**@Component** : Giúp Spring biết để quản lý **instance** của class nào **(Bean)** và tạo **instance** của class đó cho mình.

**@Autowired** : Spring sẽ tìm trong các **Component** khớp với **Dependencies** và inject vào.

**@ComponentScan** : Component Scan mặc định sẽ scan ở package mà @SpringBootApplicationđược đặt. Muốn scan ở package khác thì dùng @ComponentScan(‘Path package’).

Khi có trên 2 implementation từ 1 interface ta phải chỉnh autowire cho Component để tránh conflict:

Có 3 cách : **@Primary** (có độ ưu tiên cao hơn và hiệu quả hơn).

**Name** (chỉ cần đổi tên biến).

**@Qualifier**.

Trong Spring các Bean mặc định là **Singleton**.

**@Scope**: - Singleton (mỗi lần gọi đều trả về 1 instance).

* Prototype (mỗi lần gọi trả về 1 instance khác).

**@PostConstruct**: Ngay khi Bean được khởi tạo cùng với các dependency thì PostConstruct sẽ được gọi. Ta có thể config 1 số thứ trong này.

**@PreDestroy**: Method có annotation này sẽ được gọi trước khi Bean bị hủy.

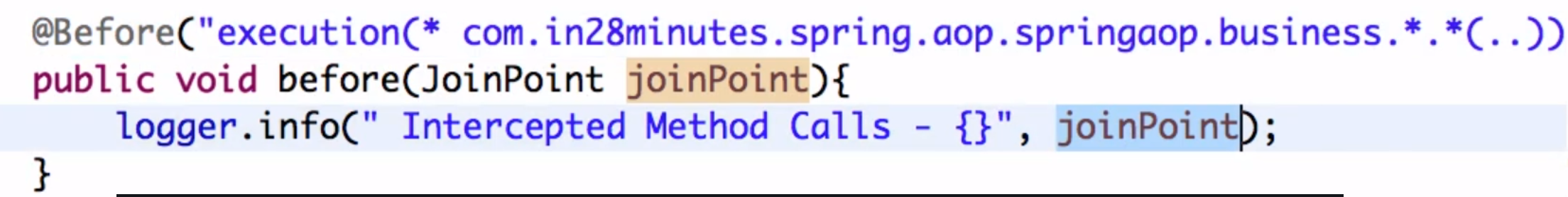
**@Value(“${…}”)**: lấy giá trị từ file properties.

**@PropertySource(“classpath:\*name file\*”)**: Config đường dẫn tới file property nào.

**@Configuration:** được dùng để định nghĩa class chứa cấu hình của Spring Beans.

**AOP FRAMEWORK**

**@Before(“execution(\* PACKAGE.\*.\*(..)”):** Method với annotation này sẽ được gọi trước khi 1 method trong package chỉ định được thực thi.

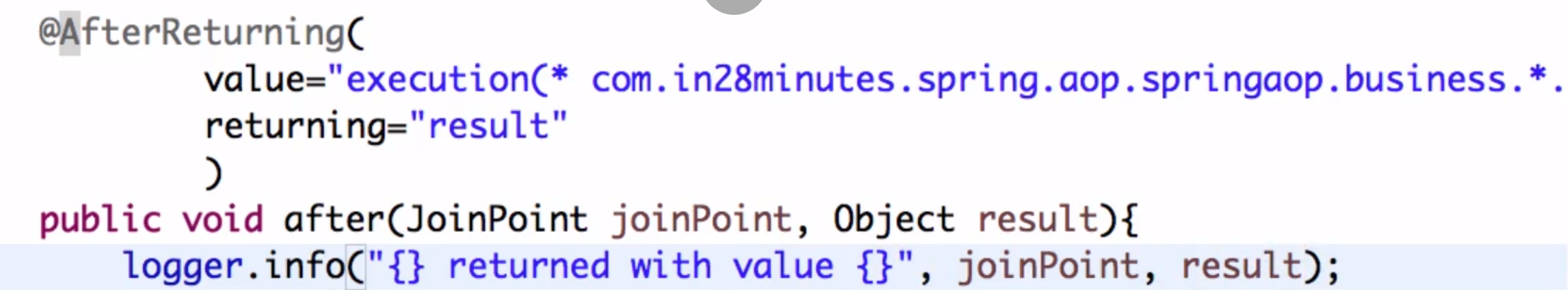


**@Before** thường dc sử dụng để check quyền access của user. Giúp mình implement sercurity ở 1 chỗ thay vì ở nhiều point của methods.

