selsoft

build better, deliver faster

Enterprise Application Development with Spring

Chapter 1: Introduction to Spring

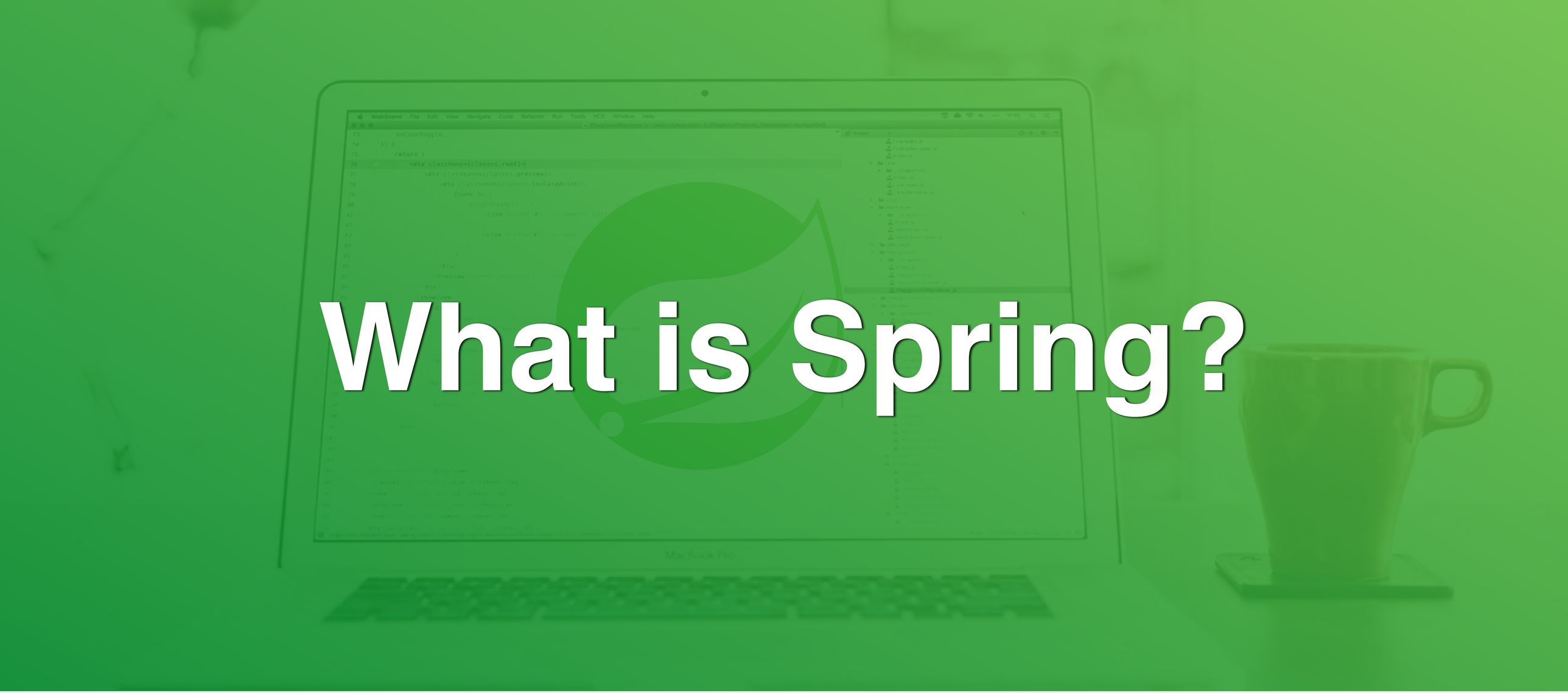
Eğitmen:

Akın Kaldıroğlu

Çevik Yazılım Geliştirme ve Java Uzmanı

Topics

- **What is Spring?**
  - History of Spring
  - Usage Scenarios
  - Spring Documentation



What is Spring?

selsoft

build better, deliver faster

What is Spring?

- **Spring** is an application development framework for Java.
- **Spring** makes the use of Java easier thus making developer more productive.
- **Spring** became the most used application development framework in Java world.
- **Spring** is not only for Java EE but also Java SE and mobile Java.

What is Spring?

- **Spring** defines itself as
  - **Spring** makes Java simple
  - **Spring** makes Java productive
  - **Spring** makes Java reactive
  - **Spring** makes Java cloud-ready

Why Spring?

