Ưu điểm của Spring Framework:

* Lightweight: Rất nhẹ những phần core của nó chưa đến 2mb tức là những gói Jar của nó khi mà chúng ta sử dụng trong spring framework
* Predefined Templates để cho phép người dùng dễ dàng khởi tạo 1 project để có thể đưa vào dự án nhanh chóng
* Loose Coupling with IoC and DI: Về mặt design Framework cho phép chúng ta có những thiết kế Loose Coupling như là IoC và DI
* Easy to Test:
* Fast Development: Chúng ta có thể dễ dàng khởi tạo ra các project ngoài ra tại thời điểm này chúng ta có spring boot cho phép tạo ra application trong spring với một số bước đơn giản trông thời gian ngắn.
* Powerful Abstraction:việc sử dụng IoC và DI mà spring framework có 1 tính Abstraction rất mạnh bởi vì nó chủ yếu sử dụng Interface để giao tiếp.
* Declarative support : Spring framework cho phép chúng ta khai báo thông qua XML
* Aspect Oriented Programming (AOP): Spring framework cũng hỗ trợ cho AOP – lập trình hướng sự kiện
* Well-designed Web MVC framwork
* Transaction Management:
* Security:
* Various Products:

# **Dependency Injection trong Spring**

**Dependency Injection (DI)** trong Spring là một mẫu thiết kế được sử dụng để loại bỏ sự phụ thuộc giữa các mã chương trình, giúp cho việc quản lý và kiểm thử ứng dụng dễ dàng hơn. Dependency Injection làm cho mã chương trình ít bị phụ thuộc vào nhau hơn.

# IoC Containers trong Spring

IoC có nghĩa là một object không tạo ra các object khác nếu muốn sử dụng thay vào đó nó sẽ sử dụng các object khác thông qua việc truyền vào từ bên ngoài, ví dụ các thuộc tính của class sẽ được truyền vào từ bên ngoài thông qua các hàm khởi tạo hoặc setter thay vì khởi tạo trực tiếp trong class,

Đây 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 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

Khác nhau giữa BeanFactory và ApplicationContext là: BeanFactory chỉ tạo ra đối tượng chúng ta cần khi chúng ta gọi phương thức getBean() còn ApplicationContext sẽ tạo hết tất cả các đối tượng chúng ta cần ngay khi bạn gọi đến Spring container.

BeanFactory: Đây là container đơn giản nhất cung cấp hỗ trợ cơ bản cho DI.

ApplicationContext:

# Bean trong spring

Các đối tượng tạo thành xương sống của ứng dụng và được quản lý bởi Spring IoC container được gọi là Bean. Một bean là một đối tượng được khởi tạo, lắp ráp và được quản lý bởi một Spring IoC container. Các bean này được tạo ra bằng siêu dữ liệu cấu hình mà bạn cung cấp cho container.

# Phân biệt java beans và spring beans

Java Beans: là các class mà có hàm khởi tạo không tham số được đặt public từ đó nó gọi các hàm khởi tạo này để tạo ra các object, java beans phải kế thừa interface Serializable để có thể truyền qua mạng theo cơ chế remove hoặc lưu xuống các file để đọc lại 1 cách bình thường.

# Phạm vi của Bean trong spring

Khi định nghĩa một bean trong Spring, bạn có các tùy chọn để khai báo phạm vi của bean đó. Ví dụ, bạn muốn Spring tạo ra một bean mới khi cần thiết thì bạn nên khai báo thuộc tính phạm vi của bean là prototype. Tương tự như vậy, nếu bạn muốn Spring trả về cùng một bean (bean duy nhất) khi cần thiết, bạn nên khai báo thuộc tính phạm vi của bean là Singleton

5 Phạm vi:

* Singleton: Đây là phạm vi mặc định, phạm vi này nói với Spring IoC container rằng chỉ tạo ra một thể hiện duy nhất của bean này trên toàn bộ dứng dụng
* Prototype: Phạm vi này nói với Spring IoC container rằng tạo ra một thể hiện mới của bean mỗi khi cần thiết.
* Request: Phạm vi này định nghĩa một bean cho một HTTP request. Chỉ có giá trị trong ngữ cảnh của một ứng dụng Spring Web.
* Session: Phạm vi này định nghĩa một bean cho một HTTP session. Chỉ có giá trị trong ngữ cảnh của một ứng dụng Spring Web.
* Global-session: Phạm vi này định nghĩa một bean cho một global HTTP session. Chỉ có giá trị trong ngữ cảnh của một ứng dụng Spring Web.

