in28minutes

Spring Master Class - Course Guide

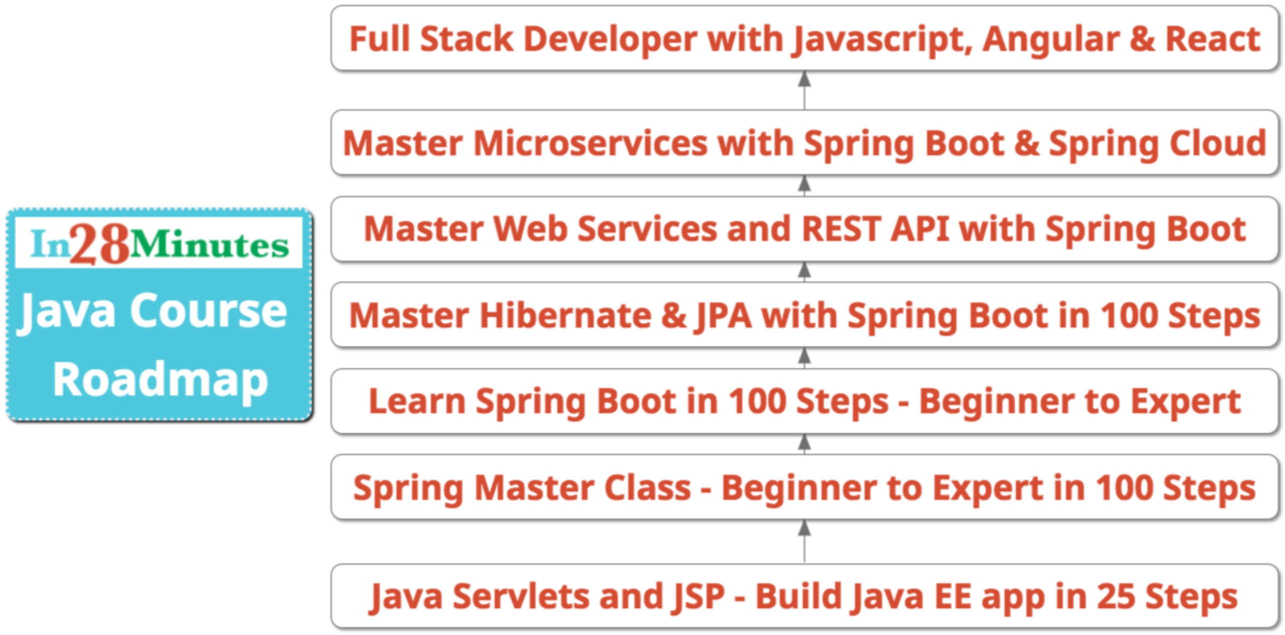
Master the most popular Java framework in six easy levels!

Table of Contents

Congratulations

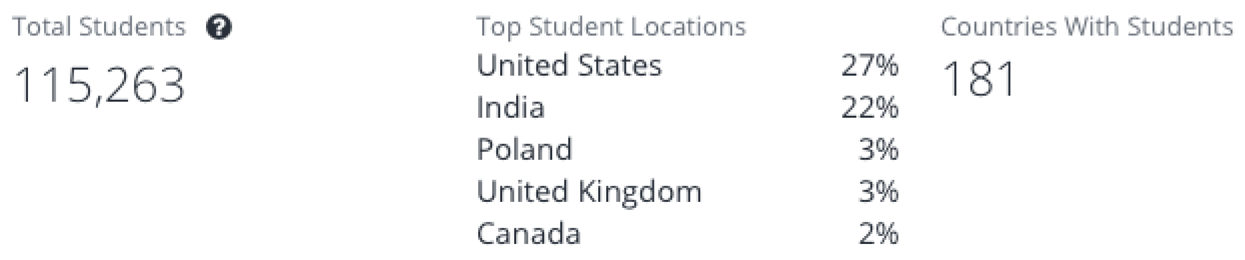
You have made a great choice in learning with in28Minutes. You are joining 100,000+ Learners learning everyday with us.

100,000+ Java beginners are learning from in28Minutes to become experts on APIs, Web Services and Microservices with Spring, Spring Boot and Spring Cloud.


In28Minutes
Java Course Roadmap
Full Stack Developer with Javascript, Angular & React
Master Microservices with Spring Boot & Spring Cloud
Master Web Services and REST API with Spring Boot
Master Hibernate & JPA with Spring Boot in 100 Steps
Learn Spring Boot in 100 Steps - Beginner to Expert
Spring Master Class - Beginner to Expert in 100 Steps
Java Servlets and JSP - Build Java EE app in 25 Steps

About in28Minutes

How did in28Minutes get to 100,000 learners across the world?



Total Students

115,263

Top Student Locations

United States	27%
India	22%
Poland	3%
United Kingdom	3%
Canada	2%

Countries With Students

181

We are focused on creating the awesome course (learning) experiences. Period.

An awesome learning experience?

What's that?

You need to get insight into the in28Minutes world to answer that.

You need to understand "***The in28Minutes Way***"

- What are our beliefs?
- What do we love?
- Why do we do what we do?
- How do we design our courses?

Let's get started on "***The in28Minutes Way***"!

Important Components of "The in28Minutes Way"

- Continuous Learning
- Hands-on
- We don't teach frameworks. We teach building applications!
- We want you to be strong on the fundamentals

- Step By Step
- Efficient and Effective
- Real Project Experiences
- Debugging and Troubleshooting skills
- Modules - Beginners and Experts!
- Focus on Unit Testing
- Code on Github

- Design and Architecture
- Modern Development Practices
- Interview Guides
- Bring the technology trends to you
- Building a connect
- Socially Conscious
- We care for our learners
- We love what we do

Troubleshooting Guide

We love all our 100,000 learners. We want to help you in every way possible.

We do not want you to get stuck because of a simple error.

This 50 page troubleshooting guide and faq is our way of thanking you for choosing to learn from in28Minutes.

.in28Minutes Trouble Shooting Guide

Getting Started

Recommended Versions

Tool/Framework/Language	Recommended Version	More Details
Java	Java 8	http://www.in28minutes.com/spr...
Eclipse	Eclipse Java EE Oxygen	Basics
Spring Boot	Spring Boot 2.0.0.M3	Configure 2.0.0.M3
Spring	Any Release of Spring 5.0.0	

Installation

- Video : https://www.youtube.com/playlist?list=PLBBog2r6uMCSmMVTW_QmDLyASBvovyAO3
- PDF : https://github.com/in28minutes/SpringIn28Minutes/blob/master/InstallationGuide-JavaEclipseAndMaven_v2.pdf
- More Details : https://github.com/in28minutes/getting-started-in-5-steps

Troubleshooting

- A 50 page troubleshooting guide with more than 200 Errors and Questions answered

Spring Master Class - Course Overview

Github Repository :

https://github.com/in28minutes/spring-master-class

Spring Level 1 to Level 6 - Sections Overview

Title	Category	Github Folder
Spring Framework in 10 Steps	Spring - Level 1	Project Folder on Github
Spring in Depth	Spring - Level 2	Project Folder on Github
Unit Testing with Spring Framework	Spring - Level 3	Project Folder on Github
Spring Boot in 10 Steps	Spring - Level 4	Project Folder on Github
Spring AOP	Spring - Level 5	Project Folder on Github
Spring JDBC and JPA	Spring - Level 6	Project Folder on Github

5 Bonus Sections - Introduction to Tools and Frameworks

Title	Category	Github Folder
Eclipse in 5 Steps	Introduction	Project Folder on Github
Maven in 5 Steps	Introduction	Project Folder on Github
JUnit in 5 Steps	Introduction	Project Folder on Github

Mockito in 5 Steps	Introduction	Project Folder on Github
Basic Web Application with Spring MVC	Introduction	Project Folder on Github

Spring Level 1 - First 10 Steps in Spring

Spring Level 1 - First 10 Steps in Spring

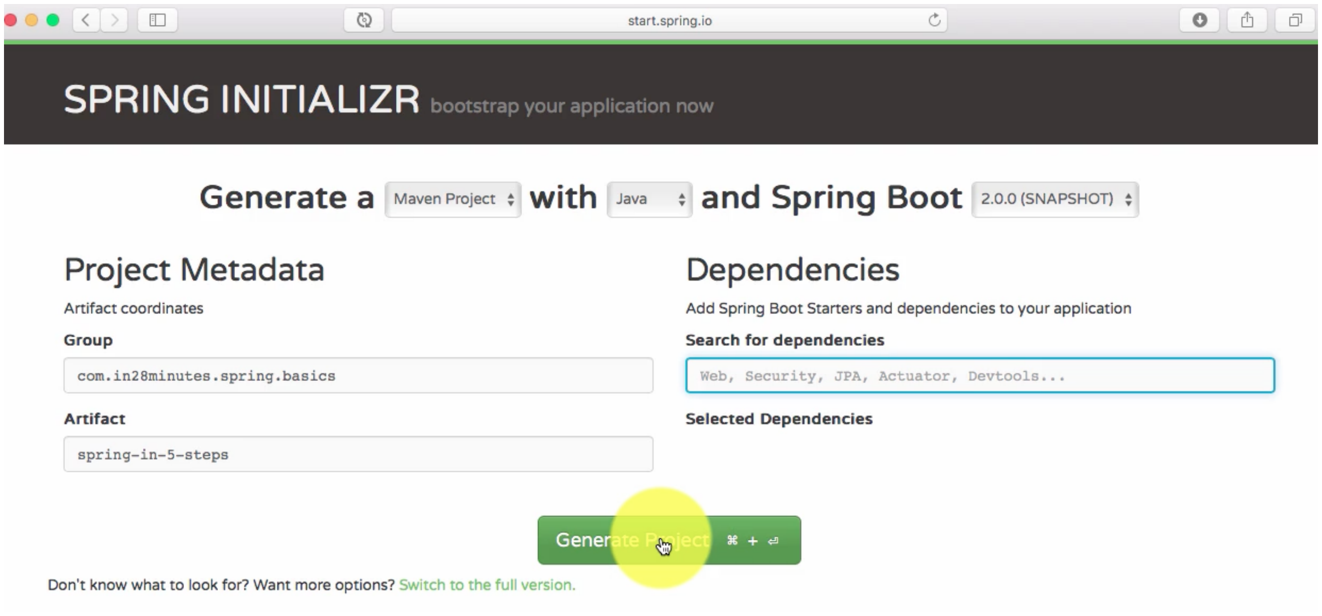
Title	Category	Github
Spring Framework in 10 Steps	Spring - Level 1	Project Folder on Github

- Step 1 : Setting up a Spring Project using htttp://start.spring.io
- Step 2 : Understanding Tight Coupling using the Binary Search Algorithm Example
- Step 3 : Making the Binary Search Algorithm Example Loosely Coupled
- Step 4 : Using Spring to Manage Dependencies - @Component, @Autowired
- Step 5 : What is happening in the background?
- Step 6 : Dynamic auto wiring and Troubleshooting - @Primary
- Step 7 : Constructor and Setter Injection
- Step 8 : Spring Modules
- Step 9 : Spring Projects
- Step 10 : Why is Spring Popular?