- https://spring.io/why-spring says:
  - **Spring** makes programming Java quicker, easier, and safer for everybody.
  - **Spring**’s focus on speed, simplicity, and productivity has made it the world's most popular Java application framework.

What Do We Mean by Spring?

- Back in the first years of **Spring**, when we said **Spring** we meant mainly DI via **Spring** and some other components such as Spring MVC.
- But now, there are many projects such as Spring Framework, Spring Roo, Spring Cloud, Spring HATEOAS, etc. in **Spring**.
- So the Spring Framework is a foundation project and it provides basic mechanism such as IoC, DI, AOP and all other projects are built upon it.

History of Spring

Java Enterprise Edition - I

- **Java Enterprise Edition**, **Java EE**, is a set of standards for enterprise software:
  - Web, web services, persistence, transaction, validation, integration, security, distributed computing etc.
- Java EE is based on Java SE and it has many implementations along with reference implementation called **Glassfish**.

- JSR 366 – Java EE 8 Platform
- JSR 365 – Contexts and Dependency Injection (CDI) 2.0
- JSR 367 – The Java API for JSON Binding (JSON-B) 1.0
- JSR 369 – Java Servlet 4.0
- JSR 370 – Java API for RESTful Web Services (JAX-RS) 2.1
- JSR 372 – JavaServer Faces (JSF) 2.3
- JSR 374 – Java API for JSON Processing (JSON-P)1.1
- JSR 375 – Java EE Security API 1.0
- JSR 380 – Bean Validation 2.0
- JSR 250 – Common Annotations 1.3
- JSR 338 – Java Persistence 2.2
- JSR 356 – Java API for WebSocket 1.1
- JSR 919 – JavaMail 1.6

Java Enterprise Edition - II

- J2EE launched at the end of 1999 and last version Java EE 8 in 2017.
- Java EE moved to Eclipse Foundation in 2017 and was named **Jakarta EE** (https://jakarta.ee/)
- Jakarta EE 9 is planned to be available in November of 2020.

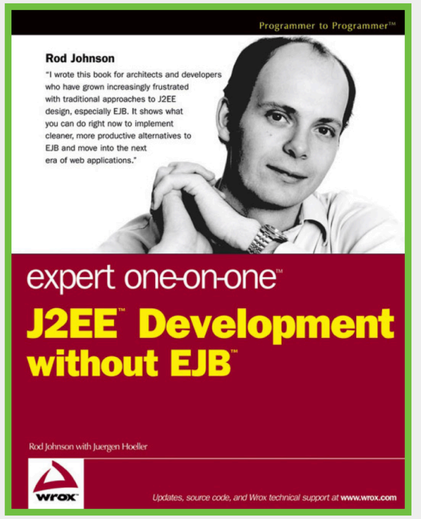
Enterprise JavaBeans (EJBs)

- EJBs (Enterprise JavaBeans) are the main vehicle of J2EE.
  - EJBs had two types: session beans and entity beans.
- EJBs were too complicated to develop and need an EJB container namely application server to live in.
- They provided many advanced approaches and techniques such as context and dependency injection, declarative transaction and security management, AOP, etc. for that time.

How Did Spring Start? - I

- **Spring** has been created by Rod Johnson in 2003 as a response to the complexity of Java EE standard and its high-cost containers.

- He explained his approach to J2EE development in his first book **Expert One-on-One J2EE Design and Development** and then in second book **Expert One-on-One J2EE Development without EJB** where he introduced **Spring** framework.



Lighter-Weight, More Flexible

- He says in **Expert One-on-One J2EE Development without EJB**:

  The traditional approach to J2EE architecture has often produced disappointing results: applications that are more complex than their business requirements warrant, show disappointing performance, are hard to test, and cost too much to develop and maintain.

  It doesn’t need to be so hard. There is a better way for most applications. In this book, we’ll describe a simpler, yet powerful architectural approach, building on experience with J2EE and newer technologies such as Inversion of Control and AOP. Replacing EJB with lighter-weight, more flexible, infrastructure typically produces significant benefits. We and many others have used this approach in many production

Lightweight J2EE Architecture

- Main properties of the light weight J2EE architecture he offered in **Expert One-on-One J2EE Development without EJB** are:
  - Web tier is provided by a MVC framework
  - Business objects will be POJO (Plain Old Java Objects)
  - Entity beans are not an option, so data access will be through an O/R mapping solution or JDBC

Advantages of Avoiding ASs

- In his book he talked about the real advantages in avoiding an application server:
  - Lower license costs, in the case of commercial products. Industry research groups have reported significant overspending on application server licenses for organizations that didn’t need their capabilities.
  - Quicker server startup, saving development time.
  - Simpler administration and reduced learning curve for developers.

