# 1. SPRING CORE

## 1.1 Spring architecture

### Почему Spring?

**Легковесный фреймворк**. Состоит из набора jar-файлов небольшого размера. Например, если не надо работать с базой, то можно просто не подключать spring-jdbc. Во втором смысле легковесный потому, что у наших классов слабая связность со Spring.

**Spring – это контейнер**. Он управляет жизненным циклом объектов, которые он создает, и которые в нем живут. Не надо писать new, dispose, создавать зависимые объекты. То есть, они все в контейнере будут жить, крутиться, и там же и вариться.

**Spring – это фреймворк**. Он предоставляет кучу готовых велосипедов, которые очень хорошо ездят. В нем есть много утилитарных классов, упрощающих работу с базой данных, с почтой, веб-сервисами, и так далее.

**Dependency Injection (Inversion of Control)**. В Spring реализован принцип инверсии контроля, когда объекты не создают свои зависимости, а получают их.

**AOP (аспектно-ориентированное проиграммированое)**. Спринг сам реализует поддержку АОП, а многие его утилитарные классы и функции работают с помощью этой технологии.

### Основные модули Spring.

**Core container**, состоит из ключевых модулей.

Основные 2 модуля – Beans и Core, которые управляют бинами или реализуют внедрение зависимостей. Context – управляет контекстом, где хранятся бины, обеспечивает доступ к ним. Expression – специальный язык выражений, который может использоваться для поиска и модификаций графа бинов во время выполнения программы.

**AOP** – модуль поддержки AOP. Сюда входит внутренний встроенный модуль AOP, который позволяет вам использовать собственно АОП. Плюс есть модуль Aspects, обеспечивает поддержку библиотеки AspectJ.

**Instrumentation** – необходим, когда вы используете Спринг в каком-то application server (Tomcat, JBoss). Обеспечивает загрузку классов в контейнер и позволяет ему управлять Спринг контекстом и бинами. Содержит единственный модуль Instrument.

**Data access and integration** – модуль доступа к данным и БД в частности. Это и ORM, и JDBC, и JMS, и Transaction (поддержка транзакций), и других интеграционных вещей.

**Web & Remoting** – позволяют создавать компоненты для веб-приложений, и непосредственно веб-приложения. Есть и поддержка базовых вещей – например, реализация загрузки файлов на сервер. (Web, Servlet, Struts). Есть реализация паттерна MVC для создания веб-приложения. И компоненты для создания веб-сервисов, обеспечения безопасности и интеграции с другими веб-технологиями.

**Test** – упрощает тестирование Spring-приложений. С помощью него можно поднять Spring Context, создать бины, вызвать их методы и протестировать.

### Проблемы приложения, сделанного в рамках ветки SpringArchitecture

1. Внесение изменений проблематично. Данные внутри кода (захотим другого клиента, например, и придется ради этого перекомпилировать код). Решить проблему несложно – вынести в .property файлы, например.

2. Невозможность масштабирования. Жестко прописанный логгер, поменять на лету логику его работы без перекомпиляции приложения нельзя.

3. Сложно тестировать. Юнит-тест для метода App.logEvent() непрямо будет также тестировать ConsoleEventLogger.logEvent().

Можно, конечно, решить все три проблемы в отдельности, но их общая проблема в наличии **сильной связности** в приложении. Когда классы друг друга создают, когда в код зашиты данные, логи, которые обрабатывают эти данные, тоже создают новые классы.

## 1.2 Dependency Injection

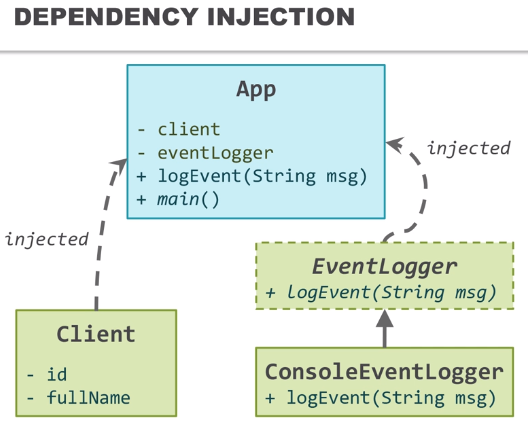
### Как решить описанные выше проблемы?