Step 1 : Setting up a Spring Project using htttp://start.spring.io
One of the most important features of Spring Framework is dependency injection. Spring framework helps in creating loosely coupled applications. To be able to appreciate dependency injection you should understand tight coupling and how to create loosely coupled applications. What we will start with is setting up a simple example to be able to understand tight couplings and also dependency injection.

Creating a Spring Project with Spring Initializr is a cake walk.

> *Spring Initializr http://start.spring.io/ is great tool to bootstrap your Spring Boot projects.*


start.spring.io

SPRING INITIALIZR bootstrap your application now

Generate a Maven Project with Java and Spring Boot 2.0.0 (SNAPSHOT)

Project Metadata
Artifact coordinates
Group
com.in28minutes.spring.basics
Artifact
spring-in-5-steps

Dependencies
Add Spring Boot Starters and dependencies to your application
Search for dependencies
Web, Security, JPA, Actuator, Devtools...
Selected Dependencies

Generate Project ⌘ + ⏎

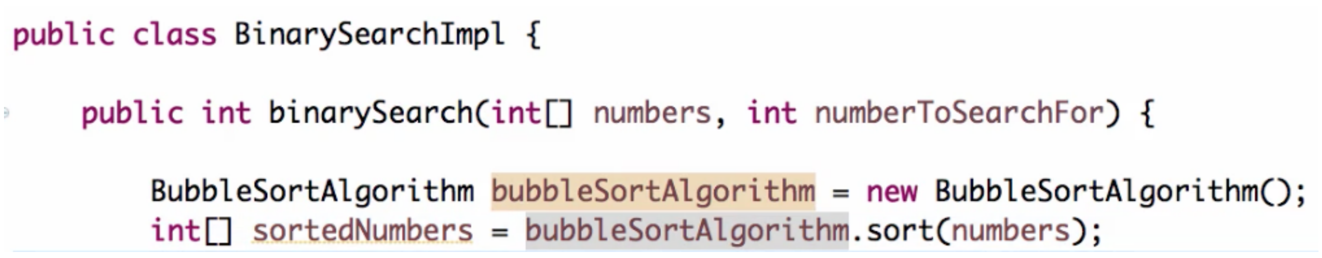
Don't know what to look for? Want more options? Switch to the full version.

As shown in the image above, following steps have to be done

- Launch Spring Initializr and choose the following
    - Choose `com.in28minutes.spring.basics` as Group
    - Choose `spring-in-5-steps` as Artifact
    - Do not choose any dependencies
        - By default Basic Starter is included, which has the core spring framework and the spring test starter.

- Click Generate Project.
- Import the project into Eclipse.
- If you want to understand all the files that are part of this project, you can go here.

Step 2 : Understanding Tight Coupling using the Binary Search Algorithm Example
Set up an example of tight coupling with Binary Search and Bubble Sort Algorithm as shown in the picture below.


```
public class BinarySearchImpl {

    public int binarySearch(int[] numbers, int numberToSearchFor) {

        BubbleSortAlgorithm bubbleSortAlgorithm = new BubbleSortAlgorithm();
        int[] sortedNumbers = bubbleSortAlgorithm.sort(numbers);
```

However, we have a problem with above implementation. If we want to use binary search with a different sort algorithm, I would need to change the code.

We want to make the binary search algorithm loosely coupled so that it can work with any sort algorithm.

Think about the solution before moving to next step!

Step 3 : Making the Binary Search Algorithm Example Loosely Coupled
Introduce an interface to make the Binary Search loosely coupled to the sort algorithm.

```
package com.in28minutes.spring.basics.springin5steps;

public interface SortAlgorithm {
        public int[] sort(int[] numbers);
}

public class BinarySearchImpl {

        private SortAlgorithm sortAlgorithm;
```

Step 4 : Using Spring to Manage Dependencies - @Component, @Autowired

In the previous steps - we wrote code to create objects of the bubble sort algorithm and binary search. We also managed the dependencies. It would be great actually if some framework can take control of creation of the beans and autowiring the dependencies.

That's where Spring Framework comes in!

Let's start using Spring to do autowiring.

Notes

- Sort algorithm is a dependency of the binary search.

```
@Component
public class BinarySearchImpl {

        @Autowired
        private SortAlgorithm sortAlgorithm;

@Component
public class BubbleSortAlgorithm implements SortAlgorithm {
        public int[] sort(int[] numbers) {
                // Logic for Bubble Sort
                return numbers;
        }
}
```

Step 5 : What is happening in the background?
Enable debug logging and check what's happening in the background.

/src/main/resources/application.properties

```
logging.level.org.springframework = debug
```

- Spring does a Component scan on the parent package "com.in28minutes.spring.basics.springin5steps" to find the components - classes that have @Component on them.
- It identifies components and dependencies
- It identifies that BinarySearchImpl has a dependency SortAlgorithm
- It identifies that SortAlgorithm does not have a dependency. So, it creates an instance of it and autowires it into an instance of BinarySearchImpl

Step 6 : Dynamic auto wiring and Troubleshooting - @Primary
What if we add one more SortAlgorithm?

```
package com.in28minutes.spring.basics.springin5steps;

import org.springframework.stereotype.Component;

@Component
public class QuickSortAlgorithm implements SortAlgorithm {
        public int[] sort(int[] numbers) {
                // Logic for Quick Sort
                return numbers;
        } }
```

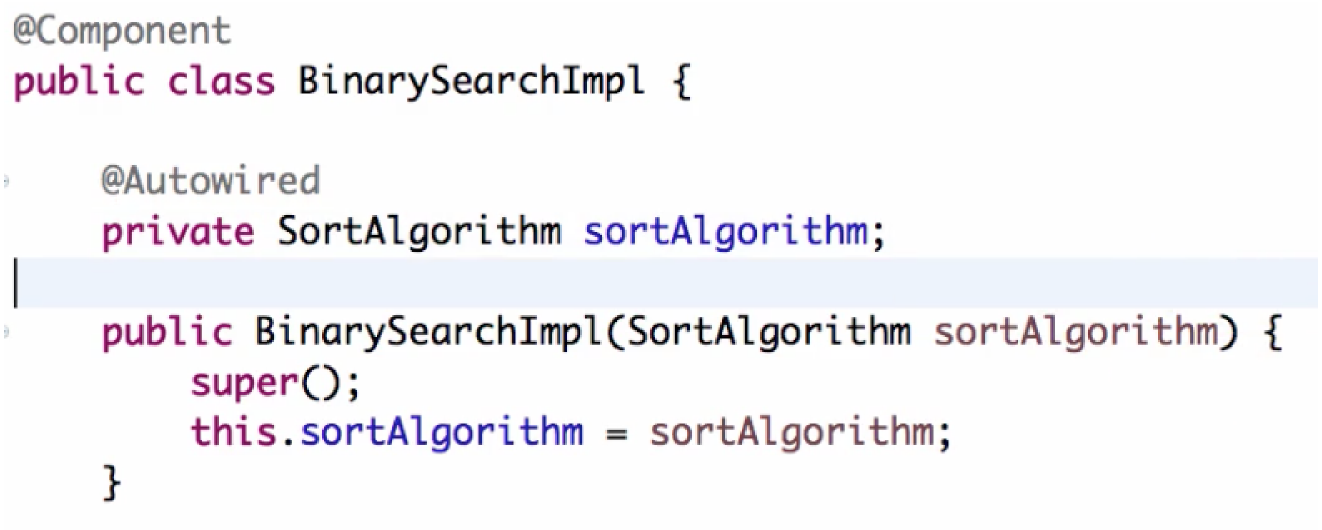
There are now two matching SortAlgorithm instances. Spring throws an exception because it does not know which one to use.

