



Spring Framework Fast Track 2019

A complete of essential concepts and examples

Author: Erfin Feluzy & Wahyu Sudrajat
March 2019

Profile

ERFIN FELUZY

- 11+ years professional experience in Java Programming (10+ years spring framework)
- General Manager in Software Solution
- Sr Middleware Solution Architect

WAHYU SUDRAJAT

- 8+ years professional experience in Spring
- Linux master & Mathematician
- Solution Architect in Software Solution



Agenda Day 1 (March 9, 2019)

09:00 – 10:00 : Spring Framework intro

10:00 – 10:30 : Spring Boot intro (Hands-On)

10:30 – 11:00 : Basic build with Maven (Hands-On)

11:00 – 12:00 : Hands-On prerequisite setup

12:00 – 13:00 : Ishoma

13:00 – 14:00 : Spring web MVC (Hands-On)

14:00 – 15:00 : Spring Data Database (Hands-On)

15:00 – 16:00 : Spring Web Service REST producer (Hands-On)

16:00 – 17:00 : Spring Web Service SOAP producer (Hands-On)

Agenda Day 2 (March 16, 2019)

09:00 – 10:00 : Spring Web Service REST consumer (Hands-On)

10:00 – 11:00 : Spring Web Service SOAP consumer (Hands-On)

11:00 – 12:00 : Spring JMS Publisher & Subscriber (Hands-On)

12:00 – 13:00 : Ishoma

13:00 – 14:30 : Lab : Use Case 1

14:30 – 16:00 : Lab : Use Case 2

16:00 – 17:00 : Recap & Closing



What is Spring?

Back then, enterprise application commonly use a technology called **Enterprise Java Bean (EJB)** which relatively heavy and tightly depend on enterprise solution such as: Application Web Server(s).

What is Spring?

- An open-source framework.
- An alternative to heavier enterprise Java technologies.
- Addresses the complexity of enterprise application development, thus it “Simplifies Java development”

<https://www.youtube.com/watch?v=a14xfNV-UIs>



Spring is Non-Invasive

What does that mean?

- You are not forced to import or extend any Spring APIs
- An invasive framework takes over your code.
- Anti-pattern: (sample of invasive framework)
 - **EJB** forces you to use **Java Naming and Directory Interface (JNDI)**, commonly found at enterprise application framework
 - **Struts** forces you to extend **Action, ActionForm**

Invasive frameworks are inherently difficult to test. You have to stub the runtime that is supplied by the application server.

<https://www.slideshare.net/pmanvi/spring-framework-overview-ppt-3068245>

Benefit of Using Spring

Spring is a Framework which help to solve common software developing problems with Java programming language, such as:

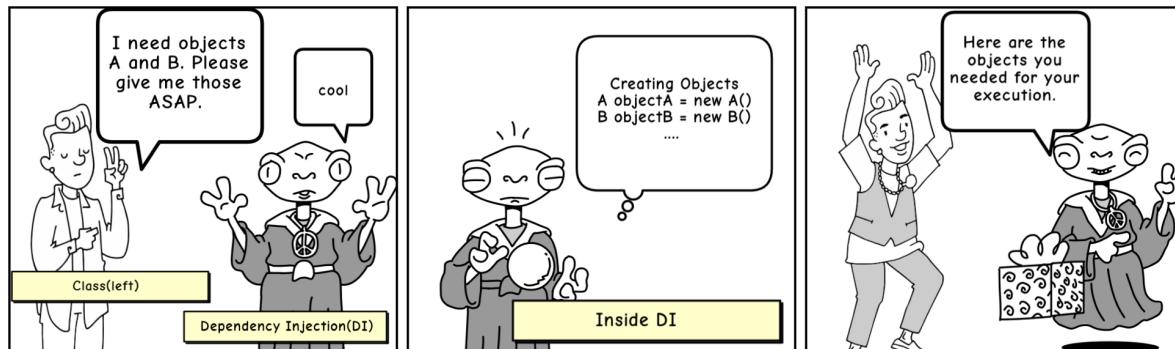
- Manage lifecycle of Java classes (called beans)
- Dependency Injection framework
- Provide Data Access which commonly works through Java Database Connectivity (JDBC), Object-Relational Mapping (ORM), etc.
- Provide Spring-MVC which standardize patterns and structure to develop Web Application
- more...

