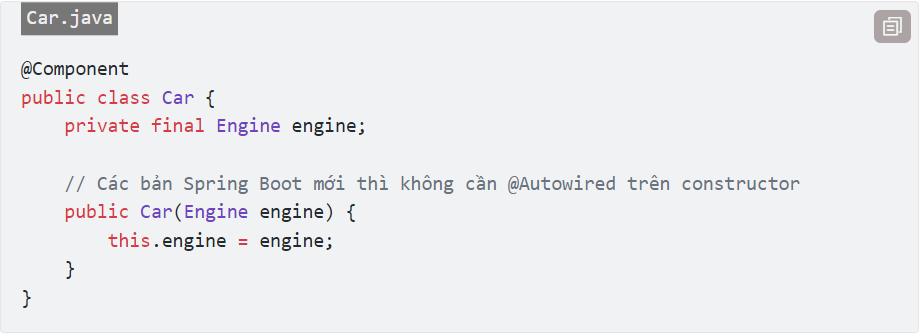
**7. có mấy cách inject bean trong spring?**

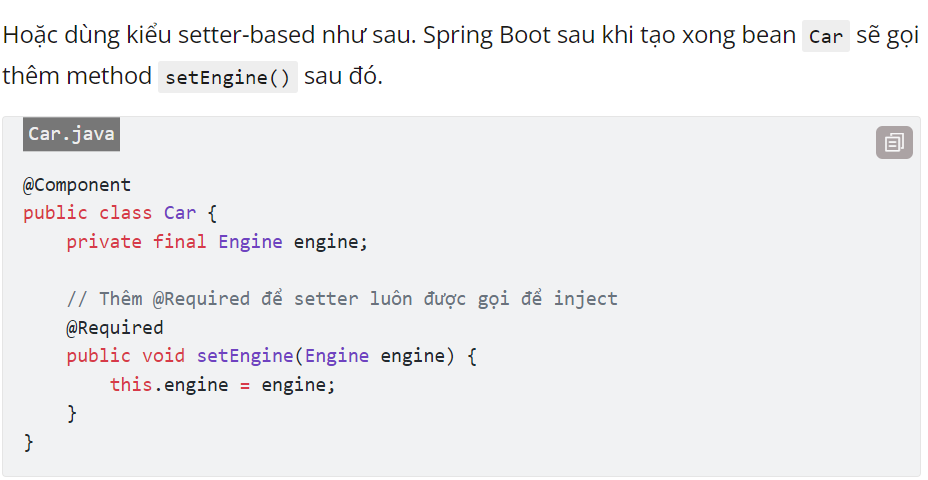
Sử dụng @Autowired





Inject qua constructor hoặc setter





- Method injection

- Field injection

**8. @Autowired là inject kiểu gì? cách nào nên dùng nữa?**

@Autowire: tự động tìm bean và gán vào khi dùng

- @Required (Spring 2.0)

- @Inject (Spring 3.0)

**Buổi 5**

**Phân trang bằng Pageable**

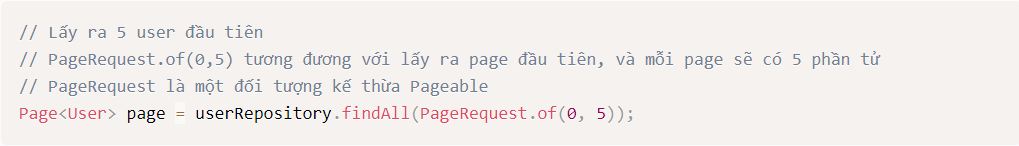
Tại sao cần phân trang?

- Vì nó giúp tiết kiệm băng thông và tăng tốc xử lý, ngoài ra nó cũng giảm thiểu việc hiển thị các thông tin thừa không cần thiết.

JPA Pageable

Để có thể query lấy dữ liệu theo dạng Page.