We use @Primary to mark one of the SortAlgorithm implementations is more important!

```
package com.in28minutes.spring.basics.springin5steps;

import org.springframework.context.annotation.Primary;
import org.springframework.stereotype.Component;

@Component
@Primary
public class BubbleSortAlgorithm implements SortAlgorithm {
        public int[] sort(int[] numbers)  {
                // Logic for Bubble Sort
                return numbers;
        }
}
```

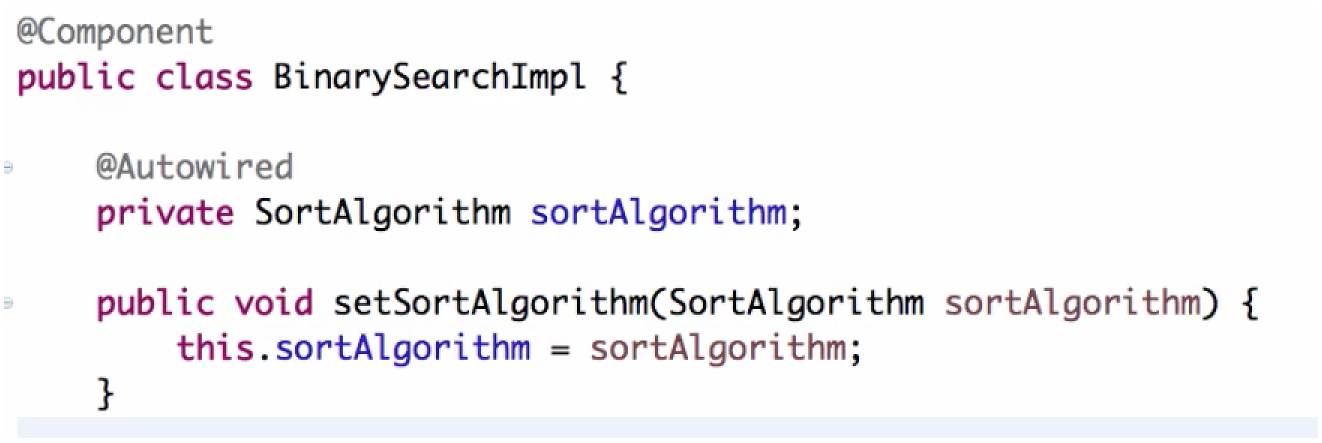
Step 7 : Constructor and Setter Injection

Constructor Injection



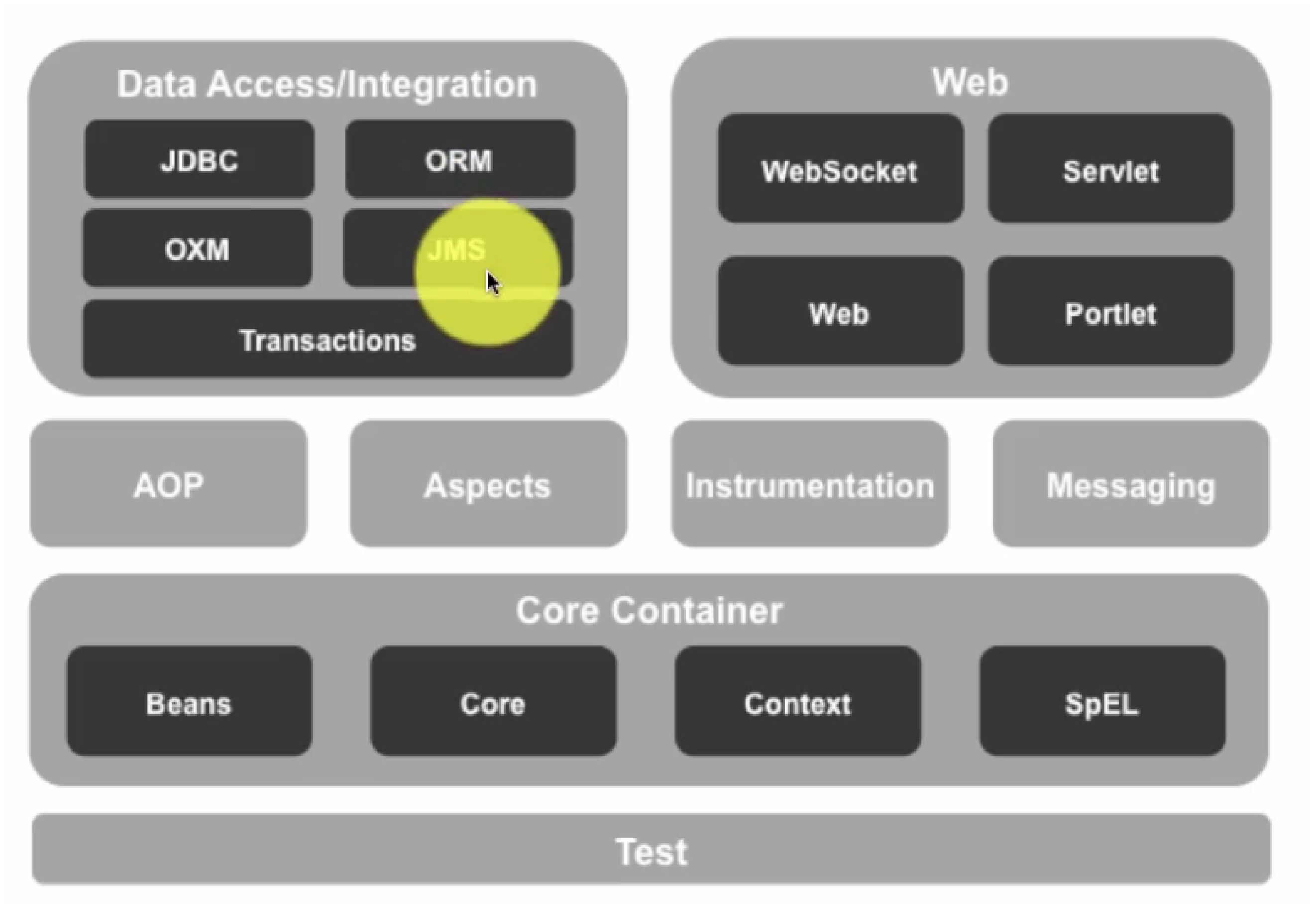
```
@Component
public class BinarySearchImpl {

    @Autowired
    private SortAlgorithm sortAlgorithm;

    public BinarySearchImpl(SortAlgorithm sortAlgorithm) {
        super();
        this.sortAlgorithm = sortAlgorithm;
    }
```

Setter Injection

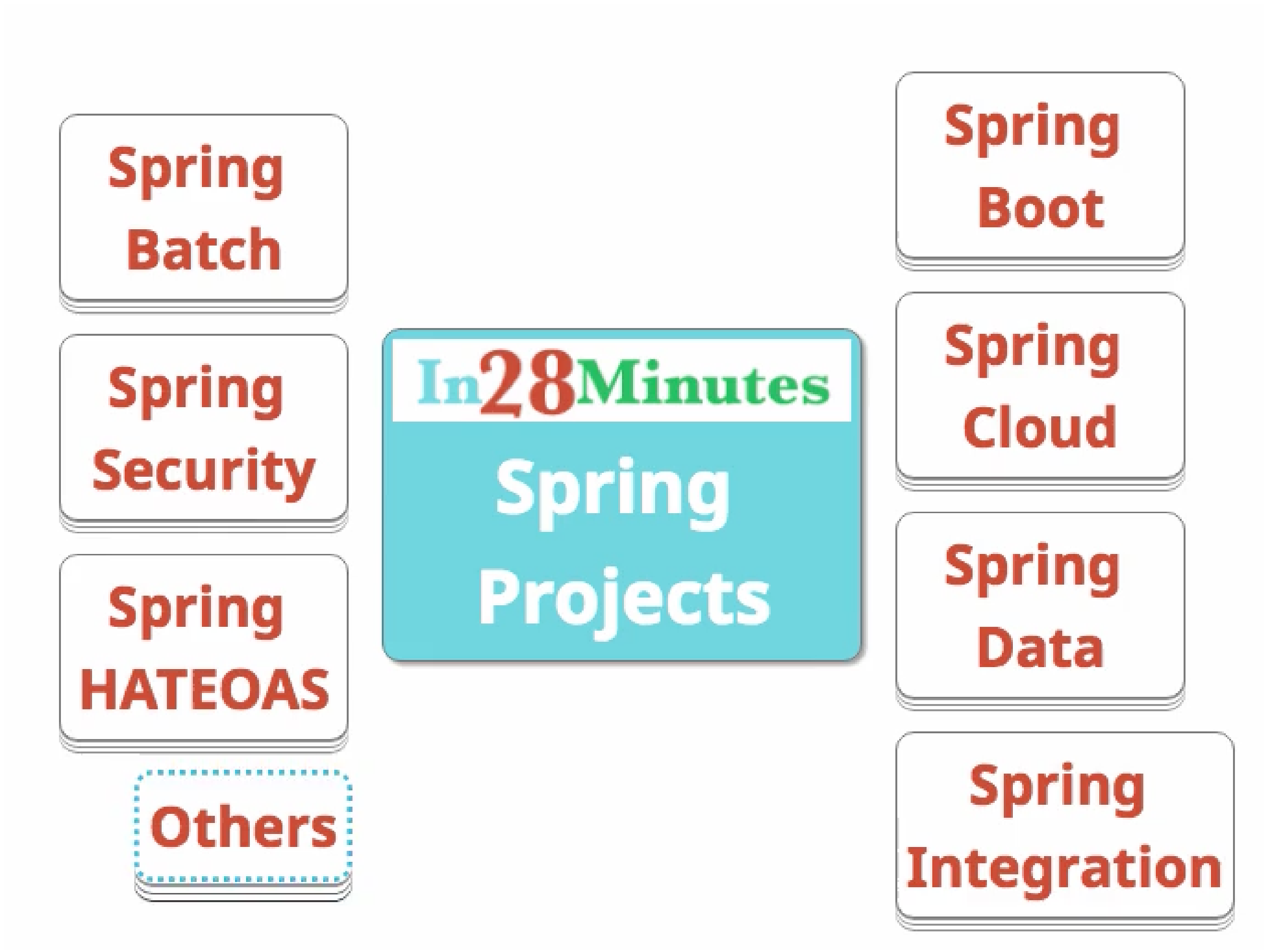


```
@Component
public class BinarySearchImpl {

    @Autowired
    private SortAlgorithm sortAlgorithm;

    public void setSortAlgorithm(SortAlgorithm sortAlgorithm) {
        this.sortAlgorithm = sortAlgorithm;
    }
```

Step 8 : Spring Modules
Spring is built in a very modular way and this enables use to use specific modules without using the other modules of Spring.



