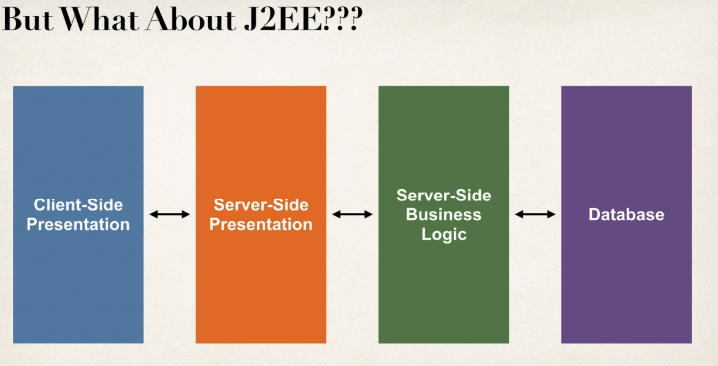
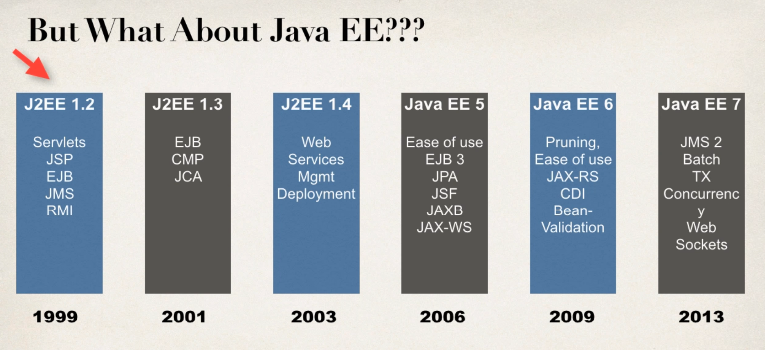
**Spring Framework**

Q. Why Spring?

> Spring is a very popular framework for building Java applications.

> Spring was initially a simpler & lightweight alternative to J2EE i.e. provides a large no. of helper classes… makes things easier.





JSP – Jakarta Server Pages / JavaServer Pages

EJB – Jakarta Enterprise Beans / Enterprise JavaBeans

JMS – Java Message Service

RMI – Remote Method Invocation

CMP – Container Managed Persistence

JCA – Java Connector Architecture

JPA – Java Persistence API

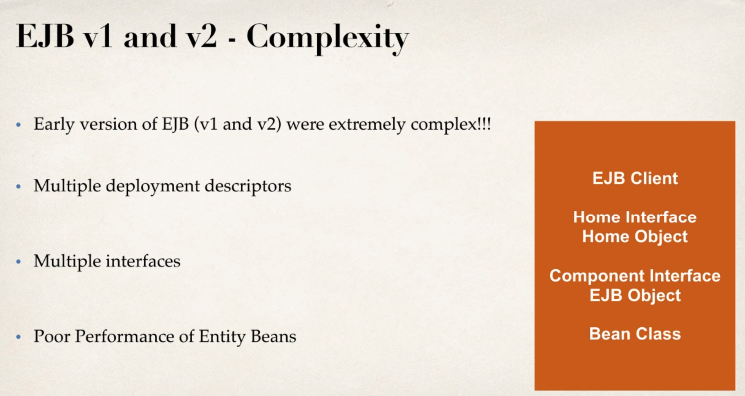
JSF – Java Server Faces

JAXB – Java API for XML Binding

JAX-WS – Java Web Services (SOAP)

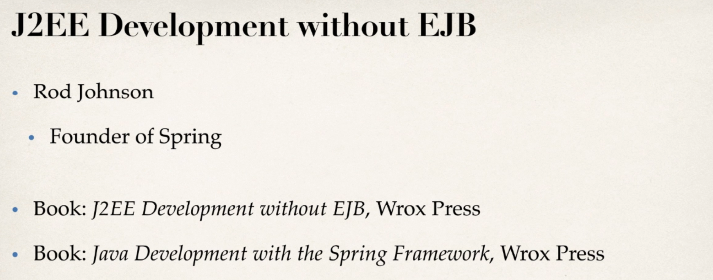
JAX-RS – Java Web Services (REST)

CDI – Context Dependency Injection (IoC)

