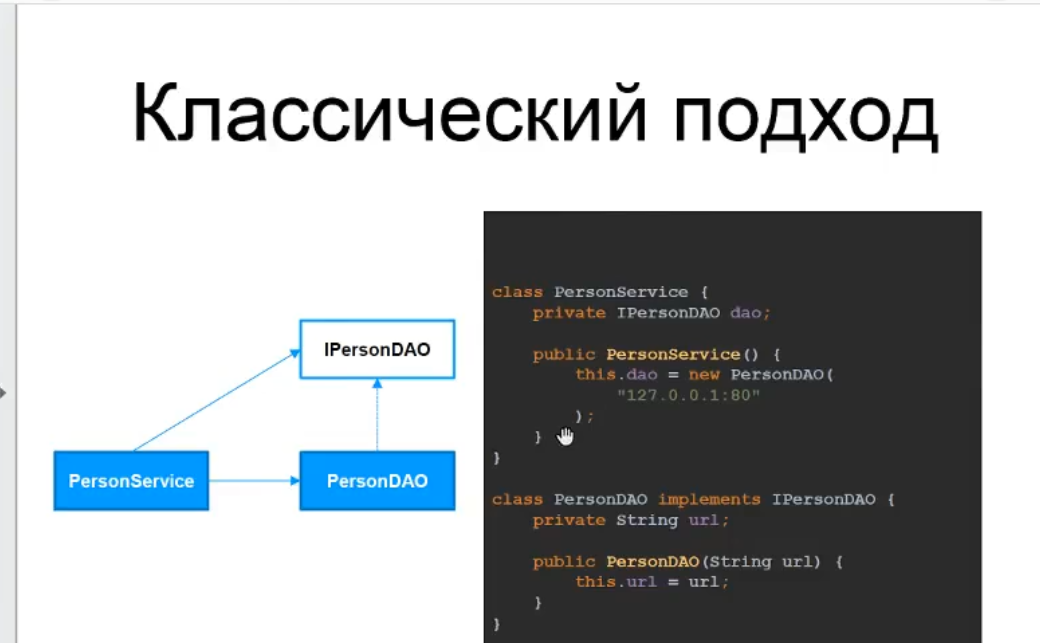
Spring Otus course

Лекция №1

Рассмотрим классический подход:

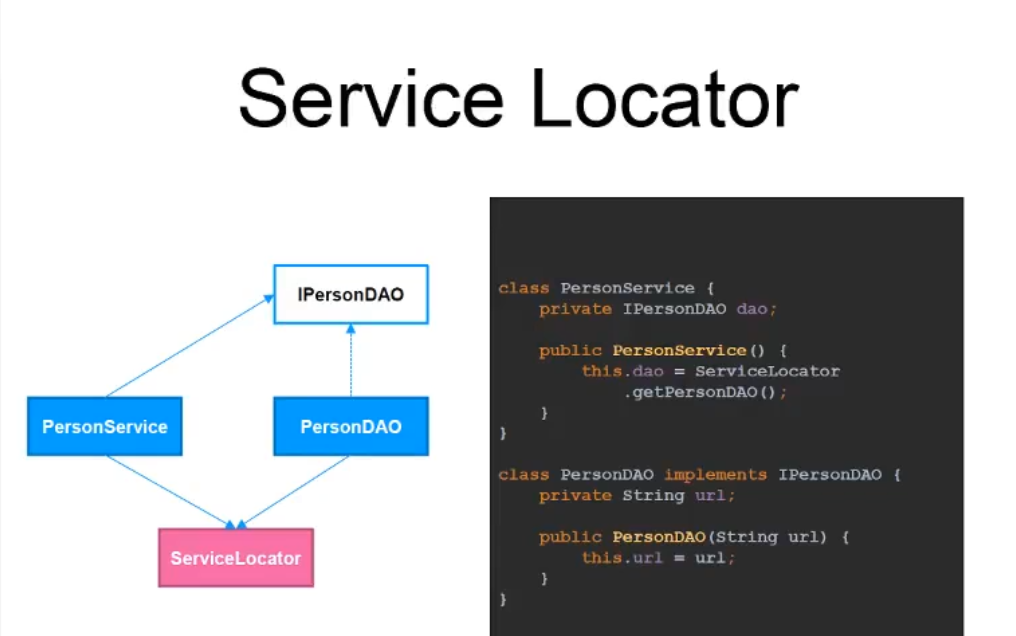


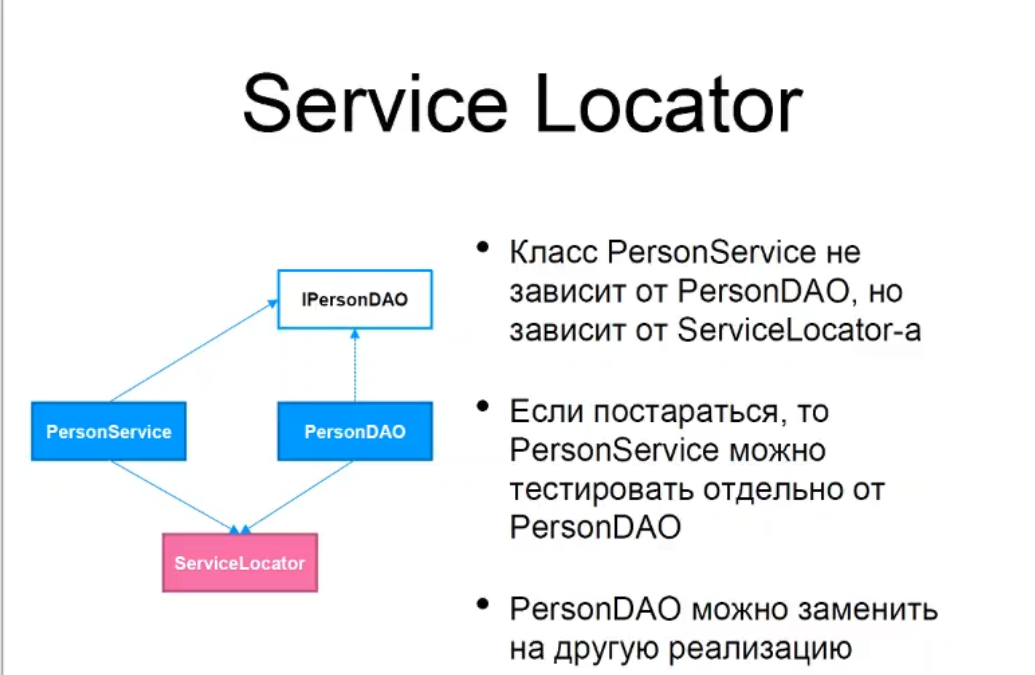


Проблемы классического подхода описаны выше.

Service Locator

Рассмотрим паттерн Service Locator:

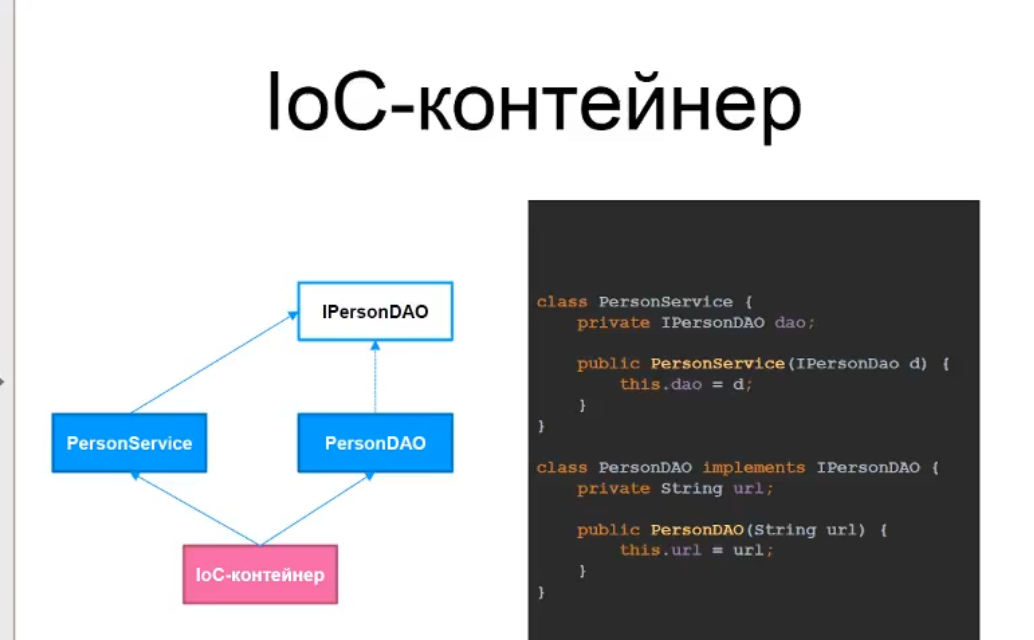




Service Locator позволяет тестировать классы отдельно друг от друга, а так же заменять реализацию PersonDAO на другую, но теперь оба класса не зависят друг от друга, а зависят от ServiceLocator класса

IoС-контейнер

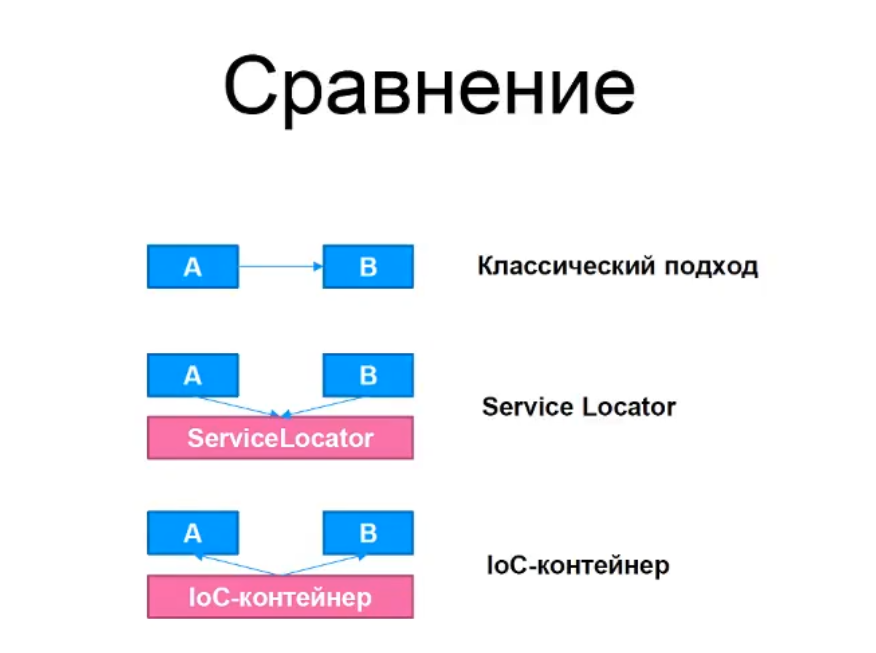
Рассмотрим современный подход для внедрения зависимостей:



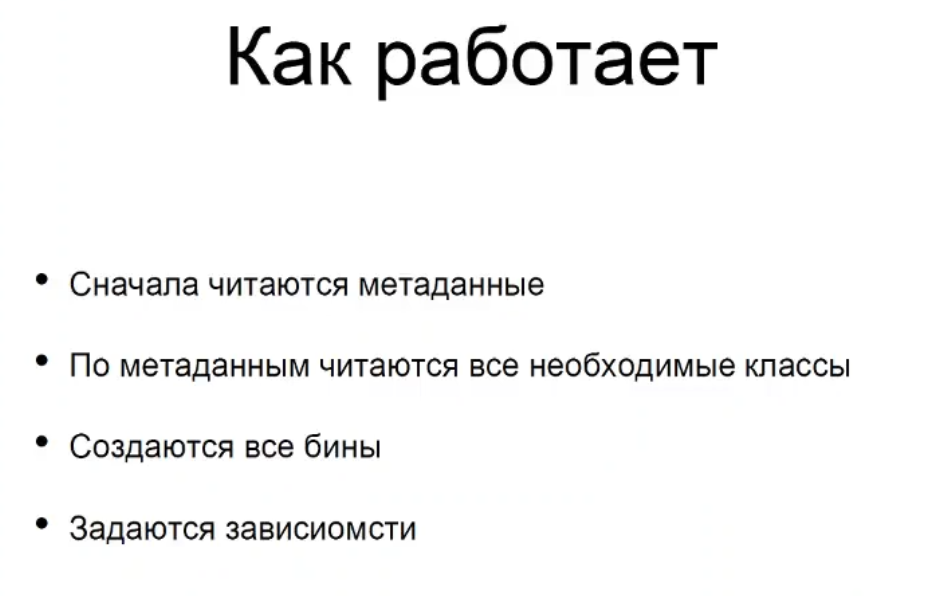


Теперь классы независимы друг от друга PersonService и PersonDAO не зависят друг от друга при создании, их теперь создаёт Spring контейнер, и теперь их удобно тестировать и заменять реализации PersonDAO на другие.

Сравнение зависимостей







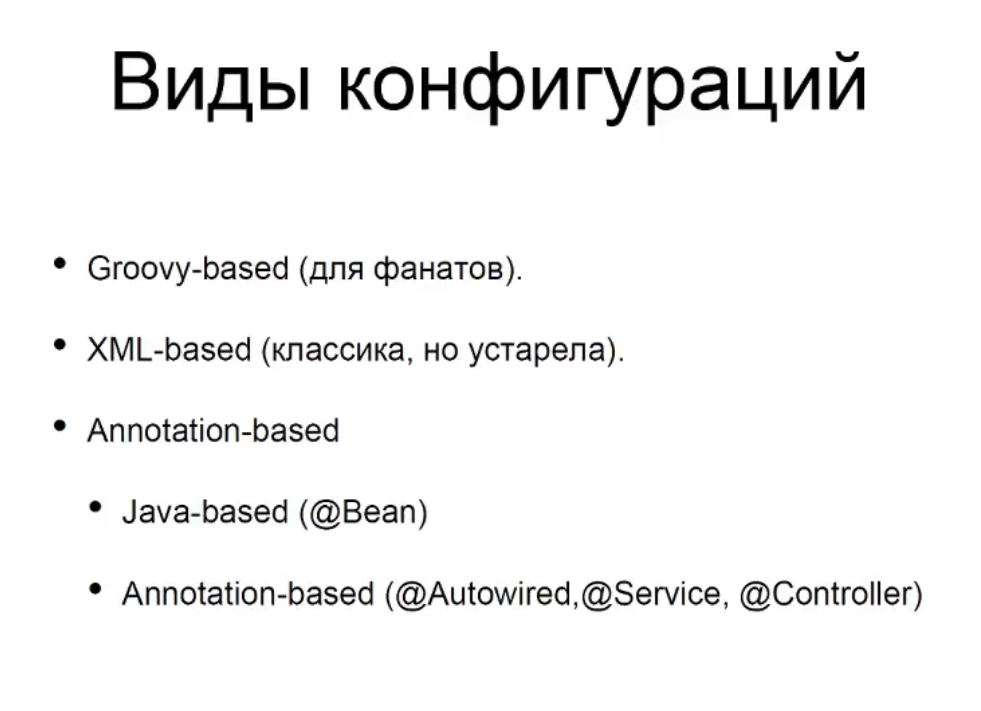
В данном случае метаданные это файлы конфигурации.