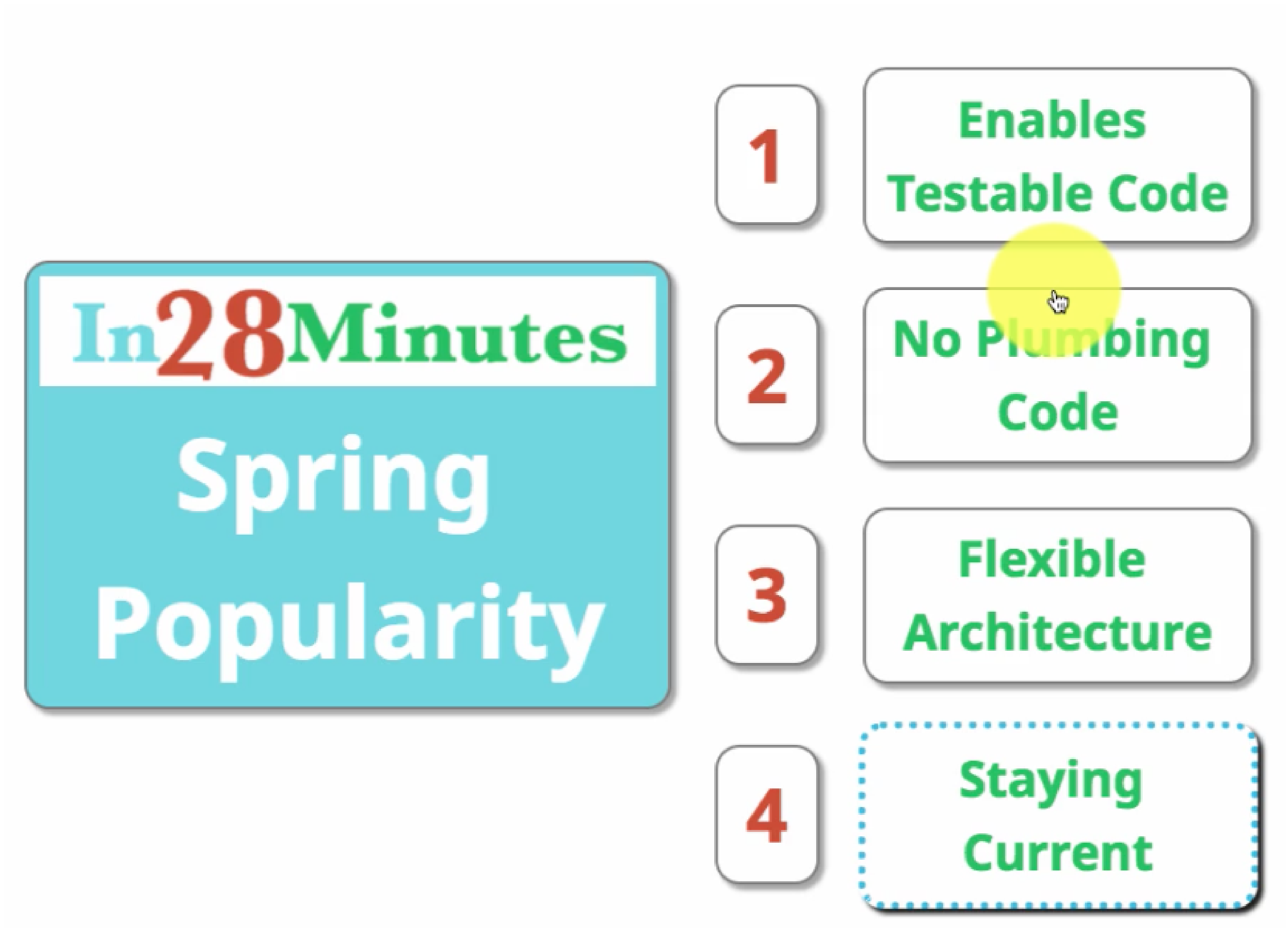
Step 9 : Spring Projects
Spring projects provide solutions for different problems faced by enterprise applications.



What we're looking at are seven of the Spring projects that are just touching the tip of the iceberg. There are a lot of other Spring projects like Spring webservices, Spring session, Spring social, Spring mobile and Spring Android which are solving problems in various different spaces.

Spring has not really restricted itself to just the Spring framework and got involved in a lot of wide variety of projects.

Step 10 : Why is Spring Popular?



Spring is one of the very few frameworks that remains as popular today as it was 15 years back.

Spring Level 2 - Spring in depth

Spring Level 2 - Spring in depth

Title	Category	Github
Spring in Depth	Spring - Level 2	Project Folder on Github

- Step 11 - Dependency Injection - A few more examples
- Step 12 - Autowiring in Depth - by Name and @Primary
- Step 13 - Autowiring in Depth - @Qualifier annotation
- Step 14 - Scope of a Bean - Prototype and Singleton
- Step 15 - Complex scenarios with Scope of a Spring Bean - Mix of Prototype and Singleton
- Step 15B - Difference Between Spring Singleton and GOF Singleton
- Step 16 - Using Component Scan to scan for beans
- Step 17 - Lifecycle of a Bean - @PostConstruct and @PreDestroy
- Step 18 - Container and Dependency Injection (CDI) - @Named, @Inject
- Step 19 - Removing Spring Boot in Basic Application
- Step 20 - Fixing minor stuff - Add Logback and Close Application Context
- Step 21 - Defining Spring Application Context using XML - Part 1
- Step 22 - Defining Spring Application Context using XML - Part 2
- Step 23 - Mixing XML Context with Component Scan for Beans defined with Annotations
- Step 24 - IOC Container vs Application Context vs Bean Factory
- Step 25 - @Component vs @Service vs @Repository vs @Controller
- Step 26 - Read values from external properties file

Step 11 - Dependency Injection - A few more examples
Step 12 - Autowiring in Depth - by Name and @Primary
Step 13 - Autowiring in Depth - @Qualifier annotation

```
@Component
@Scope(ConfigurableBeanFactory.SCOPE_SINGLETON)
public class BinarySearchImpl {

        @Autowired
        @Qualifier("bubble")
        private SortAlgorithm sortAlgorithm;
```

```
@Component
@Qualifier("bubble")
public class BubbleSortAlgorithm implements SortAlgorithm {
```

```
@Component
@Qualifier("quick")
public class QuickSortAlgorithm implements SortAlgorithm {
```

Step 14 - Scope of a Bean - Prototype and Singleton
Step 15 - Complex scenarios with Scope of a Spring Bean - Mix of Prototype and Singleton

```
package com.in28minutes.spring.basics.springin5steps.scope;

import
org.springframework.beans.factory.config.ConfigurableBeanFa
ctory;
import org.springframework.context.annotation.Scope;
import
org.springframework.context.annotation.ScopedProxyMode;
import org.springframework.stereotype.Component;

@Component
@Scope(value=ConfigurableBeanFactory.SCOPE_PROTOTYPE,
                proxyMode = ScopedProxyMode.TARGET_CLASS)
public class JdbcConnection {
        public JdbcConnection() {
                System.out.println("JDBC Connection");
        }
}
```

```
package com.in28minutes.spring.basics.springin5steps.scope;

import
org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

@Component
public class PersonDAO {

        @Autowired
        JdbcConnection jdbcConnection;

        public JdbcConnection getJdbcConnection() {
                return jdbcConnection;
        }

        public void setJdbcConnection(JdbcConnection
jdbcConnection) {
                this.jdbcConnection = jdbcConnection;
        }
}
```

```
package com.in28minutes.spring.basics.springin5steps;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.boot.SpringApplication;
import
org.springframework.boot.autoconfigure.SpringBootApplicatio
n;
import org.springframework.context.ApplicationContext;

import
com.in28minutes.spring.basics.springin5steps.scope.PersonDA
O;

@SpringBootApplication
public class SpringIn5StepsScopeApplication
```

```
 {

        private static Logger LOGGER =

LoggerFactory.getLogger(SpringIn5StepsScopeApplication.clas
s);

        public static void main(String[] args) {

                ApplicationContext applicationContext =

SpringApplication.run(SpringIn5StepsScopeApplication.class,
args);

                PersonDAO personDao =

applicationContext.getBean(PersonDAO.class);

                PersonDAO personDao2 =

applicationContext.getBean(PersonDAO.class);

                LOGGER.info("{}", personDao);
                LOGGER.info("{}",
personDao.getJdbcConnection());

                LOGGER.info("{}", personDao2);
                LOGGER.info("{}",
personDao.getJdbcConnection());

        }
}
```

Step 15B - Difference Between Spring Singleton and GOF Singleton

Step 16 - Using Component Scan to scan for beans

```
package com.in28minutes.spring.basics.componentscan;

import
```

```
 org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

@Component
public class ComponentDAO {

        @Autowired
        ComponentJdbcConnection jdbcConnection;

        public ComponentJdbcConnection getJdbcConnection()
{
                return jdbcConnection;
        }

        public void
setComponentJdbcConnection(ComponentJdbcConnection
jdbcConnection) {
                this.jdbcConnection = jdbcConnection;
        }
}
```

```
package com.in28minutes.spring.basics.componentscan;

import
org.springframework.beans.factory.config.ConfigurableBeanFa
ctory;
import org.springframework.context.annotation.Scope;
import
org.springframework.context.annotation.ScopedProxyMode;
import org.springframework.stereotype.Component;

@Component
@Scope(value=ConfigurableBeanFactory.SCOPE_PROTOTYPE,
                proxyMode = ScopedProxyMode.TARGET_CLASS)
public class ComponentJdbcConnection {
        public ComponentJdbcConnection() {
                System.out.println("JDBC Connection");
        }
```

```
}

package com.in28minutes.spring.basics.springin5steps;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.boot.SpringApplication;
import
org.springframework.boot.autoconfigure.SpringBootApplicatio
n;
import org.springframework.context.ApplicationContext;
import
org.springframework.context.annotation.ComponentScan;

import
com.in28minutes.spring.basics.componentscan.ComponentDAO;

@SpringBootApplication
@ComponentScan("com.in28minutes.spring.basics.componentscan
")
public class SpringIn5StepsComponentScanApplication {

        private static Logger LOGGER =

LoggerFactory.getLogger(SpringIn5StepsComponentScanApplicat
ion.class);

        public static void main(String[] args) {

                ApplicationContext applicationContext =

SpringApplication.run(SpringIn5StepsComponentScanApplicatio
n.class, args);

                ComponentDAO componentDAO =

applicationContext.getBean(ComponentDAO.class);

                LOGGER.info("{}",
```

```
 componentDAO);

        }
}
```

Step 17 - Lifecycle of a Bean - @PostConstruct and @PreDestroy
BinarySearchImpl.java

```
        @PostConstruct
        public void postConstruct() {
                logger.info("postConstruct");
        }

        @PreDestroy
        public void preDestroy() {
                logger.info("preDestroy");
        }
```

