**What is Spring?**

Spring is a very popular framework for building enterprise Java applications with. The goal of the Spring framework is the following:

* Lightweight development with Java POJOs (**P**lain-**O**ld-**J**ava-**O**bjects)
  + Normal Java objects like MyClass myClass = new MyClass();
* Dependency injection to promote loose coupling
* Declarative programming with AOP (**A**spect **O**riented **P**rogramming)
  + Adds function for logging, security, transactions et.c without cluttering your normal code.

So for example if you want to check how long it takes to run a method instead of declaring a timer at the start and then printing the time before exiting you can define through AOP that you want to check how long it takes to run. Then at compilation time or class loading time the code will be injected so it will look like how you originally had wrote it.

* Minimize boilerplate Java code

**Spring History**

Spring started as a lightweight and simpler version of J2EE (Java2 EE). The issue with J2EE at the time was that EJB (**E**nterprise **J**ava **B**eans) was very complex to work with, it required multiple deployment descriptors, interfaces and on top of it the beans had very poor performance.

The performance of EJB was so poor that developers ended up having to pull code back from production because it sank the system.

Implementing one bean also meant a lot of code, so something simple as creating an animal class that can make a sound would take up 3-4 java classes if it was implemented with J2EE. This caused a lot of bugs in codes since there were so much to write to get things done.

Developers eventually found a way around working with EJB so they cut it out entirely and this was what became the groundwork for the Spring framework.

**Why Spring?**

Spring remains among the top skills that is requested by Java employers and recruiters. There is JavaEE but because of the EJB debacle in the late 90s it have left a sour taste in people’s mouths Another huge plus with Spring is that it provides so many helper classes that it have essentially became a platform in its own right. Spring has support for:

* Testing
* Hibernate
* JDBC (reduces your JDBC code by roughly 50%)
* Et.c

**How does Spring core work (on the surface)?**

Spring core is built on two core design principles, IoC and DI.

* IoC
  + Inversion of Control, design principle where your code receives it flow from a generic framework. This promotes loose coupling by outsourcing the creation of objects to a framework.
* DI
  + Inject classes and fields to other classes.

**Demos**

Three short demos to show how to turn a normal Java application into a Spring application. Demo0 is pure Java while Demo1 and Demo2 make use of the two Spring concepts mentioned above (IoC, DI)

Demo0

*Assignment: Create a program that can retrieve different coaches*

Demo1 (IoC)

*Assignment: With the help of Spring we want to make Demo0 easy to configure which coach we should retrieve*

Demo2 (IoC&DI)

*Assignment: Modify Demo1 to inject another class into the coaches*

**Spring Boot vs Spring Core**

Spring boot is essentially Spring Core but on steroids, it provides a suite, pre-configured, pre-sugared set of frameworks/technologies to reduce boiler plate configuration. Providing you with the shortest way to have a Spring web application up and running with smallest line of code/configuration out of the box. Below is an example that make use of Spring Boot and Tomcat, it’s a simple hello world web page:

@Controller // Tell Spring that this is our entry point

@EnableAutoConfiguration

public class SampleController {

@RequestMapping("/") // Catch any requests that comes in, regex equivalent to ‘\*’

@ResponseBody // For any request we catch answer with the method below

String home() {

return "Hello World!";

}

public static void main(String[] args) throws Exception {

SpringApplication.run(SampleController.class, args);

}

}

**Closing notes**

As a final note the question between which frameworks to choose is that you should try both of them to see which one fits you best. Since Spring started as a JavaEE offshoot and then JavaEE adapted functions that existed in Spring they are in present time more or less the same. So if you know how to use Spring it’s very easy to learn how to use JavaEE and vice-versa.