<https://www.youtube.com/watch?v=gq4S-ovWVIM>

Dependency Injection (DI)

Wikipedia definition of DI:

*“In software engineering, **dependency injection** is a technique whereby one object (or static method) supplies the dependencies of another object. A dependency is an object that can be used (a service)”*



This comic was created at www.MakeBeliefsComix.com. Go there and make one now!

<https://medium.freecodecamp.org/a-quick-intro-to-dependency-injection-what-it-is-and-when-to-use-it-7578c84fa88f>

What is Spring Boot?

Although Spring does simplify Java development process, it can give programmers an adversity while developing an application.

- Boiler plate configuration
 - Programmer writes a lot of codes (configurations) to do minimal task
- Takes time to have Spring application up and running

Spring Boot offers the solution for those problems. Spring Boot is a suite of pre-configured framework and technologies and it is a shortest way to have Spring application up and running.

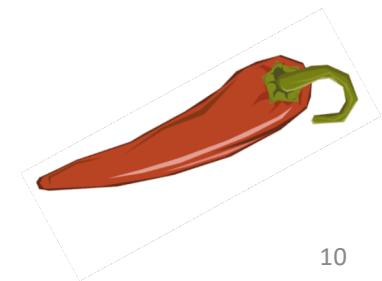
<https://www.youtube.com/watch?v=LLfZ2g2Jalc>

Hand-On Prerequisites Setup

- Oracle SDK 8+, you might also use OpenJDK, make sure to have JDK installed instead of JRE
- Java IDE, Eclipse 2018-12 is preferred
 - Eclipse by default comes with Apache Maven embedded
- Properly setup Java Runtime Environment (JRE) on Environment Variable (Windows), or PATH variable (Unix based)
- Lombok Project for Simplify POJO