**Aspect** là sự kết hợp giữa **PointCut** và **Advice**.

**JoinPoint** là sự can thiệp thực thi cụ thể nào đó.

**@AfterReturning(“execution(\* PACKAGE.\*.\*(..)”)**: Method với annotation này sẽ được gọi sau khi kết quả của 1 method được trả về.

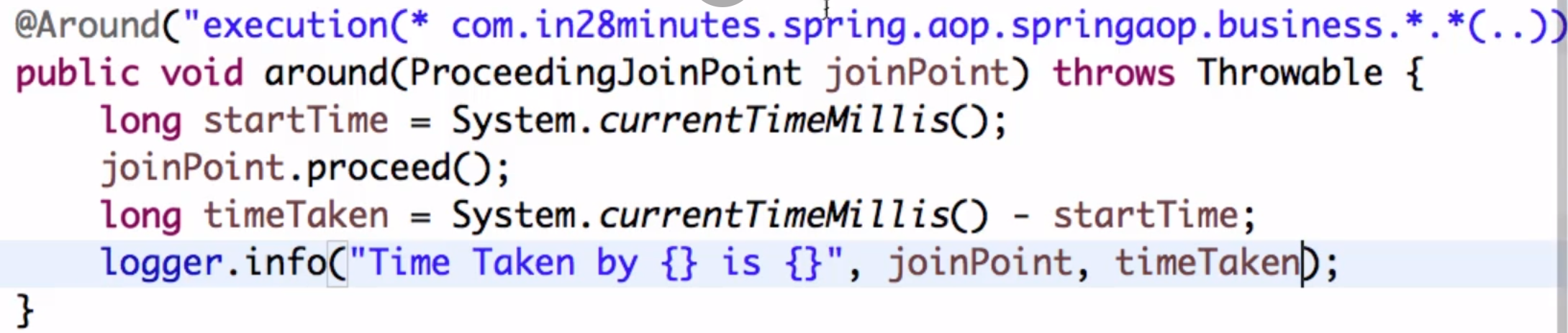


**@AfterReturning**: Được gọi khi method đó thực thi thành công.

**@AfterThrowing**: Được gọi khi method throw exception.

**@After**: Được gọi với cả 2 trường hợp trên.

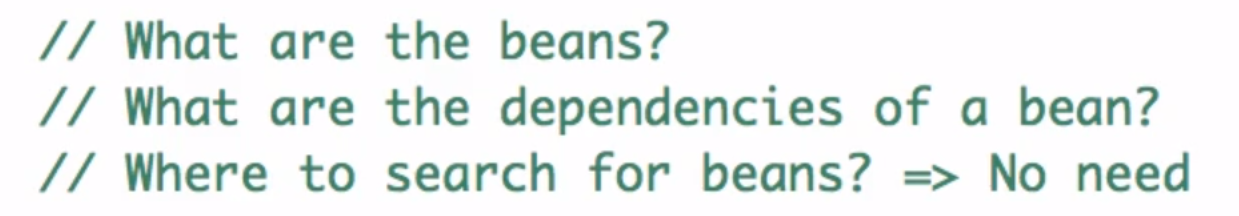
**@Around(“execution(\* PACKAGE.\*.\*(..)”)**: Dùng để đo thời gian thực thi của method.



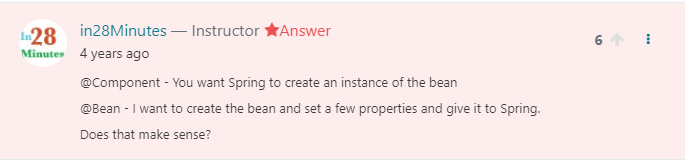
joinPoint.proceed() : Bắt đầu thực thi method.

Core logic của Spring là **Dependency Injection**.