How Did Spring Start? - I

- So **Spring** started as an endeavour to provide what Java EE provided in a simpler programming model and without the need for costly application servers.
- That's why two of the points **Spring** emphasized in its early days is the use of POJO and its own lighweight container.
- 1st version was released in 2003, 2nd in 2006, 3rd in 2009, 4th in 2013 and 5th in 2017.
  - Spring Boot was introduced in 2014

Java EE vs. Spring - I

- We kept saying that if a standard soution is offered by Java itself, prefer it to any third party such as **Spring**.
  - That’s because Java SE and EE are standards and **Spring** is a third party framework.
- But as of now nobody is sure about the feature of Java EE but **Spring** is still there, keeps growing and paving the way in areas such as cloud and reactive programming.

Java EE vs. Spring - II

- IMHO Java EE will still specify the rules and **Spring** will continue to provide enhancements and add ons on the top of Java EE.
- No question about Java SE as the language.
- And **Spring** is enlarging to Kotlin, Android and even to Python.

Usage Scenarios

Usage Scenarios of Spring

- There can be different categories of scenarios regarding how **Spring** are used in Java applications:
  - **Spring enhances Java**
  - **Spring complements Java**
  - **Spring competes Java**
- This categorization is made only to make understanding how **Spring** sits in Java ecosystem easy and includes lots of gray areas.

Spring Enhances Java

- Most of the time **Spring** enhances Java's components to make their usage easier mostly by providing boilerplate code either by configuration or by its API.
  - **Spring**'s support for JDBC and JPA
  - Most of **Spring** integrations such as JMS
- Sometimes what **Spring** enhances is not Java's standard component but a very well-known and used framework such as JUnit.
- This saves us lots of lines of boilerplate code.

Spring Complements Java - I

- Very frequently **Spring** complements Java when Java does not offer a solution or what Java offers is just an advice, not a standard component.
- This scenario is more of fill-the-gaps because using Java only requires lots of low level architectural decisions and coding.
  - Spring MVC is a totally glueing solution for servlets and JSPs in order to use them in a standard architecture.
  - Spring Boot, Security, Cloud, Integration and many more projects are of this kind, complementary solutions.

Spring Complements Java - II

- Reason why Java leaves such gaps might be multifold:
  - Sun or Oracle does not choose to implement such a solution because they do not see or believe the gap or even they believe it they leave it to the community or they did not have resource to implement such a solution.
  - The role of Sun and Oracle is mostly for specifying the standart with the help of community.
  - In these two categories **Spring** obeys the standards of Java.

Spring Competes Java

- Sometimes **Spring** competes Java by providing alternative solutions to what already exists in Java as a standard.
  - DI, Restful WS are all part of Java standards but **Spring** provides alternatives.
- In this category **Spring** does not obey the standards of Java.

Why Spring Competes Java?

- Why does **Spring** provide alternatives?
- There are some different reasons:
  - For the case of DI, declarative transactions, etc. the reason is to make the solution Java EE provides accessible to anybody in a simpler model and without any cost.
  - For the case of Spring Rest, it is just habitual or psychological: People are used to Spring MVC and love it and want to continue to use almost the same approach in Restful web services.

Usage Scenarios of Spring

- For more info regarding the usage scenarios of **Spring** you can have a look at **Usage Scenarios** part of early **Spring** references such as https://docs.spring.io/spring/docs/1.2.0/reference/introduction.html

Spring Documentation

Spring Framework Documentation

- https://spring.io/
- **Spring** guides: https://spring.io/guides
- For current **Spring** documentation: https://docs.spring.io/spring/docs/current/spring-framework-reference/
- The wiki that complements **Spring** documentation: https://github.com/spring-projects/spring-framework/wiki
- The root for all versions of **Spring** documentation: https://docs.spring.io/spring-framework/docs/