OpenJDK

eclipse

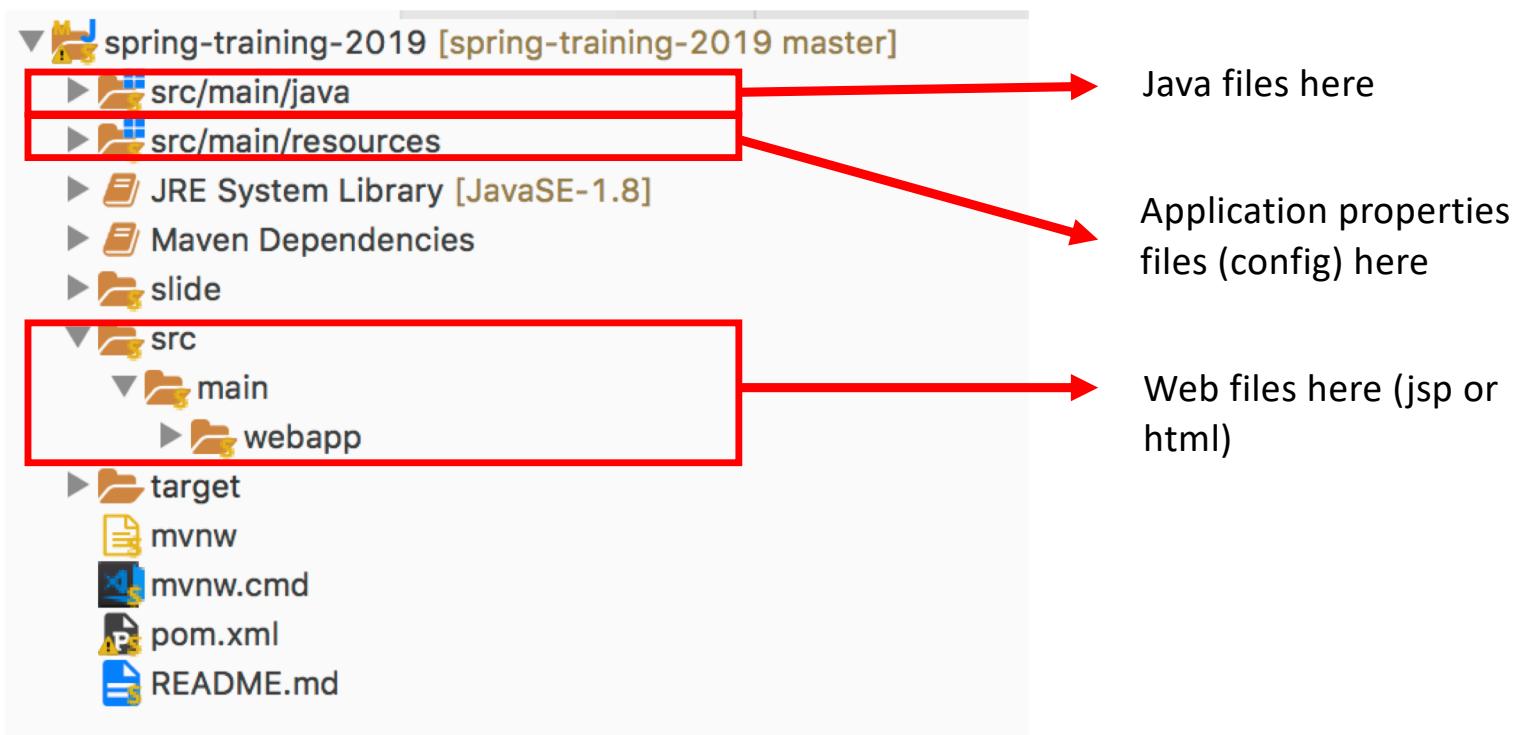




Spring Boot Hands-On

Spring Boot Intro

Maven Common Project Structure



Spring Boot's pom.xml

```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>1.4.3.RELEASE</version>
</parent>
```

Spring boot parent artifact

```
<dependencies>
  <!-- Spring Boot -->
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
```

Based on web MVC

Spring Boot Starter

```
J Application.java ✘
1 package com.erfinfeluzy.training;
2
3 import org.springframework.boot.SpringApplication;
4
5
6 @SpringBootApplication(scanBasePackages = { "com.erfinfeluzy" })
7
8 @PropertySource({
9     "classpath:apps.properties"
10 })
11
12 public class Application {
13
14     public static void main(String[] args) {
15
16         SpringApplication.run(Application.class, args);
17     }
18
19 }
20
21
```

Scan all spring bean inside base package

Spring boot custom configuration.
Eg: server port, database, etc

Spring boot custom configuration.
Eg: server port, database, etc

Maven Hands-On

Basic Build with Apache Maven

Apache Maven

Maven is a software management and comprehension tool based on the concept of Project Object Model(POM). It can manage project build, reporting and documentation, from a central piece of information. Basically, this tool can be used as both, a build tool (just like ANT tool), as well as a project management tool. However, in this article we will use it just for building our project.