1. Вынести все статические данные во внешние файлы. Все, что когда-нибудь может измениться, нужно выносить прочь от кода.

2. Разъединение на интерфейсы. Это поможет избежать сильной связности. Если есть сервисные классы, их можно вызвать через интерфейс. Таким образом, конкретный класс не будет зависеть от конкретной реализации, а только от интерфейса сервиса. Перебарщивать с ними тоже не надо – нет смысла создавать интерфейс для класса, только хранящего данные (DAO).

3. Внедрить зависимости (dependency injection). Позволить объектам получать, а не создавать объекты. Сделать так, чтобы не мы создавали класс, а брали созданный кем-то (в нашем случае – Spring Framework). Вообще, существуют и другие фреймворки.

### Dependency Injection



Выглядеть будет примерно так. У нас есть класс App. У него есть все те же самые функции, те же методы, но класс не создает клиента или логгер, а получает их, а к логгеру обращается через интерфейс. Таким образом, App не знает, какой логгер он получит, что позволит нам изменить функциональность в случае необходимости. Такой подход позволяет решить все проблемы, которые были в нашем приложении.

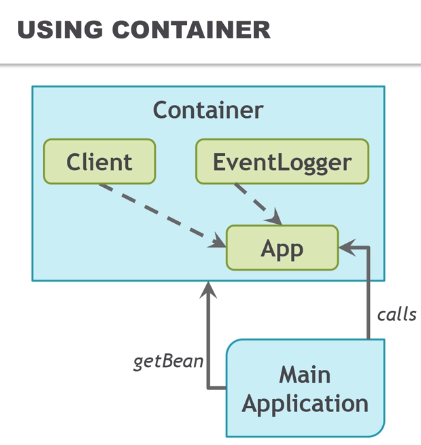
Но кто же нам создаст все эти объекты?

### Spring container

Контейнер будет наши классы создавать, соединять, передавать один в другой, а вы будете обращаться к нему за необходимой информацией. То есть, он создаст объекты классов, которые вы опишете. Данные объекты в Spring называются бинами.

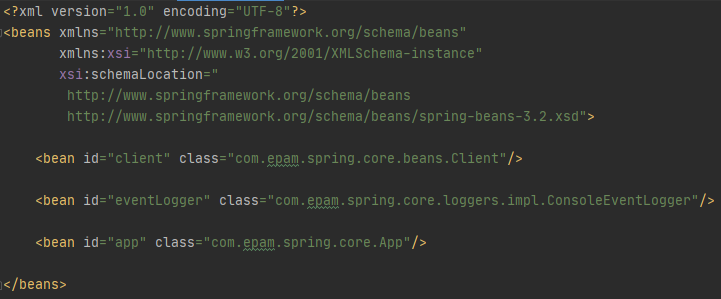
Спринг проинджектит нужные зависимости. То есть, он поймет, что нужно клиента и логгера засунуть в App, возможно, другие объекты между собой соединит, проверит, чтобы не было циклических зависимостей, и все. Объекты живут, зависимость у них есть. Осталось только к нему обратиться.

Наше приложение обратится к контейнеру, попросит у него определенный бин, а дальше будет вызывать его методы. Теперь, если нам нужен какой-то объект, то мы не создаем его, а обращаемся к контейнеру, чтобы его получить, либо делаем навигацию по графу объектов (получаем первый, через него получаем следующий, и так далее). Обращаться к контейнеру нам понадобится только в исходной точке нашей программы. Остальные объекты получат бины через внедрение зависимостей.



Как же Спринг поймет, какие объекты ему создавать? Для этого в Спринг используется xml файл. В нем содержится описание контейнера – тех бинов, которые вы хотите создать. Кроме xml, можно использовать аннотации. Ими вы отмечаете классы, которые станут бинами, а также места, где их надо проинджектить. В принципе, конфигурацию Спринг можно описать и полностью без xml, используя для этого только Java код.

### Spring.xml



Давайте разберем его по частям.

Главным образом вы определяете тег beans, а внутри него уже остальные бины. Также в этом теге указываются, какие xml схемы мы будем использовать. Основные схемы, которые нужно подключить – это перечисленные здесь. Без них ничего работать не будет.

Дальше вы определяете ваши бины. Назначаете им какие-то id, и указываете class, объект которого хотите создать.

Такой файл лучше создавать в каталоге ресурсов (src/main/resources).