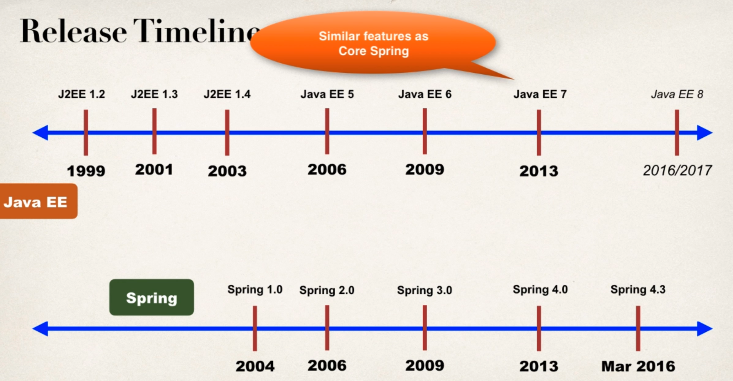


> Entity Beans basically mapping between java class & Database table are just awful slow. Even on one deployment, we actually had to pull our code back out of the production & actually remove the EJB functionality because it slowed everything down.

That’s a lot of developers started to continue to do J2EE but they actually would do it without Enterprise JavaBeans.



Release Timeline of J2EE & Spring

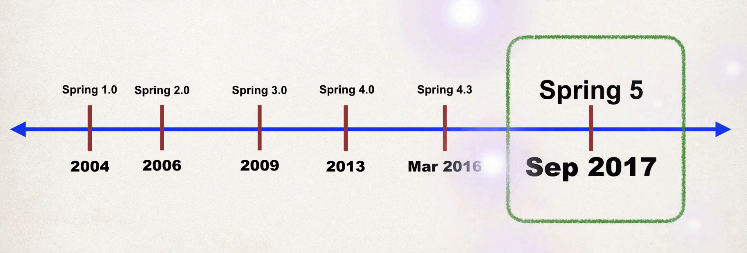


EJB 3.1 in Java EE 6 was much easier to use i.e. in J2EE 6, added CDI Context Dependency Injection IoC.

So from **Java EE 7**, you can do everything as **Spring** can do but the only problem though is that they were just a little bit too late & unfortunately, EJB just kind of has a bad name & also, Spring has huge momentum, huge market share.

> Spring is a lightweight framework, simple, easy to use & a lot of developers like it.

What’s New in Spring 5?



> Updated minimum requirements for Java 8 or higher.

> Deprecated legacy integration for: Tiles, Velocity, Portlet, Guava etc.

> Upgraded Spring MVC to use new versions of Servlet API 4.0

> Added new reactive programming framework: Spring WebFlux

**Spring Framework Overview**

> Spring official Website: - www. spring.io

**> Goals of spring**

a) Lightweight development with Java POJOs (Plain – Old – Java – Objects))

[Make it much simpler to build, as compared to the heavyweight EJBs from the early versions of J2EE]

b) Dependency injection to promote loose coupling

[So instead of hard wiring your objects together, you simply specify the wiring via a configuration file or annotations.]

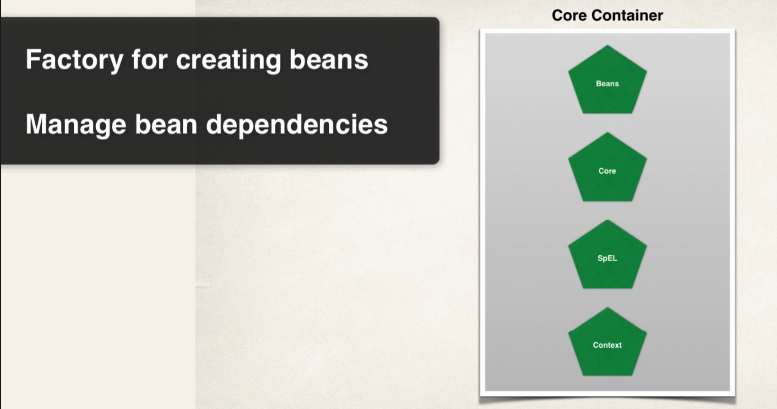
c) Declarative programming with Aspect – Oriented – Programming (AOP)

[Basically allow you to add some application wide services to your given objects]

d) Minimize boilerplate Java code.

