

У интерфейса ApplicationContext есть большое количество реализаций:

\* ClassPathXmlApplicationContext;

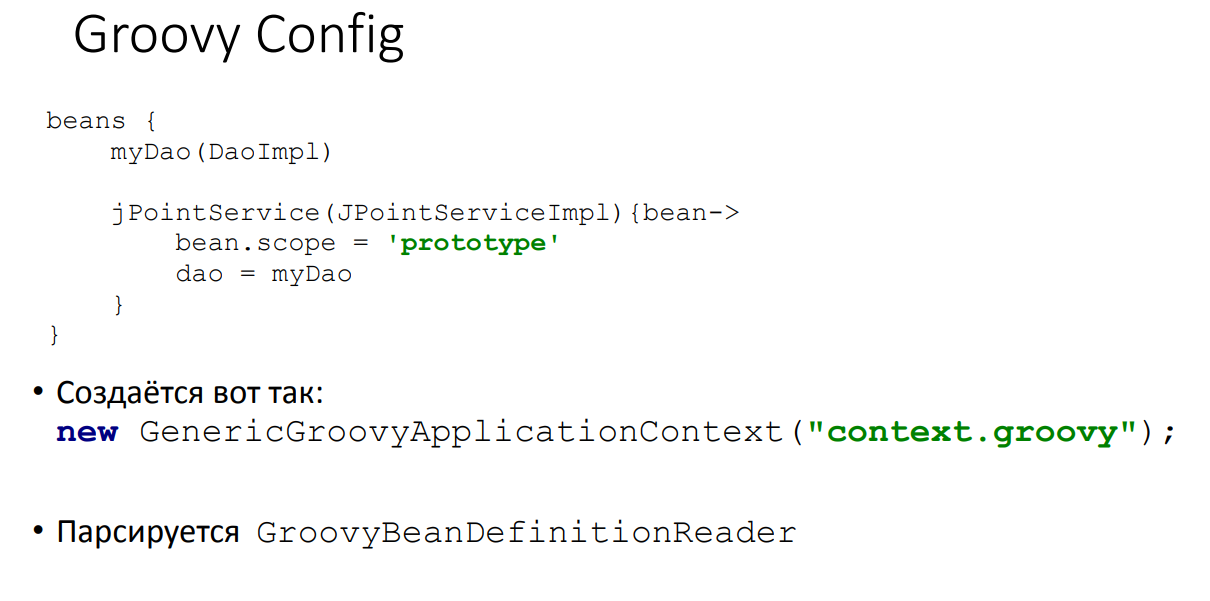
\* FileSystemXmlApplicationContext;

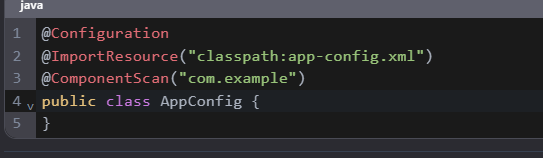
\* GenericGroovyApplicationContext;

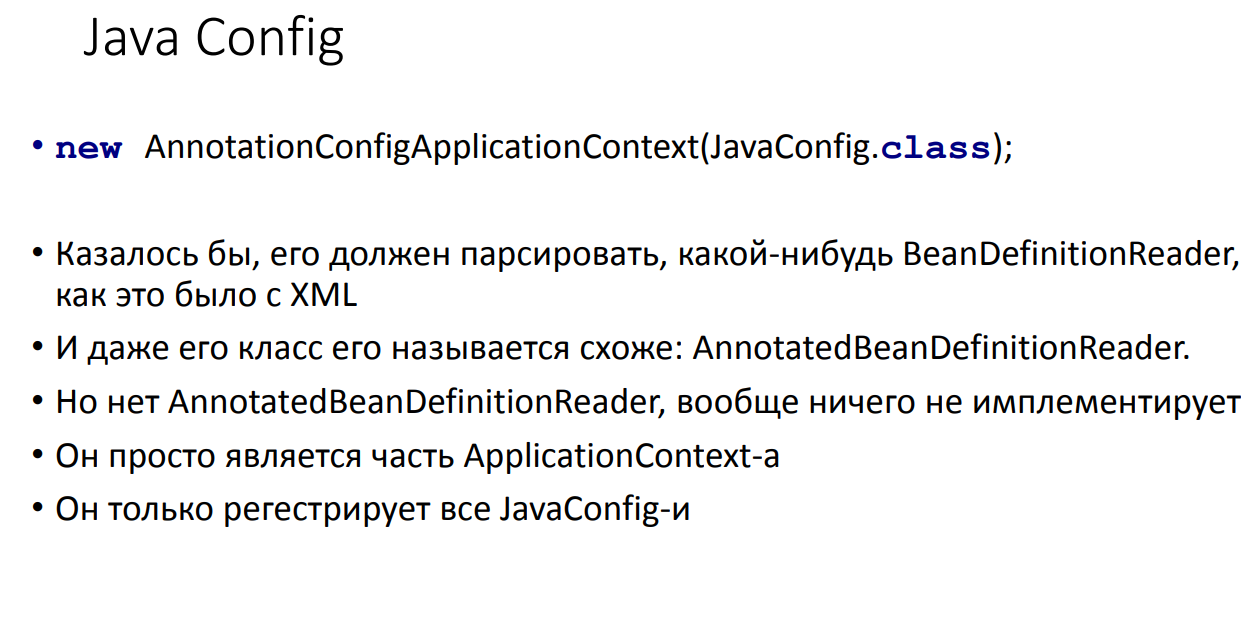
\* AnnotationConfigApplicationContext;

