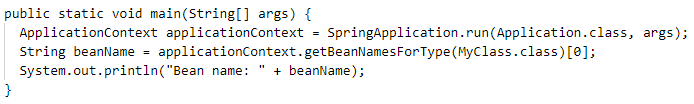
**BEAN**

**@Component**

* **@Component** được sử dụng để đánh dấu một class là một bean có thể được quản lý bởi Spring. Khi một class được đánh dấu bằng **@Component**, Spring sẽ tạo một instance của class đó bằng cách sử dụng constructor mặc định (default constructor) và quản lý nó trong container của mình.
* Khi khởi tạo một bean với constructor có tham số trong Spring, các tham số của constructor thường cần phải là các bean khác đã được quản lý bởi Spring container.
* Khi sử dụng **@Component**, chúng ta có thể tự định nghĩa tên cho bean thông qua thuộc tính *value*, ví dụ: **@Component("myBean")**. Nếu chúng ta không chỉ định tên, Spring sẽ tạo tên bean mặc định dựa trên tên class, ví dụ: MyClass -> myClass.



**Mô hình quản lý bean trong Spring**

* ApplicationContext và Context: Khi chạy một ứng dụng Spring Boot, container chính của Spring được gọi là ApplicationContext được tạo ra. Container này có tác dụng quản lý một không gian lưu trữ gọi là Context. Container được tạo ra sẽ tìm kiếm các Bean trong ứng dụng để đưa vào Context lưu trữ
* Bean: trong Spring là các đối tượng Java thông thường, được tạo ra bởi container và quản lý bởi container. Các bean này có thể là các class đánh dấu bằng các annotation như **@Component**, **@Controller**, **@Service**, **@Repository** hoặc được định nghĩa bằng cách sử dụng XML hoặc Java Config
* Quản lý vòng đời bean: Container cũng quản lý vòng đời của các bean, tức là quản lý quá trình khởi tạo, sử dụng và huỷ bỏ các bean. Container sẽ tạo ra các bean khi cần thiết, tiêm các phụ thuộc (dependencies) vào các bean, và cung cấp các bean cho các thành phần khác. Khi ứng dụng kết thúc hoặc bean không còn cần thiết, container sẽ huỷ bỏ các bean để giải phóng tài nguyên
* *Vậy dùng new như trước vẫn ok?* Đối với trường hợp new bình thường, giả sử ta có 1 class A có chứa nhiều properties là các object B, C, E, … và ta cần phải tạo ra object A thì trước đó ta phải tạo ra các object B, C, E, … để inject vào, mà biết đâu B, C, E, … lại phụ thuộc vào n object khác => Việc tạo ra 1 object hoàn chỉnh chứa đầy đủ các dependency trở nên khó kiểm soát. Còn đối với cơ chế bean của spring ta chỉ cần dependency injection để yêu cầu Spring tạo và cung cấp các bean phụ thuộc. Spring sẽ quản lý vòng đời của các bean, tiêm các phụ thuộc vào các bean và cung cấp các bean khi được yêu cầu

**Chi tiết về bean**

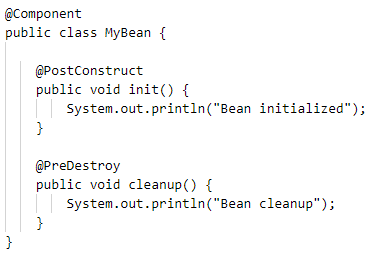
* Khi chúng ta yêu cầu Spring trả về một bean bằng cách sử dụng getBean(), nếu chúng ta chỉ định một interface làm kiểu của bean, Spring sẽ tự động tìm kiếm và trả về một bean implement (thực hiện) interface đó
* **@PostConstruct** và **@PreDestroy**

**@PostConstruct**:

* + **@PostConstruct** được đặt trước một phương thức trong một bean. Phương thức này sẽ được gọi tự động sau khi bean được khởi tạo và trước khi nó được sử dụng.
  + **@PostConstruct** thường được sử dụng để thực hiện các tác vụ khởi tạo, cấu hình, hoặc chuẩn bị trước khi bean được sử dụng.