Step 18 - Container and Dependency Injection (CDI) - @Named, @Inject
/pom.xml

```
<dependency>
        <groupId>javax.inject</groupId>
        <artifactId>javax.inject</artifactId>
        <version>1</version>
</dependency>
```

```
package com.in28minutes.spring.basics.springin5steps;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.boot.SpringApplication;
import
org.springframework.boot.autoconfigure.SpringBootApplicatio
n;
import org.springframework.context.ApplicationContext;

import
```

```
 com.in28minutes.spring.basics.springin5steps.cdi.SomeCdiBu
siness;

@SpringBootApplication
public class SpringIn5StepsCdiApplication {

        private static Logger LOGGER =

LoggerFactory.getLogger(SpringIn5StepsCdiApplication.class)
;

        public static void main(String[] args) {

                ApplicationContext applicationContext =

SpringApplication.run(SpringIn5StepsCdiApplication.class,
args);

                SomeCdiBusiness business =

applicationContext.getBean(SomeCdiBusiness.class);

                LOGGER.info("{} dao-{}", business,
business.getSomeCDIDAO());
        }
}
```

```
package com.in28minutes.spring.basics.springin5steps.cdi;

import javax.inject.Inject;
import javax.inject.Named;

@Named
public class SomeCdiBusiness {

        @Inject
        SomeCdiDao someCdiDao;

        public SomeCdiDao getSomeCDIDAO() {
                return someCdiDao;
        }

        public void setSomeCDIDAO(SomeCdiDao someCdiDao) {
                this.someCdiDao = someCdiDao;
        }
}
```

```
package com.in28minutes.spring.basics.springin5steps.cdi;

import javax.inject.Named;

@Named
public class SomeCdiDao {

}
```

Step 19 - Removing Spring Boot in Basic Application

pom.xml

```
<dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-core</artifactId>
</dependency>
<dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
</dependency>
<dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-api</artifactId>
</dependency>
<dependency>
        <groupId>ch.qos.logback</groupId>
        <artifactId>logback-classic</artifactId>
</dependency>
```

```
package com.in28minutes.spring.basics.springin5steps;

import org.springframework.context.ApplicationContext;
import
org.springframework.context.annotation.AnnotationConfigAppl
icationContext;
import
org.springframework.context.annotation.ComponentScan;
import
org.springframework.context.annotation.Configuration;

import
com.in28minutes.spring.basics.springin5steps.basic.BinarySe
archImpl;

@Configuration
@ComponentScan
public class SpringIn5StepsBasicApplication {

        public static void main(String[] args) {

                ApplicationContext applicationContext =
                                new
AnnotationConfigApplicationContext(SpringIn5StepsBasicAppli
cation.class);
```

Step 20 - Fixing minor stuff - Add Logback and Close Application Context

```
<dependency>
        <groupId>ch.qos.logback</groupId>
        <artifactId>logback-classic</artifactId>
</dependency>
```

```
@Configuration
@ComponentScan
public class SpringIn5StepsBasicApplication {

        public static void main(String[] args)
```

```
 {

                try (AnnotationConfigApplicationContext
applicationContext =
                                new
AnnotationConfigApplicationContext(

SpringIn5StepsBasicApplication.class)) {
                        //No change in code
                }
        }
}
```

Same changes in

- SpringIn5StepsCdiApplication
- SpringIn5StepsComponentScanApplication
- SpringIn5StepsScopeApplication

Step 21 - Defining Spring Application Context using XML - Part 1
Step 22 - Defining Spring Application Context using XML - Part 2

```
package com.in28minutes.spring.basics.springin5steps;

import
org.springframework.context.annotation.ComponentScan;
import
org.springframework.context.annotation.Configuration;
import
org.springframework.context.support.ClassPathXmlApplication
Context;

import
com.in28minutes.spring.basics.springin5steps.xml.XmlPersonD
AO;

@Configuration
@ComponentScan
public class SpringIn5StepsXMLContextApplication {

	public static void main(String[] args) {

		try (ClassPathXmlApplicationContext
applicationContext = new ClassPathXmlApplicationContext(
				"applicationContext.xml"))
{

			XmlPersonDAO personDao =
applicationContext.getBean(XmlPersonDAO.class);
			System.out.println(personDao);

System.out.println(personDao.getXmlJdbcConnection());
		}
	}
}
```

```
package com.in28minutes.spring.basics.springin5steps.xml;

public class XmlJdbcConnection {
	public XmlJdbcConnection() {
		System.out.println("JDBC Connection");
	}
}
```

```
package com.in28minutes.spring.basics.springin5steps.xml;

public class XmlPersonDAO {

	XmlJdbcConnection xmlJdbcConnection;

	public XmlJdbcConnection getXmlJdbcConnection() {
		return xmlJdbcConnection;
	}

	public void setXmlJdbcConnection(XmlJdbcConnection
jdbcConnection) {
		this.xmlJdbcConnection = jdbcConnection;
```

```
        }
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/b
eans
        http://www.springframework.org/schema/beans/spring-
beans.xsd">

    <bean id="xmlJdbcConnection"

class="com.in28minutes.spring.basics.springin5steps.xml.Xml
JdbcConnection">
    </bean>

    <bean id="xmlPersonDAO"
class="com.in28minutes.spring.basics.springin5steps.xml.Xml
PersonDAO">
                <property name="xmlJdbcConnection"
ref="xmlJdbcConnection"/>
    </bean>

</beans>
```

Step 23 - Mixing XML Context with Component Scan for Beans defined with Annotations

```
public class SpringIn5StepsXMLContextApplication {

        private static Logger LOGGER =
LoggerFactory.getLogger(SpringIn5StepsScopeApplication.clas
s);

        public static void main(String[] args) {

                try (ClassPathXmlApplicationContext
applicationContext = new ClassPathXmlApplicationContext(
                                "applicationContext.xml"))
```

```
 {

                        LOGGER.info("Beans Loaded -> {}",
(Object) applicationContext.getBeanDefinitionNames());
                        // [xmlJdbcConnection,
xmlPersonDAO]

        <context:component-scan base-
package="com.in28minutes.spring.basics"/>
```

Step 24 - IOC Container vs Application Context vs Bean Factory

Step 25 - @Component vs @Service vs @Repository vs @Controller

```
@Repository
public class ComponentDAO {


@Service
@Scope(ConfigurableBeanFactory.SCOPE_SINGLETON)
public class BinarySearchImpl {

@Service
@Qualifier("bubble")
public class BubbleSortAlgorithm implements SortAlgorithm {

@Service
@Qualifier("quick")
public class QuickSortAlgorithm implements SortAlgorithm {

@Repository
public class PersonDAO {
```

Step 26 - Read values from external properties file

```
package com.in28minutes.spring.basics.springin5steps;


import
org.springframework.context.annotation.AnnotationConfigAppl
icationContext;
import
org.springframework.context.annotation.ComponentScan;
```

```
import
org.springframework.context.annotation.Configuration;
import
org.springframework.context.annotation.PropertySource;

import
com.in28minutes.spring.basics.springin5steps.properties.Som
eExternalService;

@Configuration
@ComponentScan
//
@PropertySource("classpath:app.properties")
public class SpringIn5StepsPropertiesApplication {

        public static void main(String[] args) {

                try (AnnotationConfigApplicationContext
applicationContext = new
AnnotationConfigApplicationContext(

SpringIn5StepsPropertiesApplication.class)) {

                        SomeExternalService service =
applicationContext.getBean(SomeExternalService.class);

System.out.println(service.returnServiceURL());
                }
        }
}
```

```
package
com.in28minutes.spring.basics.springin5steps.properties;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;

@Component
```

```
public class SomeExternalService {

        @Value("${external.service.url}")
        private String url;

        public String returnServiceURL(){
                return url;
        }

}
```

/src/main/resources/app.properties

```
external.service.url=http://someserver.dev.com/service
```

Spring Level 3 - Unit Testing with Spring Framework

Title	Category	Github
Unit Testing with Spring Framework	Spring - Level 3	Project Folder on Github

- Step 27 - Spring Unit Testing with a Java Context
- Step 28 - Spring Unit Testing with an XML Context
- Step 29 - Spring Unit Testing with Mockito

Step 27 - Spring Unit Testing with a Java Context

```
<dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-test</artifactId>
</dependency>
<dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
</dependency>

@RunWith(SpringRunner.class)
//@SpringBootTest
public class SpringIn5StepsBasicApplicationTests {

package com.in28minutes.spring.basics.springin5steps.basic;

import static org.junit.Assert.assertEquals;

import org.junit.Test;
import org.junit.runner.RunWith;
import
org.springframework.beans.factory.annotation.Autowired;
```

```
import
org.springframework.test.context.ContextConfiguration;
import
org.springframework.test.context.junit4.SpringRunner;

import
com.in28minutes.spring.basics.springin5steps.SpringIn5Steps
BasicApplication;

//Load the context
@RunWith(SpringRunner.class)
@ContextConfiguration(classes =
SpringIn5StepsBasicApplication.class)
public class BinarySearchTest {

        // Get this bean from the context
        @Autowired
        BinarySearchImpl binarySearch;

        @Test
        public void testBasicScenario() {

                // call method on binarySearch
                int actualResult =
binarySearch.binarySearch(new int[] {}, 5);

                // check if the value is correct
                assertEquals(3, actualResult);

        }

}
```

Step 28 - Spring Unit Testing with an XML Context
/src/test/resources/testContext.xml

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<beans  xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:context="http://www.springframework.org/schema/contex
t"

xsi:schemaLocation="http://www.springframework.org/schema/b
eans
        http://www.springframework.org/schema/beans/spring-
beans.xsd
        http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-
context.xsd">

        <import
resource="classpath:applicationContext.xml"/>

</beans>
```

```
package com.in28minutes.spring.basics.springin5steps.basic;

import static org.junit.Assert.assertEquals;

import org.junit.Test;
import org.junit.runner.RunWith;
import
org.springframework.beans.factory.annotation.Autowired;
import
org.springframework.test.context.ContextConfiguration;
import
org.springframework.test.context.junit4.SpringRunner;