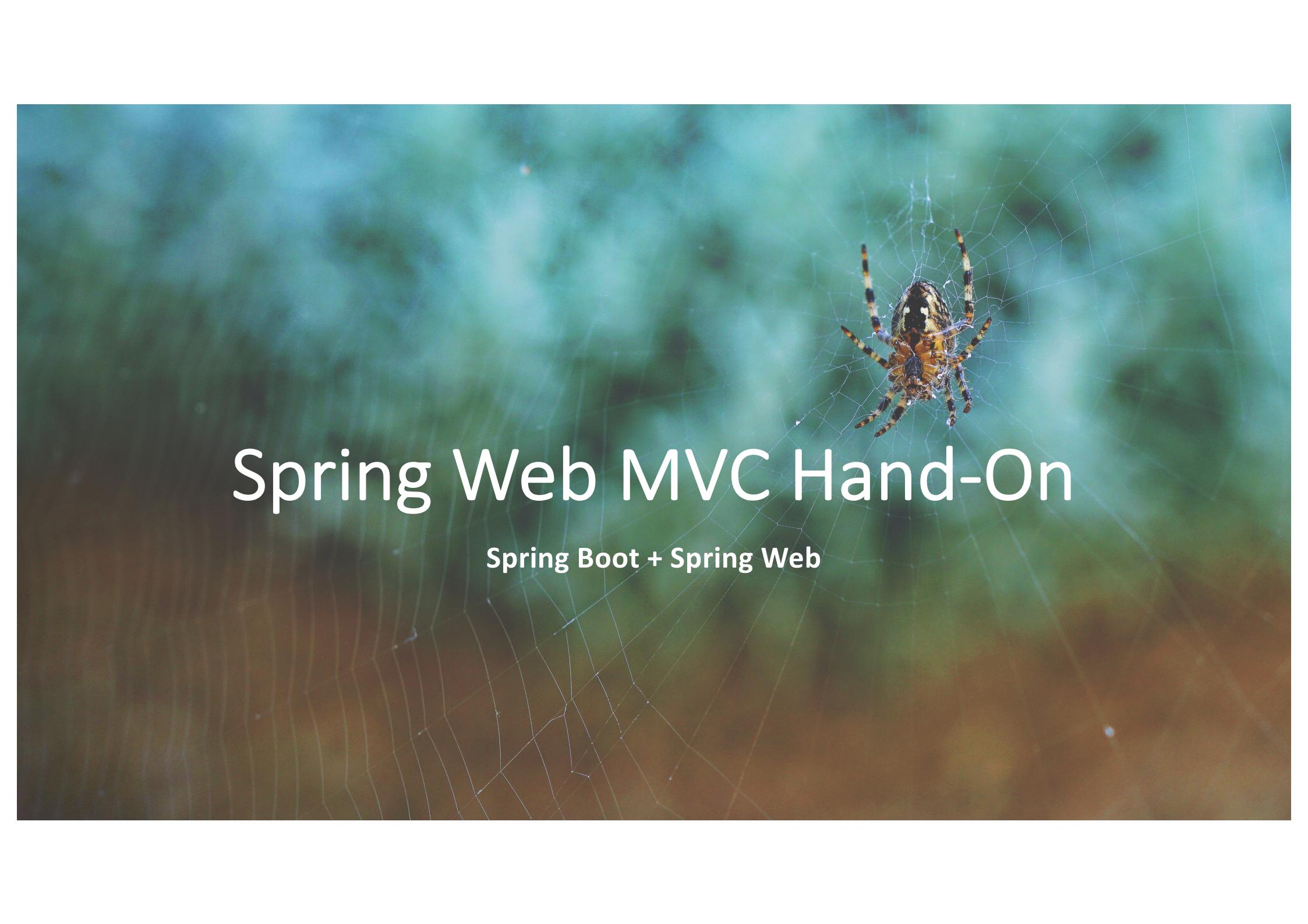
Basic usage of maven are:

```
$> mvn clean  
$> mvn compile  
$> mvn install
```



maven

<https://medium.com/@himanshuagarwal1395/getting-started-with-apache-maven-hello-world-eccb278a262a>

The background of the image is a close-up photograph of a spider's web. A single spider is visible in the center-right portion of the web. The web is set against a soft-focus background that transitions from a bright green at the top to a warm orange and yellow at the bottom.

Spring Web MVC Hand-On

Spring Boot + Spring Web

Spring MVC (Model-View-Controller)

```
J Customer.java ✘
1 package com.erfinfeluzy.training.spring.model;
2
3 import java.util.Date;
4
5 @Data
6 @Entity
7 @Table(name = "tbl_customer")
8 public class Customer {
9
10     @Id
11     @GeneratedValue
12     private Long id;
13
14     @Column(name="username")
15     private String username;
16
17     @Column(name="first_name")
18     private String firstName;
19
20     @Column(name="last_name")
21     private String lastName;
22
23     @Column(name="birth_date")
24     private Date birthdate;
25 }
```

model

```
@Controller
public class HomeController { controller
    @RequestMapping(value = "/", method = RequestMethod.GET)
    public String home() {
        System.out.println("masuk : / ");
        return "home";
    }
}
```

```
home.jsp ✘
1 <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
2 <html>
3
4     <head>
5         <title>Training</title>
6     </head>
7
8     <body>
9
10        <h2>Halo Trainee</h2>
11
12        <p>greetings: ${greetings}</p>
13
14    </body>
15
16 </html>
```

view

A close-up photograph of a yellow Ethernet cable against a solid blue background. The cable is coiled and curves across the frame, with its white RJ45 connector visible at the top center.