Внедрять зависимости лучше через конструктора, так как при создании через property, после создания объекта проходит некоторое время перед заполнением зависимостей через property, что может привести в большом приложении у ошибок NullPointerException, если поля данного объекта будут вызваны до создания. Но если в коде есть циклические зависимости, то можно применить внедрение зависимостей через property, но лучше от такого избавляться.

Java–based Configuration

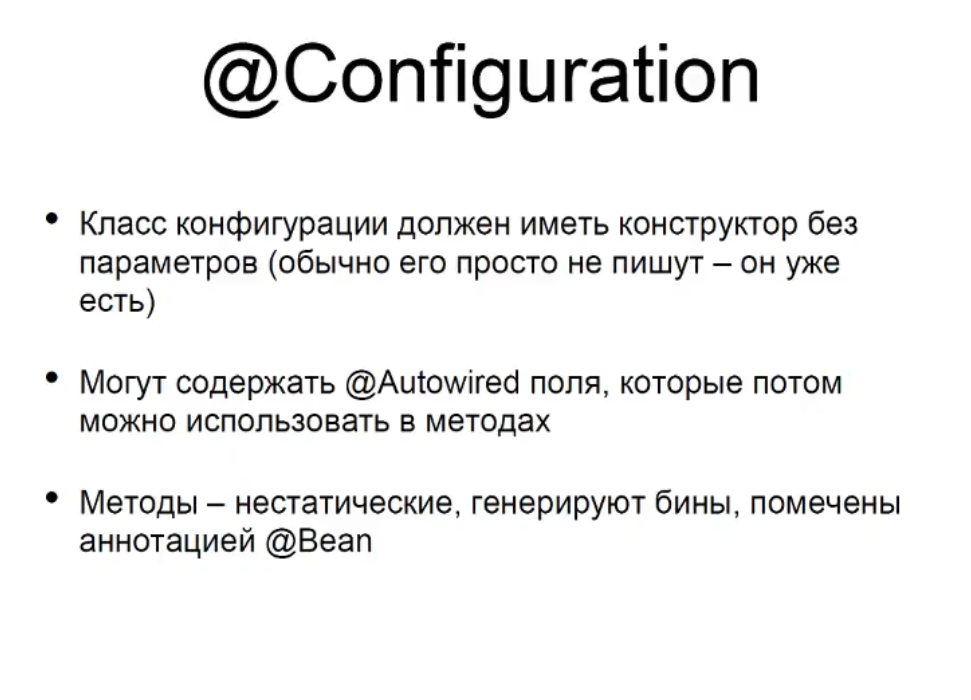
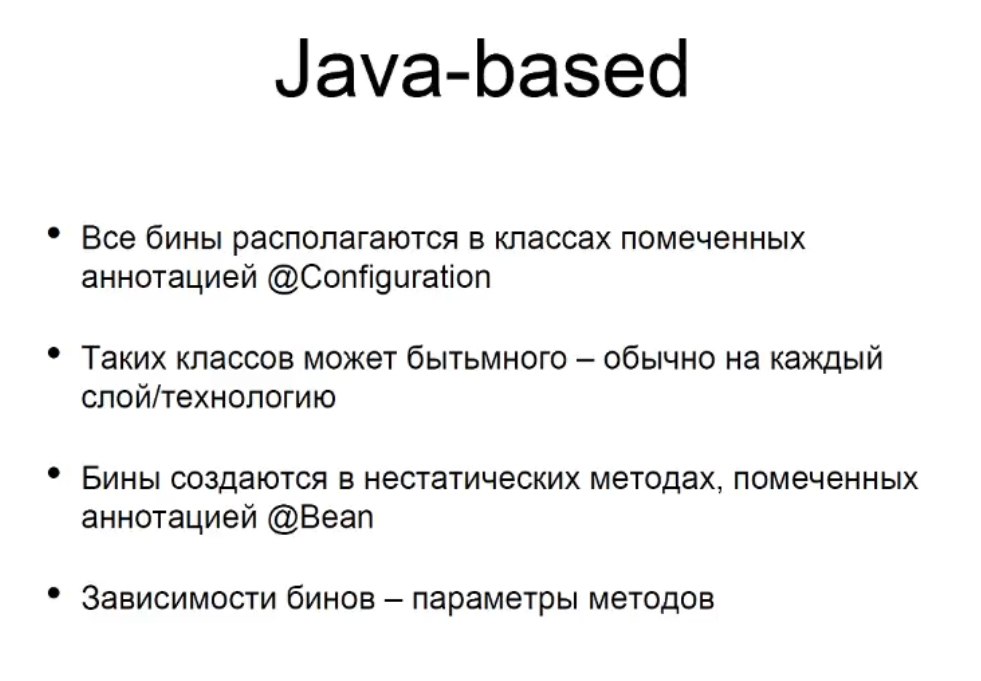
Виды конфигураций иногда ещё называют виды контекстов.