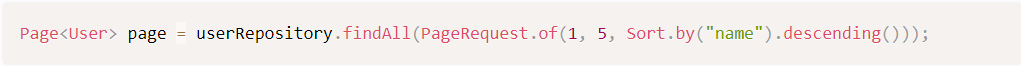
Hàm findAll (Pageable pageable) là có sẵn và trả về đối tượng Page<T>



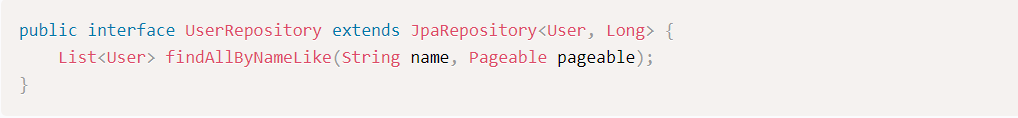
page.nextPageable()

để lấy ra đối tượng tiếp theo

Sorting

****

Có thể tự custom để hàm trả về Page<T>, Slice<T>, List<T>.

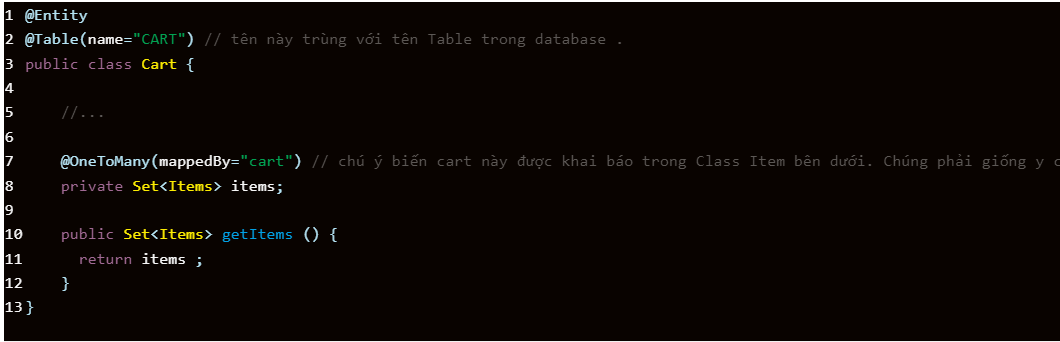
****

Tham khảo: <https://loda.me/articles/jpahuong-dan-query-phan-trang-bang-pageable-phan-1>

**@OneToMany**

Đánh dấu quan hệ 1 nhiều với đối tượng dưới

Vd:



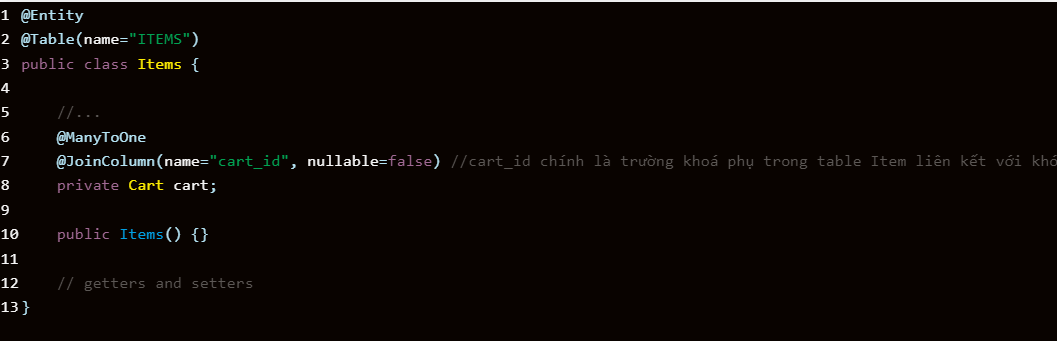
Ta sử dụng @OneToMany trong Class Cart, điều đó có nghĩa 1 giỏ hàng sẽ có nhiều sản phẩm (Items).

MappedBy dùng để định nghĩa Class Cart và Class Item sẽ liên kết với nhau thông qua tên “cart”.

MappedBy giống như là 1 cầu nối để ta có thể từ Class Cart mình gọi hàm getItems mình sẽ nhận được một danh sách Items

**@ManyToOne**

Đánh dấu quan hệ nhiều - 1



chúng ta sử dụng @ManyToOne và @JoinColumn để định nghĩa cho biến “cart” để tạo sự liên kết ngược lại giữa Class Items và Cart

@JoinColumn ta định nghĩa name = “cart\_id”. Cái ‘cart\_id ‘ chính là column khóa phụ trong table Items mà ta định nghĩa trong database . nullable = false là ta ràng buộc dữ liệu không được phép null

**Buổi 6**

1. **Lazy loading**

Lazy Loading được hiểu nôm na chính là việc load dữ liệu khi bạn cần sử dụng đến chúng. Ví dụ như khi nhiều người cùng vào 1 page sẽ không kéo xuống hết đến cuối trang để đọc nội dung thì điều bạn cần làm là load nội dung trước.