Spring API

- Current **Spring** API: https://docs.spring.io/spring-framework/docs/current/javadoc-api/

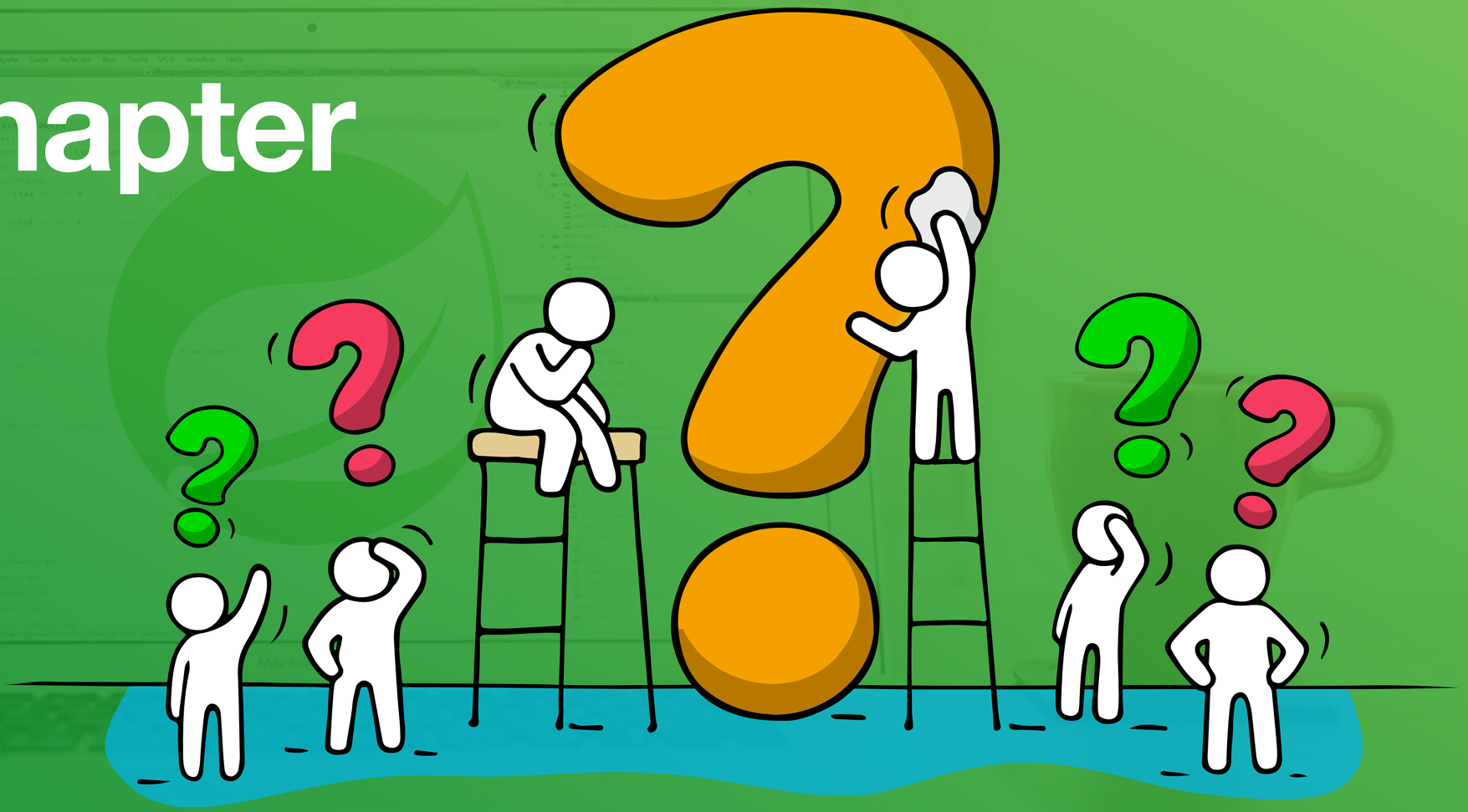
Homeworks

Homeworks

1. Please visit: https://spring.io/
2. Read **Spring Overview** https://docs.spring.io/spring/docs/current/spring-framework-reference/overview.html#overview
3. Read **Why Spring?** https://spring.io/why-spring

End of Chapter

Time for Questions!



selsoft
build better, deliver faster

info@selsoft.com.tr
selsoft.com.tr