//Load the context
@RunWith(SpringRunner.class)
@ContextConfiguration(locations="/testContext.xml")
public class BinarySearchXMLConfigurationTest {

        // Get this bean from the context
```

```
        @Autowired
        BinarySearchImpl binarySearch;

        @Test
        public void testBasicScenario() {

                // call method on binarySearch
                int actualResult =
binarySearch.binarySearch(new int[] {}, 5);

                // check if the value is correct
                assertEquals(3, actualResult);

        }

}
```

Step 29 - Spring Unit Testing with Mockito

```
public class SomeCdiBusiness {

        // SAME OLD CODE

        public int findGreatest() {
                int greatest = Integer.MIN_VALUE;
                int[] data = someCdiDao.getData();
                for (int value : data) {
                        if (value > greatest) {
                                greatest = value;
                        }
                }
                return greatest;
        }

}
```

Add a new method

```
package com.in28minutes.spring.basics.springin5steps.cdi;

import javax.inject.Named;

@Named
public class SomeCdiDao {

        public int[] getData() {
                return new int[] {5, 89,100};
        }

}
```

```
package com.in28minutes.spring.basics.springin5steps.cdi;

import static org.junit.Assert.assertEquals;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.mockito.Mockito;
import org.mockito.junit.MockitoJUnitRunner;

@RunWith(MockitoJUnitRunner.class)
public class SomeCdiBusinessTest {

        // Inject Mock
        @InjectMocks
        SomeCdiBusiness business;

        // Create Mock
        @Mock
        SomeCdiDao daoMock;

        @Test
        public void testBasicScenario() {

Mockito.when(daoMock.getData()).thenReturn(new int[] { 2, 4
});
                assertEquals(4, business.findGreatest());
        }

        @Test
        public void testBasicScenario_NoElements() {

Mockito.when(daoMock.getData()).thenReturn(new int[] { });
                assertEquals(Integer.MIN_VALUE,
business.findGreatest());
        }

        @Test
        public void testBasicScenario_EqualElements() {

Mockito.when(daoMock.getData()).thenReturn(new int[] {
2,2});
                assertEquals(2, business.findGreatest());
        }

}

<dependency>
        <groupId>org.mockito</groupId>
        <artifactId>mockito-core</artifactId>
</dependency>
```

Spring Level 4 - Introduction To Spring Boot

Spring Level 4 - Introduction To Spring Boot

Title	Category	Github
Spring Boot in 10 Steps	Spring - Level 4	Project Folder on Github

- Step 1 : Introduction to Spring Boot - Goals and Important Features
- Step 2 : Developing Spring Applications before Spring Boot
- Step 3 : Using Spring Initializr to create a Spring Boot Application
- Step 4 : Creating a Simple REST Controller
- Step 5 : What is Spring Boot Auto Configuration?
- Step 6 : Spring Boot vs Spring vs Spring MVC
- Step 7 : Spring Boot Starter Projects - Starter Web and Starter JPA
- Step 8 : Overview of different Spring Boot Starter Projects
- Step 9 : Spring Boot Actuator
- Step 10 : Spring Boot Developer Tools

Step 1 : Introduction to Spring Boot - Goals and Important Features
Goals

- Enable building production ready applications quickly
- Provide common non-functional features
  - embedded servers
  - metrics
  - health checks
  - externalized configuration

What Spring Boot is NOT!

- ZERO code generation

- Neither an application server nor a web server

Features

- Quick Starter Projects with Auto Configuration
  - Web
  - JPA

- Embedded Servers
  - Tomcat, Jetty or Undertow

- Production-ready features
  - metrics and health checks
  - externalized configuration

Step 2 : Developing Spring Applications before Spring Boot

> *Recommended Reading - http://www.springboottutorial.com/spring-boot-vs-spring-mvc-vs-spring*

Step 3 : Using Spring Initializr to create a Spring Boot Application

> *https://start.spring.io*

Step 4 : Creating a Simple REST Controller
/src/main/java/com/in28minutes/springboot/basics/springbootin10steps/BooksController.java

```
package
com.in28minutes.springboot.basics.springbootin10steps;

import java.util.Arrays;
import java.util.List;

import org.springframework.web.bind.annotation.GetMapping;
import
org.springframework.web.bind.annotation.RestController;
```

```
@RestController
public class BooksController {
        @GetMapping("/books")
        public List<Book> getAllBooks() {
                return Arrays.asList(
                                new Book(1l, "Mastering
Spring 5.2", "Ranga Karanam"));
        }
}
```

Step 5 : What is Spring Boot Auto Configuration?

Recommended Reading - http://www.springboottutorial.com/spring-boot-auto-configuration

- Spring based applications have a lot of configuration. When we use Spring MVC, we need to configure component scan, dispatcher servlet, a view resolver, web jars(for delivering static content) among other things. When we use Hibernate/JPA, we would need to configure a datasource, an entity manager factory, a transaction manager among a host of other things. Spring Boot brings in new thought process around this - Can we bring more intelligence into this? When a spring mvc jar is added into an application, can we auto configure some beans automatically?

Step 6 : Spring Boot vs Spring vs Spring MVC

Recommended Reading - http://www.springboottutorial.com/spring-boot-vs-spring-mvc-vs-spring

- Spring is about Dependency Injection. It makes it easy to develop loosely coupled applications. It makes applications testable.
- Spring MVC brings loose coupling to web mvc application development with features like Dispatcher Servlet, View Resolver etc
- Spring Boot eliminates the need for manual configuration with Spring and Spring MVC. You can use Spring and Spring MVC without needing a lot of configuration.
- Spring Boot aims to enable production ready applications in quick time.
    - Actuator : Enables Advanced Monitoring and Tracing of applications.
    - Embedded Server Integrations - Since server is integrated into the application, I

- - would NOT need to have a separate application server installed on the server.
  - Default Error Handling

Step 7 : Spring Boot Starter Projects - Starter Web and Starter JPA

Recommended Reading - http://www.springboottutorial.com/spring-boot-starter-projects

- Starters are a set of convenient dependency descriptors that you can include in your application. You get a one-stop-shop for all the Spring and related technology that you need, without having to hunt through sample code and copy paste loads of dependency descriptors. For example, if you want to get started using Spring and JPA for database access, just include the spring-boot-starter-data-jpa dependency in your project, and you are good to go.

Step 8 : Overview of different Spring Boot Starter Projects
As we see from Spring Boot Starter Web, starter projects help us in quickly getting started with developing specific types of applications.

Examples

- spring-boot-starter-web-services - SOAP Web Services
- spring-boot-starter-web - Web & RESTful applications
- spring-boot-starter-test - Unit testing and Integration Testing
- spring-boot-starter-jdbc - Traditional JDBC
- spring-boot-starter-hateoas - Add HATEOAS features to your services
- spring-boot-starter-security - Authentication and Authorization using Spring Security
- spring-boot-starter-data-jpa - Spring Data JPA with Hibernate
- spring-boot-starter-cache - Enabling Spring Framework's caching support
- spring-boot-starter-data-rest - Expose Simple REST Services using Spring Data REST
- spring-boot-starter-actuator - To use advanced features like monitoring & tracing to your application out of the box
- spring-boot-starter-undertow, spring-boot-starter-jetty, spring-boot-starter-tomcat - To pick your specific choice of Embedded Servlet Container
- spring-boot-starter-logging - For Logging using logback

- spring-boot-starter-log4j2 - Logging using Log4j2

Step 9 : Spring Boot Actuator
Spring Boot starter actuator actually exposes a lot of REST services and these services are compliant with the standard called HAL standard. And we would use a hal browser so that we can browse through the data which is provided by these services.

Spring Boot Actuator exposes a lot of data

> *application info, metrics, dump, beans, env, config properties, audit events, heap dump, loggers, trace, health mappings and auto config.*

Actuator provides more metadata about your application.

Step 10 : Spring Boot Developer Tools

```
- Why do you need to restart your server for every java and
jsp change?
- Spring Boot Developer Tools enables dynamic reloading of
modified changes.
```

Spring Level 5 - Spring AOP

Spring Level 5 - Spring AOP

Title	Category	Github
Spring AOP	Spring - Level 5	Project Folder on Github

- Step 01 - Setting up AOP Example - Part 1
- Step 02 - Setting up AOP Example - Part 2
- Step 03 - Defining an @Before advice
- Step 04 - Understand AOP Terminology - Pointcut, Advice, Aspect, Join Point, Weaving and Weaver
- Step 05 - Using @After, @AfterReturning, @AfterThrowing advices
- Step 06 - Using @Around advice to implement performance tracing
- Step 07 - Best Practice : Use common Pointcut Configuration
- Step 08 - Quick summary of other Pointcuts
- Step 09 - Creating Custom Annotation and an Aspect for Tracking Time

Step 01 - Setting up AOP Example - Part 1
Creating a Spring AOP Project with Spring Initializr is a cake walk.

> *Spring Initializr http://start.spring.io/ is great tool to bootstrap your Spring Boot projects.*

Notes

- Launch Spring Initializr and choose the following
  - Choose `com.in28minutes.spring.aop`

- - as Group
  - Choose `spring-aop` as Artifact
  - Choose the following Dependencies
    - AOP

- Click Generate Project.
- Import the project into Eclipse.
- If you want to understand all the files that are part of this project, you can go here.

Step 02 - Setting up AOP Example - Part 2

```
package com.in28minutes.spring.aop.springaop;

import
org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service
public class Business1 {

        private Logger logger =
LoggerFactory.getLogger(this.getClass());

        @Autowired
        private Dao1 dao1;

        public String calculateSomething(){
                String value = dao1.retrieveSomething();
                logger.info("In Business - {}", value);
                return value;
        }
}
```

```
package com.in28minutes.spring.aop.springaop;

import
org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
```

```
@Service
public class Business2 {

        @Autowired
        private Dao2 dao2;

        public String calculateSomething(){
                //Business Logic
                return dao2.retrieveSomething();
        }
}
```

```
package com.in28minutes.spring.aop.springaop;

import org.springframework.stereotype.Repository;

@Repository
public class Dao1 {

        public String retrieveSomething(){
                return "Dao1";
        }

}
```

```
package com.in28minutes.spring.aop.springaop;

import org.springframework.stereotype.Repository;

@Repository
public class Dao2 {

        public String retrieveSomething(){
                return "Dao2";
        }

}
```

Step 03 - Defining an @Before advice

```
public class SpringAopApplication implements
CommandLineRunner {

        @Autowired
        private Business1 business1;

        @Autowired
        private Business2 business2;

        @Override
        public void run(String... args) throws Exception {

logger.info(business1.calculateSomething());

logger.info(business2.calculateSomething());


package com.in28minutes.spring.aop.springaop.aspect;

import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import
org.springframework.context.annotation.Configuration;

//AOP
//Configuration
@Aspect
@Configuration
public class UseAccessAspect {

        private Logger logger =
 LoggerFactory.getLogger(this.getClass());

        //What kind of method calls I would intercept
        //execution(* PACKAGE.*.*(..))