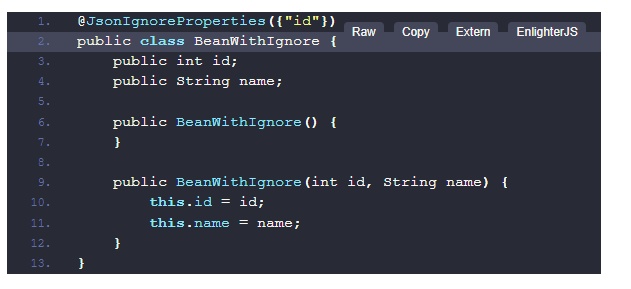
Việc nên làm đó chính là khi người dùng scroll đến đâu bạn sẽ load dữ liệu đến đó. lazy loading có thể áp dụng cho bất cứ resource nào trên 1 page, thậm chí là cả file JavaScript.

* Lazy loading qua @JsonIgnore

@JsonIgnoreProperties

*@JsonIgnoreProperties* là 1 class-level anntation dùng để đánh dấu các thuộc tính sẽ bị Jackson bỏ qua trong serialization và deserialization.

Ví dụ chúng ta sẽ ignore thuộc tính *id* trong *BeanWithIgnore* class.



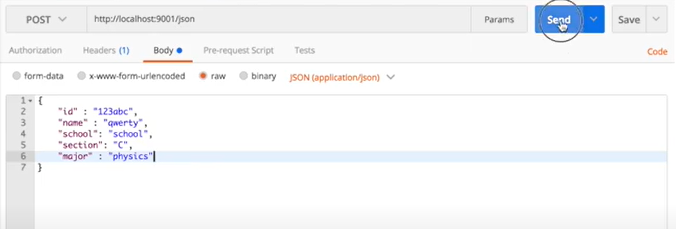
**@JsonIgnore**

Tương tự như @JsonIgnoreProperties, thế nhưng @JsonIgnore cho phép sử dụng với field-level. Trong đa số trường hợp, chúng ta nên sử dụng @JsonIgnore thay cho @JsonIgnoreProperties để tránh trường hợp phải sửa lại các thuộc tính trong @JsonIgnoreProperties khi thay đổi tên các thuộc tính của class.

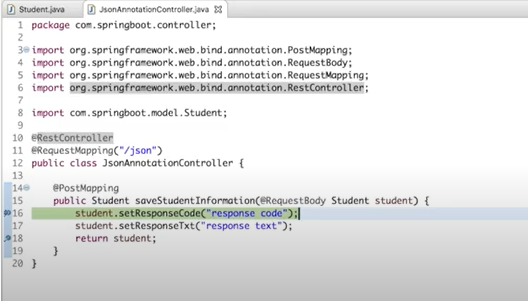
Ta có thể ví dụ như sau :



Ở đây ta xét cho @JsonIgnoreProperties của class student ở các field là major và reponseTxt, ngoài ra ta có xét @JsonIgnore cho section field.



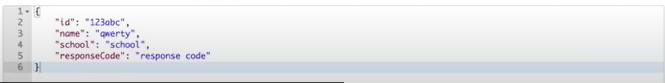
Trong JsonAnnotationController :



Khi return ra student :



Khi trả về hiển thị :



Ta có thể thấy các field ta đã chỉ định bằng @JsonIgnoreProperties Annotation là major và reponseTxt trả về giá trị là null, ngoài ra section field mà ta sử dụng @JsonIgnore cũng dc xét null khi send