Spring Data Hand-On

Spring Boot + Spring Data

Spring Boot Data (config)

```
spring.datasource.url=jdbc:h2:mem:db;DB_CLOSE_DELAY=-1;INIT=CREATE SCHEMA IF NOT EXISTS USERS
spring.datasource.username=sa
spring.datasource.password=sa
spring.datasource.driver-class-name=org.h2.Driver

spring.jpa.properties.hibernate.hbm2ddl.auto=update
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.H2Dialect
spring.jpa.properties.hibernate.show_sql=true

spring.h2.console.enabled=true
spring.h2.console.path=/console
```

Spring Boot Data (custom SQL query)

```
J CustomerRepository.java ✘
1 package com.erfinfeluzy.training.spring.service;
2
3 import java.util.List;
4
5 @Transactional
6 public interface CustomerRepository extends JpaRepository<Customer, Long> {
7
8     @Query(value = "select * from tbl_customer where first_name like :firstname",
9         nativeQuery = true)
10    List<Customer> findAllCustomerNativeSQL(
11        @Param("firstname") String firstname);
12
13 }
```

Invoke JDBC Service - Create

```
J HomeController.java X
26
27 @Autowired
28 CustomerRepository customerRepository;
29
30 @RequestMapping(value = "/customer/add", method = RequestMethod.GET)
31 public String customerAdd() {
32
33     System.out.println("/customer/add");
34
35     Customer cust = new Customer();
36     cust.setFirstName("erfin");
37     cust.setLastName("feluzy");
38     cust.setBirthdate(new Date());
39     cust.setUsername("erfinfeluzy");
40
41     customerRepository.save(cust);
42
43     return "add_customer";
44 }
45
```

Annotations and code segments are highlighted with red boxes:

- Line 28: `@Autowired CustomerRepository customerRepository;` is highlighted with a red box and has a red arrow pointing to the text "Dependency Injection (Repository)".
- Lines 35-40: `Customer cust = new Customer();
cust.setFirstName("erfin");
cust.setLastName("feluzy");
cust.setBirthdate(new Date());
cust.setUsername("erfinfeluzy");` are highlighted with a red box and have a red arrow pointing to the text "Create New Object".
- Line 41: `customerRepository.save(cust);` is highlighted with a red box and has a red arrow pointing to the text "Save new Customer to Database".

Invoke JDBC Service - Read

```
@RequestMapping(value = "/customer/all", method = RequestMethod.GET)
public String customerList(Model model) {
    List<Customer> customers = customerRepository.findAll();
    model.addAttribute("customers", customers);
    return "customers";
}
```

The diagram illustrates the execution flow of the Java code. It shows two highlighted sections of code with corresponding annotations:

- The first highlighted section is `List<Customer> customers = customerRepository.findAll();`. An arrow points from this section to the annotation **Find All Customer**.
- The second highlighted section is `model.addAttribute("customers", customers);`. An arrow points from this section to the annotation **Put as attribute on viewer (jsp)**.