        @Before("execution(*
com.in28minutes.spring.aop.springaop.business.*.*(..))")
        public void before(JoinPoint joinPoint){
                logger.info(" Check for user access ");
                logger.info(" Allowed execution for {}",
joinPoint);
        }
}
```

Step 04 - Understand AOP Terminology - Pointcut, Advice, Aspect, Join Point, Weaving and Weaver
Step 05 - Using @After, @AfterReturning, @AfterThrowing advices
Step 06 - Using @Around advice to implement performance tracing

```
package com.in28minutes.spring.aop.springaop.aspect;

import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.annotation.After;
import org.aspectj.lang.annotation.AfterReturning;
import org.aspectj.lang.annotation.AfterThrowing;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import
org.springframework.context.annotation.Configuration;

//AOP
//Configuration
@Aspect
@Configuration
public class AfterAopAspect {
        private Logger logger =
LoggerFactory.getLogger(this.getClass());

        @AfterReturning(value = "execution(*
com.in28minutes.spring.aop.springaop.business.*.*(..))",
                        returning = "result")
        public void afterReturning(JoinPoint joinPoint,
Object result) {
                logger.info("{} returned with value {}",
joinPoint, result);
        }

        @After(value = "execution(*
com.in28minutes.spring.aop.springaop.business.*.*(..))")
        public void after(JoinPoint joinPoint) {
                logger.info("after execution of {}",
joinPoint);
        }
}
```

```
package com.in28minutes.spring.aop.springaop.aspect;

import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import
org.springframework.context.annotation.Configuration;

@Aspect
@Configuration
public class MethodExecutionCalculationAspect {

        private Logger logger =
LoggerFactory.getLogger(this.getClass());
```

```
        @Around("execution(*
com.in28minutes.spring.aop.springaop.business.*.*(..))")
        public void around(ProceedingJoinPoint joinPoint)
throws Throwable {
                long startTime =
System.currentTimeMillis();

                joinPoint.proceed();

                long timeTaken = System.currentTimeMillis()
- startTime;
                logger.info("Time Taken by {} is {}",
joinPoint, timeTaken);
        }
}
```

```
package com.in28minutes.spring.aop.springaop.aspect;

import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import
org.springframework.context.annotation.Configuration;

//AOP
//Configuration
@Aspect
@Configuration
public class UserAccessAspect {

        private Logger logger =
LoggerFactory.getLogger(this.getClass());

        //What kind of method calls I would intercept
        //execution(* PACKAGE.*.*(..))
        //Weaving & Weaver
        @Before("execution(*
com.in28minutes.spring.aop.springaop.data.*.*(..))")
        public void before(JoinPoint joinPoint){
                //Advice
                logger.info(" Check for user access ");
                logger.info(" Allowed execution for {}",
joinPoint);
        }
}
```

Step 07 - Best Practice : Use common Pointcut Configuration

```
package com.in28minutes.spring.aop.springaop.aspect;

import org.aspectj.lang.annotation.Pointcut;

public class CommonJoinPointConfig {

        @Pointcut("execution(*
com.in28minutes.spring.aop.springaop.data.*.*(..))")
        public void dataLayerExecution(){}

        @Pointcut("execution(*
com.in28minutes.spring.aop.springaop.business.*.*(..))")
        public void businessLayerExecution(){}


}

public class MethodExecutionCalculationAspect {

        private Logger logger =
LoggerFactory.getLogger(this.getClass());

        @Around("com.in28minutes.spring.aop.springaop.aspec
t.CommonJoinPointConfig.businessLayerExecution()")
```

public class AfterAopAspect

```
        @AfterReturning(value =
"com.in28minutes.spring.aop.springaop.aspect.CommonJoinPoin
tConfig.businessLayerExecution()", returning = "result")
        @After(value =
"com.in28minutes.spring.aop.springaop.aspect.CommonJoinPoin
tConfig.businessLayerExecution()")
```

```
public class UserAccessAspect {

        @Before("com.in28minutes.spring.aop.springaop.aspec
t.CommonJoinPointConfig.dataLayerExecution()")
```

Step 08 - Quick summary of other Pointcuts

Step 09 - Creating Custom Annotation and an Aspect for Tracking Time

```
package com.in28minutes.spring.aop.springaop.aspect;

import org.aspectj.lang.annotation.Pointcut;

public class CommonJoinPointConfig {

        @Pointcut("execution(*
com.in28minutes.spring.aop.springaop.data.*.*(..))")
        public void dataLayerExecution(){}

        @Pointcut("execution(*
com.in28minutes.spring.aop.springaop.business.*.*(..))")
        public void businessLayerExecution(){}

        @Pointcut("dataLayerExecution() &&
businessLayerExecution()")
        public void allLayerExecution(){}

        @Pointcut("bean(*dao*)")
        public void beanContainingDao(){}
        @Pointcut("within(com.in28minutes.spring.aop.spring
aop.data..*)")
        public void dataLayerExecutionWithWithin(){}

        @Pointcut("@annotation(com.in28minutes.spring.aop.s
pringaop.aspect.TrackTime)")
        public void trackTimeAnnotation(){}

}
```

```
package com.in28minutes.spring.aop.springaop.aspect;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
public @interface TrackTime {

}
```

```
@Aspect
@Configuration
public class MethodExecutionCalculationAspect {

        @Around("com.in28minutes.spring.aop.springaop.aspec
t.CommonJoinPointConfig.trackTimeAnnotation()")
        public void around(ProceedingJoinPoint joinPoint)
throws Throwable {
```

```
public class Business1 {

        @TrackTime
        public String calculateSomething(){

```

```
@Repository
public class Dao1 {

        @TrackTime
        public String retrieveSomething(){
```

Spring Level 6 - Spring JDBC and JPA

Spring Level 6 - Spring JDBC and JPA

Title	Category	Github
Spring JDBC and JPA	Spring - Level 6	Project Folder on Github

- Step 01 - Setting up a project with JDBC, JPA, H2 and Web Dependencies
- Step 02 - Launching up H2 Console
- Step 03 - Creating a Database Table in H2
- Step 04 - Populate data into Person Table
- Step 05 - Implement findAll persons Spring JDBC Query Method
- Step 06 - Execute the findAll method using CommandLineRunner
- Step 07 - A Quick Review - JDBC vs Spring JDBC
- Step 08 - Whats in the background? Understanding Spring Boot Autoconfiguration
- Step 09 - Implementing findById Spring JDBC Query Method
- Step 10 - Implementing deleteById Spring JDBC Update Method
- Step 11 - Implementing insert and update Spring JDBC Update Methods
- Step 12 - Creating a custom Spring JDBC RowMapper
- Step 13 - Quick introduction to JPA
- Step 14 - Defining Person Entity
- Step 15 - Implementing findById JPA Repository Method
- Step 16 - Implementing insert and update JPA Repository Methods
- Step 17 - Implementing deleteById JPA Repository Method
- Step 18 - Implementing findAll using JPQL Named Query
- Step 19 - Introduction to Spring Data JPA
- Step 20 - Connecting to Other Databases

Step 01 - Setting up a project with JDBC, JPA, H2 and Web Dependencies
Creating a Spring JDBC Project with Spring Initializr is a cake walk.

> *Spring Initializr http://start.spring.io/ is great tool to bootstrap your Spring Boot projects.*

Notes

- Launch Spring Initializr and choose the following
  - Choose `com.in28minutes.database` as Group
  - Choose `database-demo` as Artifact
  - Choose the following Dependencies
    - Web
    - JDBC
    - JPA
    - H2
- Click Generate Project.
- Import the project into Eclipse.
- If you want to understand all the files that are part of this project, you can go here.

Step 02 - Launching up H2 Console
/src/main/resources/application.properties

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

Launching H2

- URL - http://localhost:8080/h2-console
- Make sure to check the db url - jdbc:h2:mem:testdb

Step 03 - Creating a Database Table in H2
/src/main/resources/data.sql

```
create table person
(
   id integer not
```

```
 null,
   name varchar(255) not null,
   location varchar(255),
   birth_date timestamp,
   primary key(id)
);
```

Step 04 - Populate data into Person Table

Step 05 - Implement findAll persons Spring JDBC Query Method

```
package com.in28minutes.database.databasedemo.entity;

import java.util.Date;

public class Person {
        private int id;
        private String name;
        private String location;
        private Date birthDate;

        public Person(int id, String name, String location,
Date birthDate) {
                super();
                this.id = id;
                this.name = name;
                this.location = location;
                this.birthDate = birthDate;
        }

        public int getId() {
                return id;
        }

        public void setId(int id) {
                this.id = id;
        }

        public String getName() {
                return name;
        }

        public void setName(String name) {
                this.name = name;
        }

        public String getLocation() {
                return location;
        }

        public void setLocation(String location) {
                this.location = location;
        }

        public Date getBirthDate() {
                return birthDate;
        }

        public void setBirthDate(Date birthDate) {
                this.birthDate = birthDate;
        }

}
```

/src/main/java/com/in28minutes/database/databasedemo/jdbc/PersonJbdcDao.java

```
package com.in28minutes.database.databasedemo.jdbc;

import java.util.List;

import
org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.BeanPropertyRowMapper;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Repository;
import com.in28minutes.database.databasedemo.entity.Person;