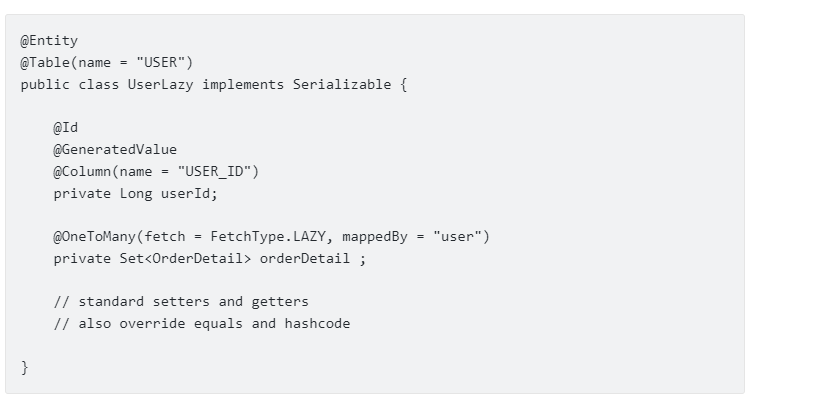
=>> Nếu 1 class có rất nhiều field ta ko nên thêm @JsonIgnore cho từng field được mà nên sử dụng @JsonIgnoreProperties.

* Eager Loading

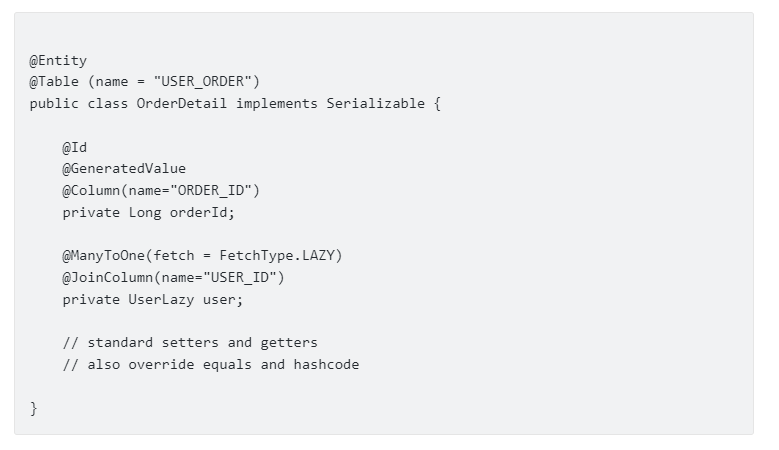
**Eager Loading** được thiết kế giúp cho việc khởi tạo đối tượng ngay lập tức.  
 **Lazy Loading** được thiết kế giúp cho việc trì hoãn việc khởi tạo đối tưởng miễn là có thể

Ví dụ:

Class UserLazy



Class Order Detail



OrderDetail Annotation [@OneToMany](https://viblo.asia/u/OneToMany) định nghĩa quan hệ giữa UserLazy và OrderDetail là 1-n.  
Trong đó có thuộc tính fetch:  
fetch = FetchType.LAZY tức là khi bạn find, select đối tượng **UserLazy**  
   
từ database thì nó sẽ không lấy các đối tượng **OrderDetail**liên quan nếu bạn sử dụng:  
  
fetch = FetchType.EAGER tức là khi bạn find, select đối tượng UserLazy từ database thì tất cả các đối tượng OrderDetail liên quan sẽ được lấy ra và lưu vào set orderDetail

**Lưu ý**  
fetch = FetchType.LAZY tức là mặc định không lấy ra các đối tượng quan hệ dù trong cùng một transaction chỉ khi bạn gọi getOrderDetail() thì lúc này truy vấn lấy dữ liệu danh sách orderDetail sẽ được thực hiện.  
 Nếu bạn không gọi getOrderDetail() thì list này sẽ không có dữ liệu và kết thúc transaction sẽ không có danh sách orderDetail nào cả =))

fetch = FetchType.EAGER thì khi lấy đối tượng UserLazy là nó mặc định query luôn các đối tượng OrderDetail liên quan và lưu vào set orderDetail, do đó khi kết thúc transaction, set orderDetail sẽ có chứa các đối tượng UserLazy của OrderDetail đó.

