# 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.