**Beans**: là các instance được **new** ra và được quản lý bởi Spring.

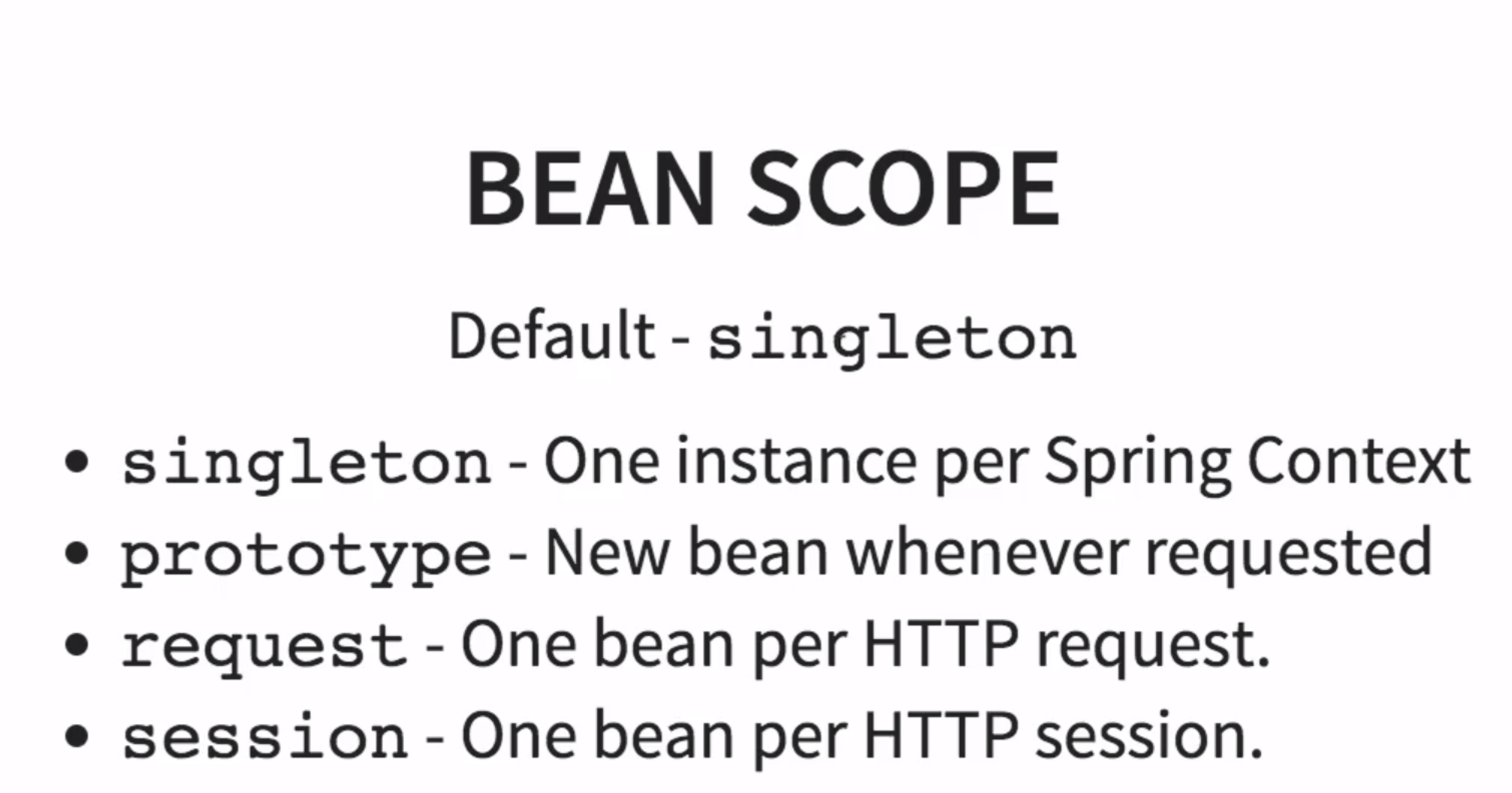


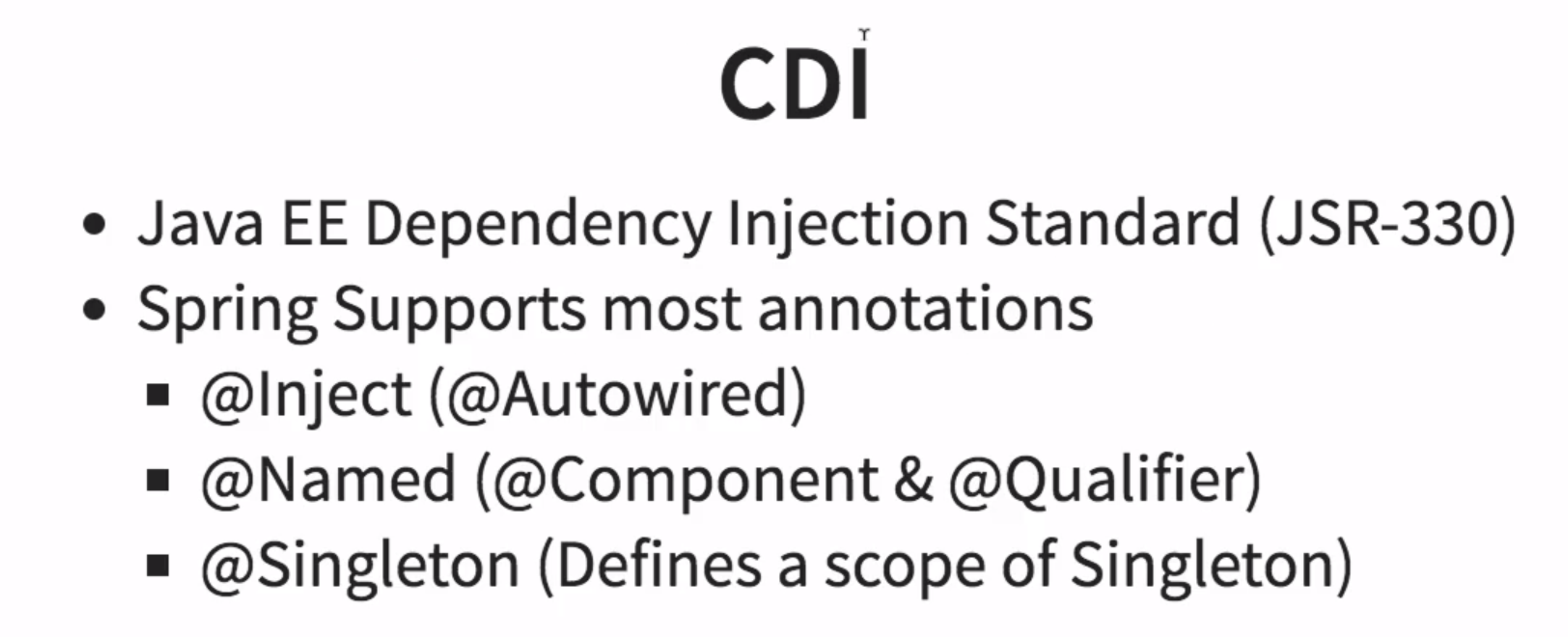
Các bước tạo bean và dependencies.



Dependencies được Auto wire thông qua Constructor hoặc Setter.

