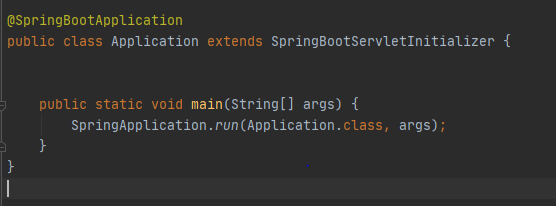
# @Component và @Autowired



Một trong những nhiệm vụ chính của Spring là tạo ra một cái Container chứa các Dependency cho chúng ta.

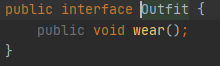
SpringApplication.run(App.class, args) chính là câu lệnh để tạo ra container. Sau đó nó tìm toàn bộ các dependency trong project của bạn và đưa vào đó.

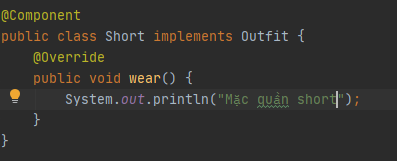
Spring đặt tên cho container là ApplicationContext

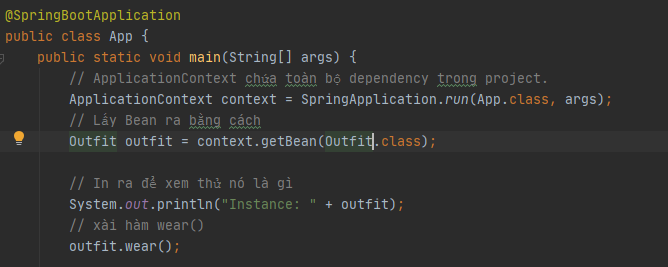
và đặt tên cho các dependency là Bean

Vậy làm sao Spring biết đâu là Bean? Chúng ta tới với khái niệm @Component

## @ Component





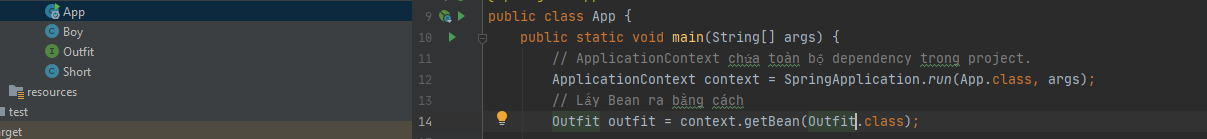


# 

Bạn sẽ thấy Outfit lúc này chính là Short. Class đã được đánh dấu là @Component.

Spring Boot khi chạy sẽ dò tìm toàn bộ các Class cùng cấp hoặc ở trong các package thấp hơn so với class App mà bạn cung cấp cho Spring (Chúng ta có thể cấu hình việc tìm kiếm này, sẽ đề cập sau). Trong quá trình dò tìm này, khi gặp một class được đánh dấu @Component thì nó sẽ tạo ra một instance và đưa vào ApplicationContext để quản lý.

Quá trình





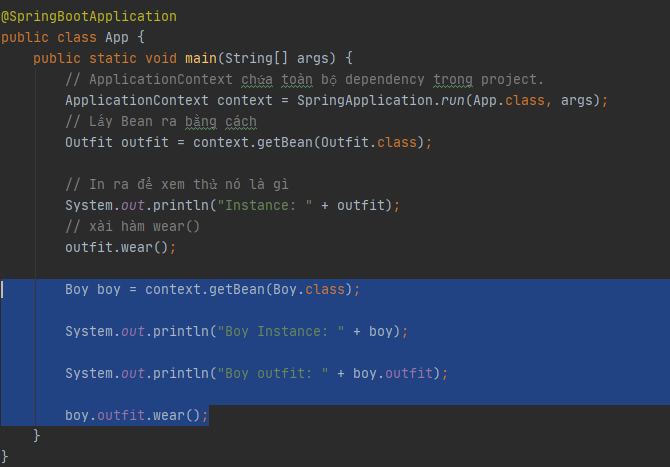
## @Autowire

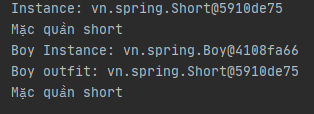
Bây giờ mình tạo ra một Class Boy và có một thuộc tính là Outfit.

Mình cũng đánh dấu Boy là một @Component. Tức **Spring Boot** cần tạo ra một instance của Boy để quản lý.

## 

Tôi đánh dấu thuộc tính Outfit của Boy bởi Annotation @Autowired. Điều này nói với Spring Boot hãy tự inject (tiêm) một instance của Outfit vào thuộc tính này khi khởi tạo Boy.

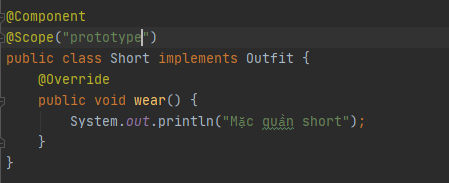




## Singleton

Điều đặc biệt là các Bean được quản lý bên trong ApplicationContext đều là singleton. Bạn chắc đã để ý điều này từ các Output ở phía trên.