**@PreDestroy**

* + @PreDestroy được đặt trước một phương thức trong một bean. Phương thức này sẽ được gọi tự động trước khi bean bị hủy (destroyed) hoặc trước khi ứng dụng dừng.
  + **@PreDestroy** thường được sử dụng để thực hiện các tác vụ dọn dẹp, đóng kết nối, hoặc giải phóng tài nguyên trước khi bean bị hủy.



A screenshot of a computer screen

Description automatically generated with low confidence

A screen shot of a computer program

Description automatically generated with low confidence

A screen shot of a computer

Description automatically generated with low confidence

**@Configuration và @Bean**

Các annotation này cho phép tạo các bean trong Spring

* **@Configuration** là một annotation được đánh dấu trên một class, cho biết rằng đó là một nơi để định nghĩa các bean. Nó thông báo cho Spring rằng class này chứa các cấu hình và phải được xem như một nguồn cấu hình bean
* **@Bean** là một annotation được đánh dấu trên các phương thức, cho biết rằng phương thức này sẽ tạo ra một bean và đưa nó vào context. **@Bean** thường được sử dụng trong các class có đánh dấu **@Configuration** để cung cấp các định nghĩa bean cụ thể
* Với **@Configuration** và **@Bean**, chúng ta có toàn quyền kiểm soát quá trình tạo bean và có thể sử dụng các constructor tùy ý để khởi tạo bean
* Trong trường hợp phương thức định nghĩa tham số và tham số chưa có giá trị, nếu trong context đã tồn tại một bean tương ứng với tham số đó, Spring sẽ tự động tiêm bean đó vào tham số của phương thức. Tuy nhiên, nếu không có bean tương ứng trong context, Spring sẽ ném ra một exception vì không thể tìm thấy bean để tiêm vào tham số

**@Conditional**

* **@Conditional** là một annotation được sử dụng để điều kiện hóa việc cấu hình các bean trong ứng dụng. Nó cho phép chúng ta chỉ định một điều kiện hoặc một tập hợp các điều kiện, và bean chỉ được tạo ra nếu các điều kiện đó được đáp ứng
* **@Conditional** có thể được sử dụng trên một lớp (class) hoặc một phương thức (method). Khi được đặt trên một lớp, nó cho phép Spring tạo một lớp cấu hình (configuration class) dựa trên một điều kiện nào đó. Khi được đặt trên một phương thức, nó cho phép Spring tạo một bean dựa trên một điều kiện nào đó
* **@Conditional** nhận một tham số là một mảng các lớp (classes) mở rộng từ interface Condition. Interface Condition có duy nhất một phương thức là matches, trả về kiểu boolean. Nếu tất cả các phương thức matches của các lớp Condition trả về true, điều đó đồng nghĩa với việc Spring sẽ tạo bean hoặc cấu hình tương ứng. Ngược lại, nếu ít nhất một phương thức này trả về false, Spring sẽ không tạo bean hoặc cấu hình nào cả
* Ngoài **@Conditional**, Spring cung cấp một số annotation hỗ trợ khác để tạo bean dựa trên các điều kiện nhất định như: **@ConditionalOnBean**, **@ConditionalOnProperty**, **@ConditionalOnExpression**, …

A screen shot of a computer code

Description automatically generated with medium confidence

A picture containing text, screenshot, font, number

Description automatically generated

**@Lazy**

**@Lazy** được giới thiệu từ phiên bản Spring 3.0, được dùng để chú thích các bean chỉ được khởi tạo ở lần đầu tiên được yêu cầu

* **@Lazy** với **@Configuration** và **@Bean**

Khi chúng ta chú thích **@Lazy** trên một **@Configuration** class, nghĩa rằng tất cả các method được chú thích **@Bean** bên trong sẽ được lazy-init (Chỉ khởi tạo tại lần đầu tiên được yêu cầu)