Nếu Dependency là Class thì Spring sẽ scan instance của class đó

Nếu Dependency là Interface thì Spring sẽ scan implementation của interface.



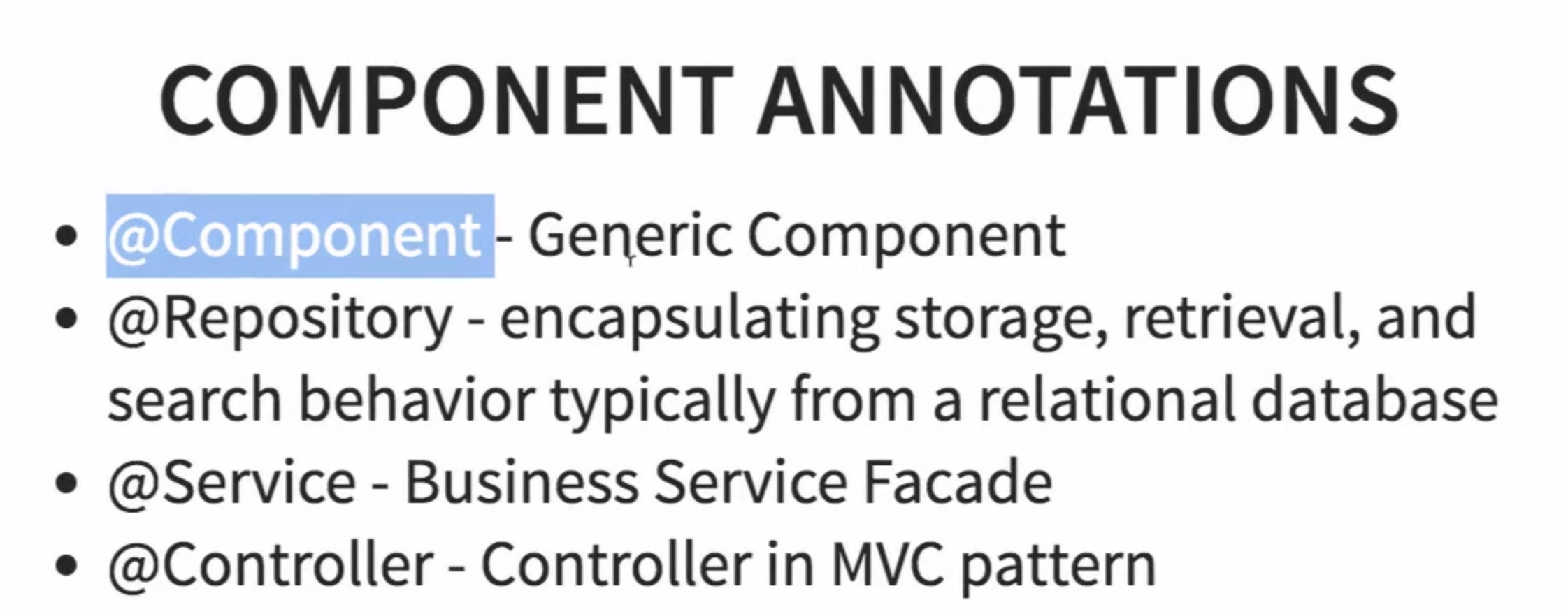
**IOC Container:** phụ trách tạo Bean, và wire các Dependencies.