[In early days of J2EE, there was a lot of code that you had to write, so the folks at Spring created a collection of helper classes to make it easier.]

**Spring Core**



> So the core container is like heart of Spring framework.

> It basically manages how beans are created.

> It has a bean factory for creating the beans.

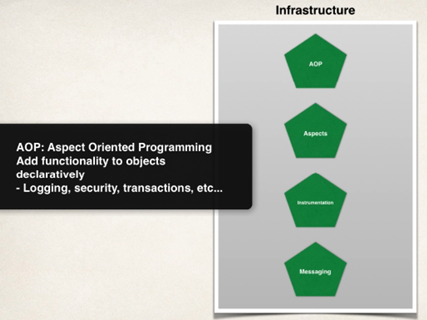
> It basically can read configuration files for setting properties in dependencies.

> Also the Context here is really the spring container that holds the Beans in memory.

> Spring Expression Language (SpEL) is a language we can use within the configuration files to refer to other beans.

**Spring AOP**

**- AOP, Aspects, Instrumentation, Messaging**

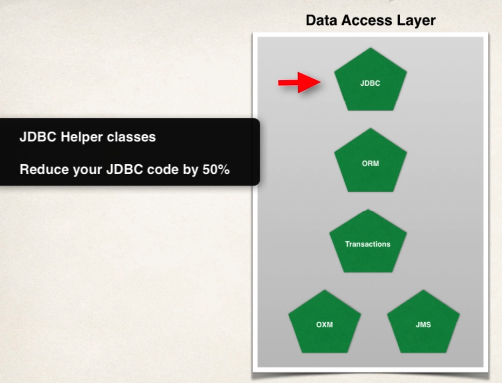


> **Spring AOP** allows you to create application wide services, like logging, security, transactions, instrumentation & then you can apply these services to your objects in a declarative fashion, so no need to modify your code to have support for this. You simply add a configuration in the configuration file or an annotation and that service will be applied to your application.

> **Spring Instrumentation** – Here you can actually make use of class loader implementations to work with different apps server. For e.g. it can be used to create a Java agent, so you can remotely monitor and instrument your application using JMX (Java Management Extension)

As a developer, you would not normally build an agent yourself. You would simply use the agents provided by the Spring team or your app server vendor [but behind the scenes, it’s making use of some really cool technologies like AOP coding, bytecode manipulation etc.], so that’s what you get in the instrumentational model.

**Spring Data Access Layer**



> **Spring Data access layer** is for communicating with the database, either a relational database or a NoSQL database and also making use of a message queue.

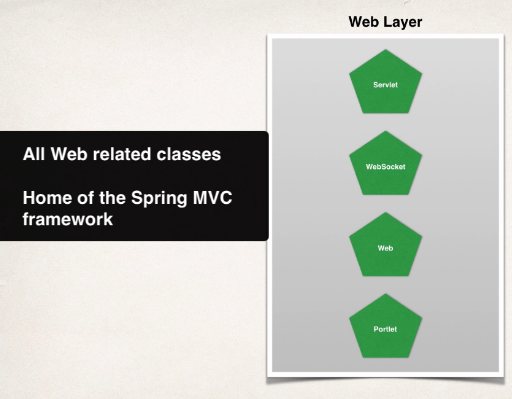
> Basically, Spring provides some helper classes to make it much easier to access a database, using JDBC & by using these Spring JDBC classes, you can actually reduce your source code by over 50%

> **ORM** (Object to Relational Mapping) is probably the most popular section of this module. Basically, it allows you to hook into Hibernate, or hook into JPAs.

> **JMS** (Java Message Service) allows you to send messages to a message queue (Message Broker) in an asynchronous fashion. That’s a core part of JAVA EE. Here, they basically provide helper classes to allow you to make use of the JMS & again, you can reduce your code by over 50% by making use of spring’s JMS integration.

> Spring has support for a **transaction manager** or supporting transactions & you can do this in a very lightweight fashion. So you can make use of transactions on methods, on database calls, & pretty much anything you want. Transactions manager makes heavy use of AOP behind the scenes.