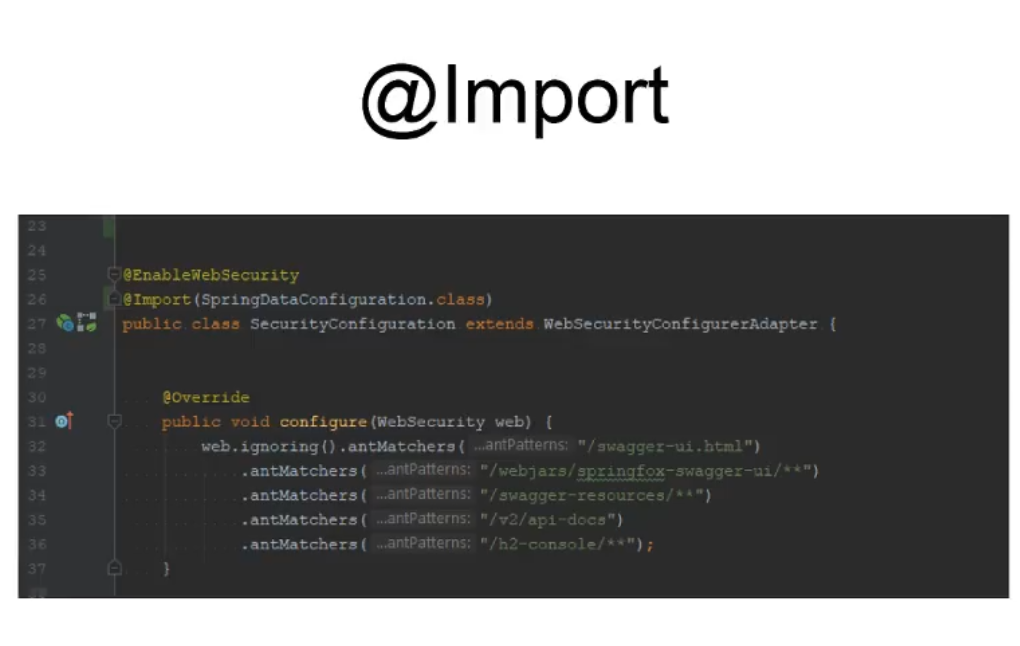


Контекст является динамически заполняемым, т.е есть методы для переопределения конфигурационных файлов как на рисунке выше.





Аннотации могут наследоваться от других аннотаций, а классы конфигураций могут наследоваться от других классов конфигураций.



Импорт нужен при добавлении бинов, в другую конфигурацию, в нашем случае @EnableWebSecurity уже наследуется от Configuration, а @Import, импортирует ещё один класс конфигураций.

Можно было бы наследовать классы через extends, но иерархия конфигурационных файлов по best practice должна быть линейной.