**Application Context** và **Bean Factory** là 2 implementiation của **IOC Container.**

**Bean Factory**: Đơn giản chỉ tạo Bean và wire các Depends ở mức basic. Dùng cho bộ nhớ thấp.

**Application Context**: Có nhiều feature hơn Bean Factory.

* AOP
* I18n capabilities.
* WebApplicationContext cho web app.



Giúp phân loại Component.

**JDBC**: Phải dùng JDBC thông thường để kết nối đến DB lấy data.

**Spring-JDBC**: Spring cung cấp class JDBCTemplate để lấy connection với DB lấy data.

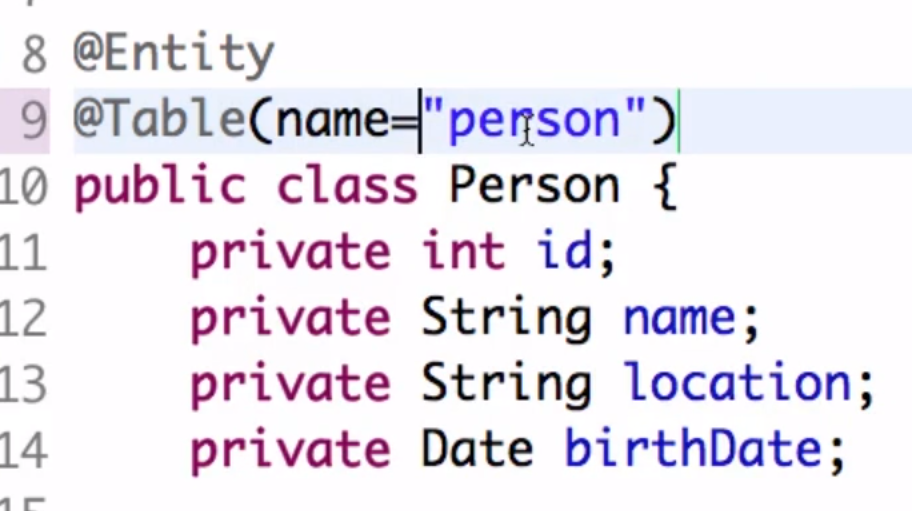
Spring JDBC cung cấp auto mapper có thể sử dụng khi tên các cột match với props của class là **BeanPropertyRowMapper**.

Nếu ta ko declare **Default Constructor** cho Class thì Java sẽ auto cung cấp **Default Constructor** với trường hợp Class đó ko có Constructor nào. Nếu Class có Constructor Param thì Java sẽ KO cung cấp **D.C**.

**JPA**

**@Entity**: Giúp Class map với table có tên trùng với tên Class.

**@Table**: Giúp config tên table để map Class với table bên DB. (Dùng trong trường hợp tên Class ko trùng với tên table).



**@Id**: Giúp chỉ định field nào là **Primary Key**.

**@GeneratedValue**: Giúp tự sinh Id theo thứ tự và map qua field ở Class.

**@Column**: Giúp map tên field ở Class với tên column ở DB.