Spring Web Service REST Producer Hand-On

Spring Boot + Spring Web



Web Service REST Config

J HelloWorldRestController.java X

```
1 package com.erfinfeluz.y.training.spring.rest;
2
3 import org.springframework.http.ResponseEntity;
4
5 @RestController
6 @RequestMapping("/api")
7 public class HelloWorldRestController {
8
9     /**
10      * Serve as {@link #getHelloWorld()} which constructed as REST producer.
11      */
12     @GetMapping(value = "/hello")
13     public ResponseEntity<?> getHelloWorld() {
14
15         Hello hello = new Hello();
16         hello.setStatus("success");
17         hello.setMessage("Hello World!");
18
19         return ResponseEntity.ok().body(hello);
20     }
21
22     // define inner class which used specifically by this controller
23     @Data
24     private class Hello {
25         private String status;
26         private String message;
27     }
28 }
```

Rest Web Service cd /api/hello

Spring Web Service REST Consumer Hand-On

Spring Boot + Spring Web

Spring Web Service SOAP Producer Hand-On

Spring Boot + Spring Web



Web Service SOAP Producer Config

```
SOAPConfig.java JmsConfig.java Sender.java Receiver.java TestJmsRestCont »11 □
```

```
2  
3④ import javax.xml.ws.Endpoint;  
17  
18 @SuppressWarnings("deprecation")  
19 @Configuration  
20 public class SOAPConfig {  
21  
22     /** SOAP producer */  
23  
24④     @Bean  
25     public ServletRegistrationBean cxfDispatcherServlet() {  
26         return new ServletRegistrationBean(new CXFServlet(), "/soap/*");  
27     }  
28  
29④     @Bean(name = Bus.DEFAULT_BUS_ID)  
30     public SpringBus springBus(){  
31         SpringBus springBus = new SpringBus();  
32         return springBus;  
33     }  
34  
35④     @Bean  
36     public Endpoint endpoint() {  
37         EndpointImpl endpoint = new EndpointImpl(springBus(), customerSoapService());  
38         endpoint.getFeatures().add(new LoggingFeature());  
39         endpoint.publish("/CustomerService");  
40         return endpoint;  
41     }  
42  
43④     @Bean  
44     public CustomerSoapService customerSoapService() {  
45         return new CustomerSoapServiceImpl();  
46     }  
47
```

SOAP Web Service:
/soap/CustomerService

Spring Web Service SOAP Consumer Hand-On

Spring Boot + Spring Web

Web Service SOAP Consumer Config

```
CustomerSoapSer CustomerSoapSer SOAPConfig.java SoapConsumerCon »12
32     return springBus;
33 }
34
35 @Bean
36 public Endpoint endpoint() {
37     EndpointImpl endpoint = new EndpointImpl(springBus(), customerSoapService());
38     endpoint.getFeatures().add(new LoggingFeature());
39     endpoint.publish("/CustomerService");
40     return endpoint;
41 }
42
43 @Bean
44 public CustomerSoapService customerSoapService() {
45     return new CustomerSoapServiceImpl();
46 }
47
48 /**
49 * SOAP consumer */
50 @Bean(name = "client")
51 public CustomerSoapService customerServiceProxy() {
52     return (CustomerSoapService) proxyFactoryBean().create();
53 }
54
55 @Bean
56 public JaxWsProxyFactoryBean proxyFactoryBean() {
57     JaxWsProxyFactoryBean proxyFactory = new JaxWsProxyFactoryBean();
58     proxyFactory.setServiceClass(CustomerSoapService.class);
59     proxyFactory.setAddress("http://localhost:8080/soap/CustomerService");
60     return proxyFactory;
61 }
62
63 }
64 }
```

SOAP Web Service:
/soap/CustomerService

Spring JMS Publisher & Subscriber Hand-On

Spring Boot + Spring JMS

JMS Publisher/Subscriber Java Config



```
CustomerSoapSer SOAPConfig.java SoapConsumerCon JmsConfig.java Sender.java >11
1 package com.erfinfeluzy.training.config;
2
3 import org.springframework.context.annotation.Bean;
4 import org.springframework.context.annotation.Configuration;
5 import org.springframework.jms.annotation.EnableJms;
6 import org.springframework.jms.support.converter.MappingJackson2MessageConverter;
7 import org.springframework.jms.support.converter.MessageConverter;
8 import org.springframework.jms.support.converter.MessageType;
9
10 @Configuration
11 @EnableJms
12 public class JmsConfig {
13
14     // Serialize message content to json using TextMessage
15     @Bean("jacksonMessageConverter")
16     public MessageConverter jacksonJmsMessageConverter() {
17         MappingJackson2MessageConverter converter = new MappingJackson2MessageConverter();
18         converter.setTargetType(MessageType.TEXT);
19         converter.setTypeIdPropertyName("_type");
20         return converter;
21     }
22
23 }
24
```