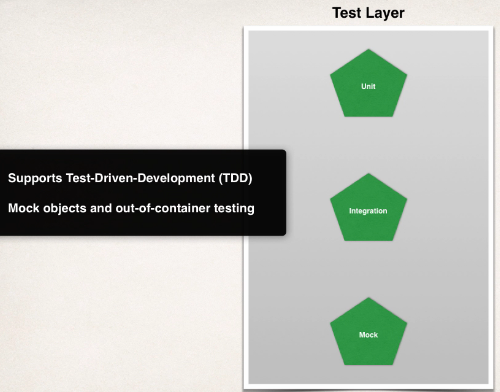
**Spring Web Layer**



> So you can build web applications using the Spring core, & also making use of Spring Controllers & Spring View so you have a full MVC layout here.

> They also have support for Remoting here, so you can actually make use of web remoting, where you can have external clients make calls into the Spring container. [like a way of doing remote procedure calls (RPC) or also doing distributed computing]

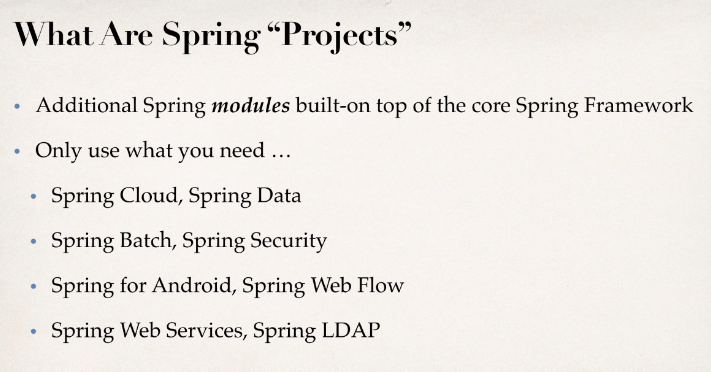
**Spring Test Layer**

****

> Spring has support for test – driven development, so the framework includes mock objects for mocking out servlets, JNDI access & so on and you can do all of this outside of the container.

> You can also make use of integration test by creating an application context & writing up your desired object. So testing is a first class citizen here, when making use of the Spring framework, so there’s a lot of good support for it.

**Spring Projects**



> Spring projects are just additional Spring modules that are built on top of the core framework, so think of them as simply add – ons.

> You only use what you need so they have projects here for Spring cloud & Spring Data so

> Cloud for doing cloud development.

> Data for database integration.

> Spring batch for creating batch processes.

> Spring security for securing your application.

> Spring for Android for Android development

> Spring Web flow for doing web flow over x number of pages.

> Spring Web Services for doing restful & soap web services.

> Spring LDAP for accessing LDAP servers.

> Location to get information on Spring projects - <https://spring.io/projects>

**Spring Environment Setup**

> You must have the Java Development Kit (JDK) installed.

> Spring 5 requires Java 8 or higher.

> Java Application server for web development like Glassfish, JBoss, WebLogic & so on. For simplicity, use Tomcat server.

> Java Integrated Development Environment (IDE) like Eclipse

> Installing Tomcat

Step 1: Go to website - <https://tomcat.apache.org/>

Step 2: Select version & download binary distribution for Core

Step 3: Install

> Installing Eclipse

Step 1: Go to website - <https://www.eclipse.org/>

Step 2: Download eclipse & install it.

> Connecting Tomcat to Eclipse

> Downloading Spring 5 JAR File & add JAR files to Eclipse project…Build Path [You can also use Maven]

> Spring Repo - <https://repo.spring.io/release/org/springframework/spring/>

> Download latest release dist file & unzip it & import it into the Java project lib

> Then go to project properties & find build path

> Add lib to build path libraries

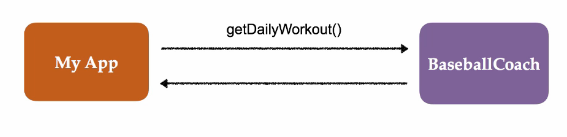
> Once libraries will be built you will see “Referenced Libraries” in package Hierarchy.

**Spring Inversion of Control**

> Inversion of Control (IoC) is simply the design process of externalizing the construction & management of objects.

> It basically says that your application’s going to outsource the creation & management of the objects & that outsourcing will be handled by an object factory.

> **Coding Scenario**



a) App should be configurable

b) Easily change the coach for another sport