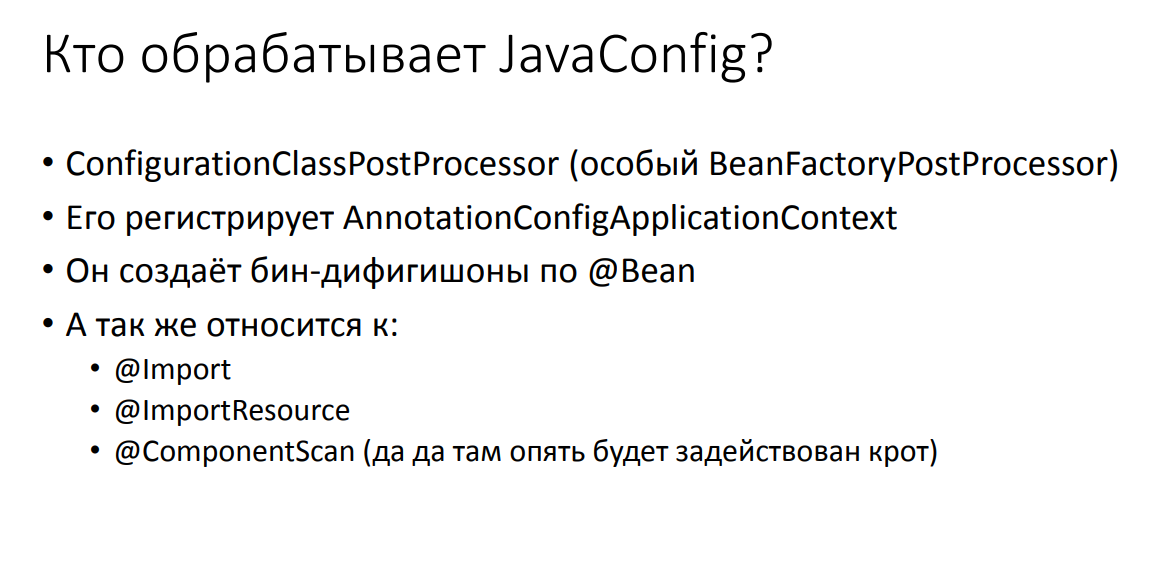
\* XmlWebApplicationContext

\* и др









**LOOKUP, INJECTION**

@ComponentScan: активирует сканирование аннотации @Component для пакета, в котором находится приложение

Сейчас я продемонстрирую пример Dependency Lookup — когда объект сам "достаёт" зависимость из контейнера при инициадизации. Мы в ручную говорим классу. «Возьми пожалуйста нужный нам сервис»

Для активации этих трех функций можно использовать единую аннотацию @SpringBootApplication:

@EnableAutoConfiguration: активирует механизм автоконфигурации Spring Boot

@ComponentScan: активирует сканирование аннотации @Component для пакета, в котором находится приложение

@SpringBootConfiguration: позволяет регистрировать дополнительные бины в контексте или импортировать дополнительные конфигурационные классы. Альтернатива стандартной аннотации @Configuration из Spring, которая помогает обнаруживать конфигурации в ваших интеграционных тестах.

В данном примере мы создаем класс обертку который нам напрямую дает доступ к контексту

И Dependency Injection – когда мы спрингу доверяем внедрение. Это проще и удобнее.

В enterprise разработке Dependency Lookup считается плахой практикой, так как мы лазим в контекст напрямую. Тем самым нарушая данный контракт спринга по автоматическому внедрению. Внутренние механизмы автовнедрения удобны и осуществляются 3 способами. Тоесть разработчики спринга создали гибкие механизмы для использования.