Message Converter for Object

JMS Publisher (Sender)

```
apps.properties JmsConfig.java Sender.java Receiver.java TestJmsRestCont »11 TOMORROW United States
2
3④ import org.springframework.beans.factory.annotation.Autowired;⑤
10
11 @Component
12 public class Sender {
13
14     private static final String MAILBOX_QUEUE_DESTINATION = "mailbox.queue";
15     private static final String TEXT_QUEUE_DESTINATION = "text.queue";
16
17
18④     @Autowired
19     @Qualifier("jacksonMessageConverter")
20     private MessageConverter jacksonMessageConverter;
21
22④     @Autowired
23     private JmsTemplate jmsTemplate;
24
25
26     /** Queue */
27
28④     public void sendTextMessage() {
29         jmsTemplate.convertAndSend(TEXT_QUEUE_DESTINATION,
30             "Test kirim pesan!");
31     }
32
33④     public void sendEmailMessage() {
34         jmsTemplate.setMessageConverter(jacksonMessageConverter);
35         jmsTemplate.convertAndSend(MAILBOX_QUEUE_DESTINATION,
36             new Email("admin@localhost", "Connection test!"));
37     }
38
39 }
40
```

The code snippet shows a Java class named `Sender`. It contains two static final strings for destinations: `MAILBOX_QUEUE_DESTINATION` and `TEXT_QUEUE_DESTINATION`. Below these, there are two methods: `sendTextMessage()` and `sendEmailMessage()`. The `sendTextMessage()` method uses the `jmsTemplate.convertAndSend` method with the `TEXT_QUEUE_DESTINATION` and a test message. The `sendEmailMessage()` method uses the same template but with the `MAILBOX_QUEUE_DESTINATION` and a new `Email` object.

A red box highlights the destination configuration (lines 14-15) with an arrow pointing to the text "Destination". Another red box highlights the message sending logic (lines 28-38) with an arrow pointing to the text "JMS Queue".

JMS Subscriber (Receiver)

```
apps.properties JmsConfig.java Sender.java Receiver.java TestJmsRestCont »11 United States
1 package com.erfinfeluzy.training.spring.jms;
2
3 import org.springframework.jms.annotation.JmsListener;
4 import org.springframework.stereotype.Component;
5
6 import com.erfinfeluzy.training.spring.model.Email;
7
8 @Component
9 public class Receiver {
10
11     private static final String MAILBOX_QUEUE_DESTINATION = "mailbox.queue";
12     private static final String TEXT_QUEUE_DESTINATION = "text.queue";
13
14
15     /** Queue ***/
16
17     @JmsListener(destination = TEXT_QUEUE_DESTINATION)
18     public void receiveTextMessage(String message) {
19         System.out.println("Queue Received <" + message + ">");
20     }
21
22     @JmsListener(destination = MAILBOX_QUEUE_DESTINATION)
23     public void receiveEmailMessage(Email email) {
24         System.out.println("Queue Received <" + email + ">");
25     }
26
27 }
28
```

The code snippet shows a Java class named `Receiver`. It contains two static final string fields: `MAILBOX_QUEUE_DESTINATION` and `TEXT_QUEUE_DESTINATION`. Below these, there are two `@JmsListener` annotations. The first listener is for the `TEXT_QUEUE_DESTINATION` and has a method `receiveTextMessage` that prints the received message to the console. The second listener is for the `MAILBOX_QUEUE_DESTINATION` and has a method `receiveEmailMessage` that also prints the received message to the console.

A red box highlights the declaration of the two destination constants and the bodies of the two listener methods. Two red arrows point from this highlighted area to the text "Destination" and "JMS Queue" respectively, indicating the relationship between the code and the concepts being explained.

Lab – Use Case 1

Create Multiple Backend Web Services

Lab – Use Case 2

Create Microservices apps with Spring Boot

Recap & Closing