**A picture containing text, screenshot, font, number

Description automatically generated**

Để áp dụng Bean được chỉ định riêng, đơn giản chỉ cần đặt **@Lazy** vào Bean được yêu cầu

A screenshot of a computer code

Description automatically generated with medium confidence

* **@Lazy** với **@Component**

**A picture containing text, screenshot, font, line

Description automatically generated**

* **@Lazy** với **@Autowired:** Chú ý rằng trong trường hợp này **@Lazy** được đặt cả ở **@Component** và **@Autowired**

**A picture containing text, screenshot, font, number

Description automatically generated**

**@DependsOn**

* **@DependsOn** được sử dụng để chỉ định rằng một bean phải được tạo trước các bean khác trong context của Spring. Điều này đảm bảo rằng các bean được đánh dấu **@DependsOn** sẽ có sự phụ thuộc đúng thứ tự khi Spring khởi tạo các bean
* Giá trị trong **@DependsOn** là một mảng chuỗi các tên bean

**A screenshot of a computer code

Description automatically generated with medium confidence**

**@ComponentScan**

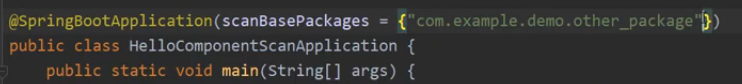
Mặc định, container của Spring Boot sẽ tìm kiếm các bean trong package chứa class main và các package con của nó. Nếu muốn tùy chỉnh phạm vi tìm kiếm của container:

* Sử dụng **@ComponentScan**

Text

Description automatically generated

* Sử dụng thuộc tính scanBasePackages của **@SpringBootApplication**



**@Autowired**

Là một annotation trong Spring Framework được sử dụng để tiêm (inject) các dependency vào các bean. Khi sử dụng **@Autowired**, Spring sẽ quét qua context để tìm các bean phù hợp với kiểu dữ liệu của dependency và tiêm (inject) nó vào

* Có 3 vị trí sử dụng **@Autowired**:
  + Thuộc tính:

A screenshot of a computer code

Description automatically generated with low confidence

* + Constructor:

A picture containing text, screenshot, font, line

Description automatically generated

Chú ý: Nếu class chỉ có một constructor và không có bất kỳ annotation **@Autowired** nào được đặt lên, Spring vẫn sẽ tự động inject dependency vào constructor đó

A picture containing text, screenshot, font, line

Description automatically generated

Trường hợp này thông thường sử dụng **@RequiredArgsConstructor** của Lombok + final

A picture containing text, screenshot, font, line

Description automatically generated

* + Setter:

A picture containing text, screenshot, font

Description automatically generated

* Khi khai báo một dependency là một interface và sử dụng **@Autowired** để tiêm nó vào một bean khác, container của Spring sẽ tìm kiếm một bean implement (thực thi) interface đó và tiêm nó vào. Nếu có nhiều bean implement interface đó, chúng ta có thể sử dụng các annotation khác như **@Primary** hoặc **@Qualifier** để chỉ định rõ ràng bean nào sẽ được tiêm vào. Nếu không sẽ nhận thông báo lỗi nhiều có nhiều hơn một implement của interface đó
* Cách Autowired bằng thuộc tính được sử dụng phổ biến

**@Scope**

* **@Scope** được sử dụng để xác định phạm vi (scope) của một bean trong container. Annotation này cho phép chúng ta kiểm soát vòng đời và cách thức tạo ra các instance của bean

Text

Description automatically generated

* Các giá trị có thể nhận của **@Scope**: singleton(mặc định), prototype, request, session

**@Primary và @Qualifier**

* **@Primary**: Ưu tiên lấy bean có đánh dấu là **@Primary**

A picture containing text, screenshot, font, line

Description automatically generated

* **@Qualifier**: Để sử dụng cách này đầu tiên ta cần định danh cho bean, sau đó sử dụng **@Qualifier** đi kèm với **@Autowired**

**A screenshot of a computer program

Description automatically generated with medium confidence**