Mặc đinh:  
annotation [@ManyToOne](https://viblo.asia/u/ManyToOne) và [@OneToOne](https://viblo.asia/u/OneToOne) thì fetchType EAGER là mặc định  
annotation [@ManyToMany](https://viblo.asia/u/ManyToMany) và [@OneToMany](https://viblo.asia/u/OneToMany) thì fetchType LAZY là mặc định

**@JsonManagedReference**

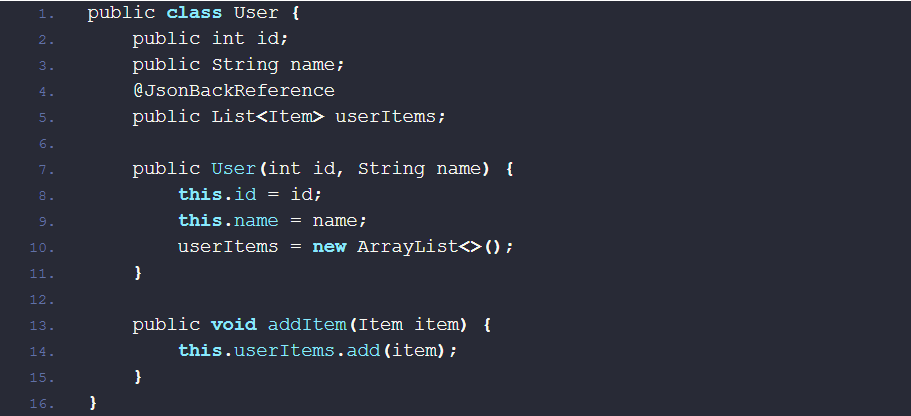
- được xem như là thành phần chính trong mối quan hệ trong serialization, vì vậy chúng sẽ được serialized bình thường.

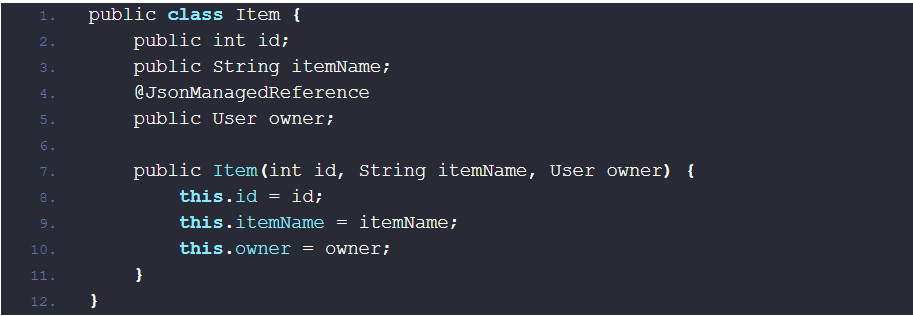
**@JsonBackReference**

- được xem như phần phụ trợ, và nó sẽ bị lược bỏ trong serialization để tránh tình trạng lặp vô tận.

Ví dụ:

sử dụng @JsonManagerReference và @JsonBackReference để chú thích cho User và Item class.







Với việc sử dụng @JsonManagedReference và @JsonBackReference có một nhược điểm cực kỳ lớn đó là chúng ta phải thay đổi vị trí đặt @JsonManagedReference để có thể serialize chứa đầy đủ thông tin.

**@JsonIndentityInfor** sinh ra nhầm giải quyết vấn đề này, nó cho phép serialize theo mã định danh khi chúng gặp lại nhau lần thứ 2 trong serialization từ đó giải pháp đệ quy vô tận

**CORS (Cross Origin Resource Sharing)**

- là một tính năng mới được tích hợp trong HTML5, thêm vào các HTTP headers chỉ dẫn cho trình duyệt web về sử dụng và quản lý nội dung cross-domain, cho phép lấy dữ liệu từ một trang khác thông qua XMLHttpRequest.

CORS hoạt động ra sao?

Khi trình duyệt gửi một request đến một domain khác để yêu cầu làm một việc gì đó, những request này sẽ được gắn thêm một header có tên là origin để xác định origin của client, giá trị này được thêm tự động bởi trình duyệt và không ai có thể thay đổi nó được. Header này đại diện cho nguồn gốc truy vấn.

Orgin được cấu tạo dựa trên ba phần:

Protocol/Scheme: (Http/Https)

Host: server/domain

Port: cổng, nếu giá trị này là giá trị mặc định 80 thì không cần

Tham khảo thêm: <https://viblo.asia/p/cors-la-gi-Qbq5Q0j3lD8>