# Vòng đời của Bean

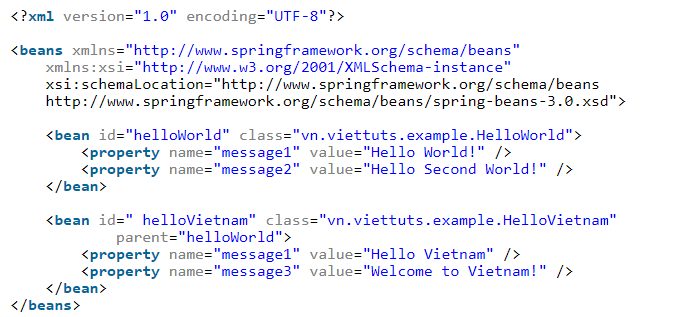
VÒng đời của Bean trong Spring bao gồm khởi tạo, sử dụng và kết thúc. Mặc dù, có một danh sách các hoạt động xảy ra đằng sau thời điểm bắt đầu và trước khi bean bị hủy.

# Kế thừa Bean trong Spring

Kế thừa Bean trong Spring, định nghĩa bean con thừa kế dữ liệu cấu hình từ định nghĩa bean cha. Định nghĩa con có thể ghi đè lên một số giá trị hoặc thêm các giá trị khác nếu cần.

Kế thừa Bean trong Spring không liên quan gì đến việc kế thừa lớp Java nhưng khái niệm kế thừa là giống nhau. Bạn có thể định nghĩa một định nghĩa bean cha làm mẫu và các bean con khác có thể kế thừa các cấu hình cần thiết từ bean cha.

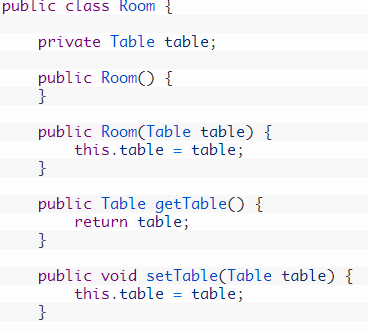
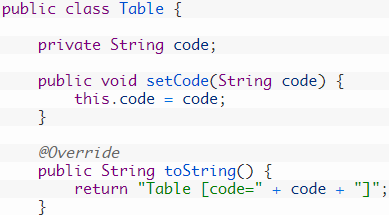
Định nghĩa bean cha trong bean con bằng thuộc tính parent

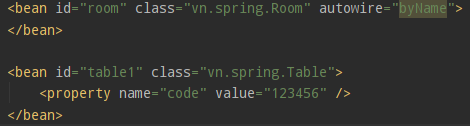


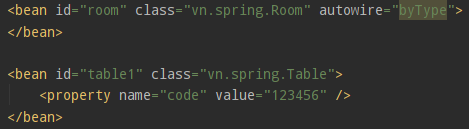
# Bean Autowiring

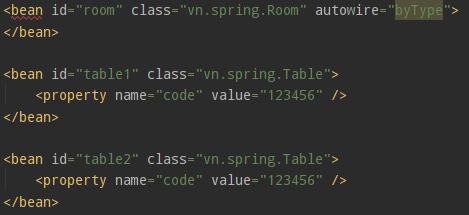
Với Bean Autowiring, chúng ta ko cần phải sử dụng các thuộc tính **ref**, **idref** hay **list,**mà chỉ cần viết code tuân thủ theo quy tắc của Spring thì đối tượng phụ thuộc sẽ tự động được đưa vào đối tượng bị phụ thuộc.

Có ba cách để sử dụng Bean Autowiring, đó là:

* byName: chúng ta khai báo thêm thuộc tính **autowire=”byName”** thì Spring sẽ tự động đưa đối tượng phụ thuộc vào đối tượng bị phụ thuộc qua phương thức setter của đối tượng bị phụ thuộc.  
    
    
  



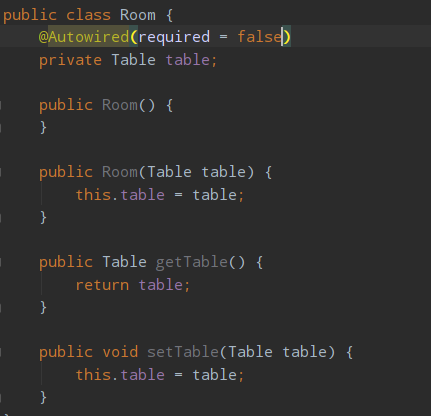
* byType: Cách này chúng ta chỉ áp dụng được với những project nhỏ khi mà mỗi đối tượng chỉ có một bean duy nhất trong khung chứa của Spring. Khi đó bằng cách thay đổi thuộc tính autowire=”byType” khi khai báo một bean, Spring sẽ tự động đưa đối tượng phụ thuộc vào đối tượng bị phụ thuộc cũng qua phương thức setter của đối tượng bị phụ thuộc.  
    