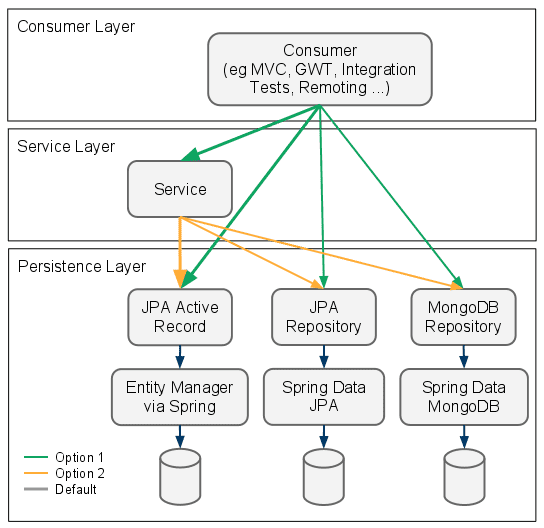
Nếu ko thích thì để prototype



# Kiến trúc trong Spring Boot

Kiến trúc MVC trong Spring Boot được xây dựng dựa trên tư tưởng "độc lập" kết hợp với các nguyên lý thiết kế hướng đối tượng. Độc lập ở đây ám chỉ việc các layer phục vụ các mục đích nhất định, khi muốn thực hiện một công việc ngoài phạm vi thì sẽ đưa công việc xuống các layer thấp hơn.

Kiến trúc Controller-Service - Repository chia project thành 3 lớp:



* Consumer Layer hay Controller: là tầng giao tiếp với bên ngoài và handler các request từ bên ngoài tới hệ thống.
* Service Layer: Thực hiện các nghiệp vụ và xử lý logic
* Repository Layer:: Chịu trách nhiệm giao tiếp với các DB, thiết bị lưu trữ, xử lý query và trả về các kiểu dữ liệu mà tầng Service yêu cầu.

## [#](https://loda.me/spring-boot-4-component-vs-service-vs-repository-loda1557627097246/#controller-vs-service-vs-repository)@Controller vs @Service vs @Repository

Để phục vụ cho kiến trúc ở trên, Spring Boot tạo ra 3 Annotation là @Controller vs @Service vs @Repository để chúng ta có thể đánh dấu các tầng với nhau.

Trong bài này, chúng ta sẽ tìm hiểu 2 Annotation @Service vs @Repository trước.

@Service Đánh dấu một Class là tầng Service, phục vụ các logic nghiệp vụ.

@Repository Đánh dấu một Class Là tầng Repository, phục vụ truy xuất dữ liệu.

Về bản chất @Service và @Repository cũng chính là @Component. Nhưng đặt tên khác nhau để giúp chúng ta phân biệt các tầng với nhau.

Về bản chất thì bạn có thể sử dụng thay thế 3 Annotation @Component, @Service và @Repository cho nhau mà không ảnh hưởng gì tới code của bạn cả. Nó vẫn sẽ hoạt động.

Tuy nhiên từ góc độ thiết kế thì chúng ta cần phân rõ 3 Annotation này cho các Class đảm nhiệm đúng nhiệm vụ của nó.

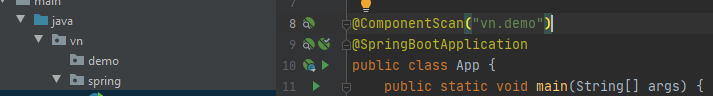
* @Service gắn cho các Bean đảm nhiệm xử lý logic
* @Repository gắn cho các Bean đảm nhiệm giao tiếp với DB
* @Component gắn cho các Bean khác.

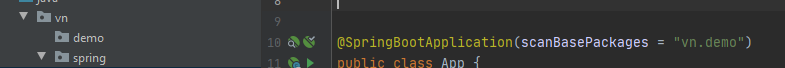
# @Component Scan

Spring Boot khi chạy sẽ dò tìm toàn bộ các Class cùng cấp hoặc ở trong các package thấp hơn và tạo ra Bean từ các Class tìm thấy

Trong trường hợp bạn muốn tuỳ chỉnh cấu hình cho Spring Boot chỉ tìm kiếm các bean trong một package nhất định thì có các cách sau đây:

1. Sử dụng @ComponentScan



1. Sử dụng scanBasePackages tromg @SpringBootApplication.  
   

# @Configuration và @Bean

Nhiều bạn sẽ tự hỏi rằng @Configuration và @Bean sẽ có ý nghĩa gì khi chúng ta đã có @Component? Sao không đánh dấu SimpleBean là @Component cho nhanh?

Các bạn thắc mắc rất đúng, và việc sử dụng @Component cũng hoàn toàn ổn.

Thông thường thì các class được đánh dấu @Component đều có thể tạo tự động và inject tự động được.

Tuy nhiên trong thực tế, nếu một Bean có quá nhiều logic để khởi tạo và cấu hình, thì chúng ta sẽ sử dụng @Configuration và @Bean để tự tay tạo ra Bean. Việc tự tay tạo ra Bean như này có thể hiểu phần nào là chúng ta đang *config* cho chương trình.

# Tạo ứng dụng đơn giản để hiểu @Controller