  

Lỗi:   


Bản chất của Spring là tạo ra các đối tượng trong container của nó để khi các bạn cần thì có thể gọi đến đối tượng bạn cần. Vì thế, Spring tìm mọi cách để hỗ trợ chúng ta có thể tạo ra đối tượng trong khung chứa của nó một cách dễ dàng nhất.

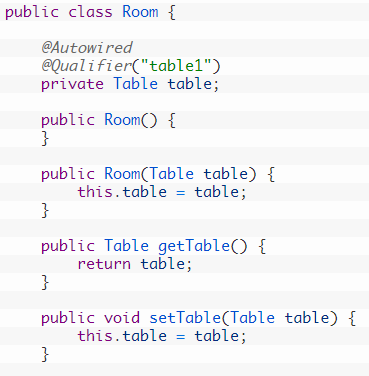
Chúng ta đã biết cách khai báo để đưa đối tượng này vào đối tượng khác, hay có thể nói là đưa đối tượng phụ thuộc vào đối tượng bị phụ thuộc, bằng cách sử dụng thuộc tính **ref, idref, list …**Ngoài ra, Spring còn hỗ trợ chúng ta đưa đối tượng phụ thuộc vào đối tượng bị phụ thuộc một cách tự động bằng cách sử dụng Bean Autowiring của nó.

# Spring - Annotation Based Configuration

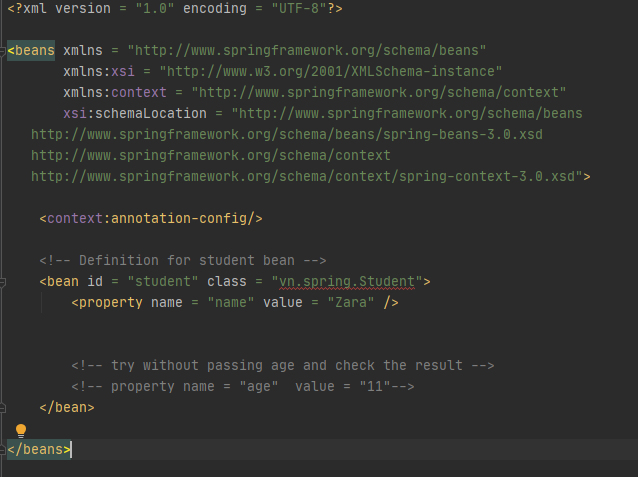
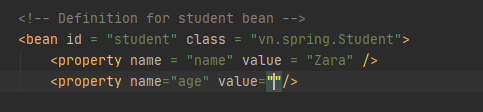
* @Autowired  
    
     
  required = true, nếu spring container không tìm thấy bean address để inject vào thì nó sẽ báo lỗi
* required = false, nếu Spring container không tìm thấy bean address để inject vào thì nó sẽ inject null

## **@Autowired with (required = false)**

@Qualifie  
  
 trong trường hợp autowire byType, chúng ta không thể khai báo nhiều hơn 1 bean của cùng một đối tượng phụ thuộc trong Spring container. Bởi vì lúc này Spring không biết phải sử dụng bean nào để mà autowire.

Vậy có giải pháp nào khi chúng ta cần khai báo nhiều bean cho cùng một đối thuộc mà vẫn autowire được hay không?  
Để giải quyết lỗi này, chúng ta sẽ sử dụng annotation **@Qualifier** để chỉ định rõ bean nào của đối tượng Table sẽ được autowire vào bean của đối tượng Room bằng cách chỉnh sửa class Room   


## @Required

# @Confrigation và @Bean

* @Configuration là một Annotation đánh dấu trên một Class cho phép **Spring** biết được đây là nơi định nghĩa ra các Bean.
* @Bean là một Annotation được đánh dấu trên các method cho phép **Spring** biết được đây là Bean và sẽ thực hiện đưa Bean này vào Context.
* @Bean sẽ nằm trong các class có đánh dấu @Configuration