@Repository
public class PersonJbdcDao {
        @Autowired
        JdbcTemplate jdbcTemplate;

        public List<Person> findAll() {
                return jdbcTemplate.query("select * from
person",
                                new
BeanPropertyRowMapper(Person.class));
        }
}
```

Add insert statements into data.sql /src/main/resources/data.sql

```
INSERT INTO PERSON (ID, NAME, LOCATION, BIRTH_DATE )
VALUES(10001,  'Ranga', 'Hyderabad',sysdate());
INSERT INTO PERSON (ID, NAME, LOCATION, BIRTH_DATE )
VALUES(10002,  'James', 'New York',sysdate());
INSERT INTO PERSON (ID, NAME, LOCATION, BIRTH_DATE )
VALUES(10003,  'Pieter', 'Amsterdam',sysdate());
```

Step 06 - Execute the findAll method using CommandLineRunner

```
public class DatabaseDemoApplication implements
CommandLineRunner {

        private Logger logger =
LoggerFactory.getLogger(this.getClass());

        @Autowired
        PersonJbdcDao dao;

        @Override
        public void run(String... args) throws Exception {
                logger.info("All users -> {}",
 dao.findAll());
```
Modified

```
@Repository
public class PersonJbdcDao {
        @Autowired
        JdbcTemplate jdbcTemplate;

        public List<Person> findAll() {
                return jdbcTemplate.query("select * from 
person",
                                new 
BeanPropertyRowMapper<Person>(Person.class));
        }
}
```
Step 07 - A Quick Review - JDBC vs Spring JDBC
Step 08 - Whats in the background? Understanding Spring Boot Autoconfiguration
Step 09 - Implementing findById Spring JDBC Query Method
Step 10 - Implementing deleteById Spring JDBC Update Method
Modified

```
package com.in28minutes.database.databasedemo.jdbc;

import java.util.List;

import 
org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.BeanPropertyRowMapper;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Repository;

import com.in28minutes.database.databasedemo.entity.Person;

@Repository
public class PersonJbdcDao {
        @Autowired
        JdbcTemplate jdbcTemplate;

        public List<Person> findAll() {
                return jdbcTemplate.query("select * from
person", new BeanPropertyRowMapper<Person>(Person.class));
        }

        public Person findById(int id) {
                return jdbcTemplate.queryForObject
                                ("select * from person
where id=?", new Object[] { id },
                                new
BeanPropertyRowMapper<Person>(Person.class));
        }

        public int deleteById(int id) {
                return jdbcTemplate.update
                                ("delete from person where
id=?", new Object[] { id });
        }


}
```

DatabaseDemoApplication

```
                logger.info("User id 10001 -> {}",
dao.findById(10001));
                logger.info("Deleting 10002 -> No of Rows
Deleted - {}", dao.deleteById(10002));
```

Step 11 - Implementing insert and update Spring JDBC Update Methods

```
        public int deleteById(int id) {
                return jdbcTemplate.update("delete from
person where id=?", new Object[] { id });
```

```
        }

        public int insert(Person person) {
                return jdbcTemplate.update("insert into
person (id, name, location, birth_date) " + "values(?,  ?,
?, ?)",
                                new Object[] {
person.getId(), person.getName(), person.getLocation(),
                                                new
Timestamp(person.getBirthDate().getTime()) });
        }

        public int update(Person person) {
                return jdbcTemplate.update("update person "
+ " set name = ?, location = ?, birth_date = ? " + " where
id = ?",
                                new Object[] {
person.getName(), person.getLocation(), new
Timestamp(person.getBirthDate().getTime()),

person.getId() });
        }

        logger.info("Deleting 10002 -> No of Rows Deleted -
{}",
                                dao.deleteById(10002));

                logger.info("Inserting 10004 -> {}",
                                dao.insert(new
Person(10004, "Tara", "Berlin", new Date())));

                logger.info("Update 10003 -> {}",
                                dao.update(new
Person(10003, "Pieter", "Utrecht", new Date())));
```

Step 12 - Creating a custom Spring JDBC RowMapper

Inner class in PersonJbdcDao

```
class PersonRowMapper implements RowMapper<Person>{
        @Override
        public Person mapRow(ResultSet rs, int rowNum)
throws SQLException {
                Person person = new Person();
                person.setId(rs.getInt("id"));
                person.setName(rs.getString("name"));

person.setLocation(rs.getString("location"));

person.setBirthDate(rs.getTimestamp("birth_date"));
                return person;
        }

}

public List<Person> findAll() {
        return jdbcTemplate.query("select * from person",
new PersonRowMapper());
}
```

PersonJbdcDao

Step 13 - Quick introduction to JPA

Step 14 - Defining Person Entity

```
package com.in28minutes.database.databasedemo.entity;

import java.util.Date;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;

@Entity
```

```
public class Person {

        @Id
        @GeneratedValue
        private int id;

        //No change in rest of the code

}
```

Step 15 - Implementing findById JPA Repository Method
DatabaseDemoApplication renamed to SpringJdbcDemoApplication

```
package com.in28minutes.database.databasedemo.jpa;

import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import javax.transaction.Transactional;

import org.springframework.stereotype.Repository;

import com.in28minutes.database.databasedemo.entity.Person;

@Repository
@Transactional
public class PersonJpaRepository {

        //connect to the database
        @PersistenceContext
        EntityManager entityManager;

        public Person findById(int id) {
                return entityManager.find(Person.class,
id);//JPA
        }
}
```

/src/main/resources/application.properties

```
spring.jpa.show-sql=true
```

/src/main/resources/data.sql - Comment Everything

```
/*
*/
```

JpaDemoApplication

```
package com.in28minutes.database.databasedemo;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import
org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import
org.springframework.boot.autoconfigure.SpringBootApplicatio
n;

import
com.in28minutes.database.databasedemo.jpa.PersonJpaReposito
ry;

@SpringBootApplication
public class JpaDemoApplication implements
CommandLineRunner {

        private Logger logger =
LoggerFactory.getLogger(this.getClass());

        @Autowired
        PersonJpaRepository repository;

        public static void main(String[] args) {

SpringApplication.run(JpaDemoApplication.class, args);
```

```
        }

        @Override
        public void run(String... args) throws Exception {

                logger.info("User id 10001 -> {}",
repository.findById(10001));

                /*
                logger.info("All users -> {}",
repository.findAll());
                logger.info("Deleting 10002 -> No of Rows
Deleted - {}",

repository.deleteById(10002));

                logger.info("Inserting 10004 -> {}",
                                repository.insert(new
Person(10004, "Tara", "Berlin", new Date())));

                logger.info("Update 10003 -> {}",
                                repository.update(new
Person(10003, "Pieter", "Utrecht", new Date())));
                */
        }
}
```

Step 16 - Implementing insert and update JPA Repository Methods

Step 17 - Implementing deleteById JPA Repository Method

Step 18 - Implementing findAll using JPQL Named Query

```
                logger.info("Inserting -> {}",
                                repository.insert(new
Person("Tara", "Berlin", new Date())));
                repository.deleteById(10002);



@Entity
```

```
@NamedQuery(name="find_all_persons", query="select p from
Person p")
public class Person
```

```
package com.in28minutes.database.databasedemo.jpa;

import java.util.List;

import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import javax.persistence.TypedQuery;
import javax.transaction.Transactional;

import org.springframework.stereotype.Repository;

import com.in28minutes.database.databasedemo.entity.Person;

@Repository
@Transactional
public class PersonJpaRepository {

        // connect to the database
        @PersistenceContext
        EntityManager entityManager;

        public List<Person> findAll() {
                TypedQuery<Person> namedQuery =
entityManager.createNamedQuery("find_all_persons",
Person.class);
                return namedQuery.getResultList();
        }

        public Person findById(int id) {
                return entityManager.find(Person.class,
id);// JPA
        }
        public Person update(Person person) {
                return entityManager.merge(person);
        }

        public Person insert(Person person) {
                return entityManager.merge(person);
        }

        public void deleteById(int id) {
                Person person = findById(id);
                entityManager.remove(person);
        }

}
```

Step 19 - Introduction to Spring Data JPA

JpaDemoApplication - comment out @SpringBootApplication

```
package com.in28minutes.database.databasedemo.springdata;

import
org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import com.in28minutes.database.databasedemo.entity.Person;

@Repository
public interface PersonSpringDataRepository
                                extends
JpaRepository<Person, Integer>{
}
```

```
package com.in28minutes.database.databasedemo;

import java.util.Date;

import org.slf4j.Logger;
```

```
import org.slf4j.LoggerFactory;
import
org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import
org.springframework.boot.autoconfigure.SpringBootApplicatio
n;

import com.in28minutes.database.databasedemo.entity.Person;
import
com.in28minutes.database.databasedemo.springdata.PersonSpri
ngDataRepository;

@SpringBootApplication
public class SpringDataDemoApplication implements
CommandLineRunner {

        private Logger logger =
LoggerFactory.getLogger(this.getClass());

        @Autowired
        PersonSpringDataRepository repository;

        public static void main(String[] args) {

SpringApplication.run(SpringDataDemoApplication.class,
args);
        }

        @Override
        public void run(String... args) throws Exception {

                logger.info("User id 10001 -> {}",
repository.findById(10001));

                logger.info("Inserting -> {}",
                                       repository.save(new
Person("Tara", "Berlin", new
 Date())));

                logger.info("Update 10003 -> {}",
                                repository.save(new
Person(10003, "Pieter", "Utrecht", new Date())));

                repository.deleteById(10002);

                logger.info("All users -> {}",
repository.findAll());
        }
}
```
Step 20 - Connecting to Other Databases
Connecting to My SQL and Other Databases
Spring Boot makes it easy to switch databases! Yeah really simple.

Steps

- Install MySQL and Setup Schema
- Remove H2 dependency from pom.xml
- Add MySQL (or your database) dependency to pom.xml
```xml

mysql mysql-connector-java

```
- Configure application.properties

```properties
spring.jpa.hibernate.ddl-auto=none
spring.datasource.url=jdbc:mysql://localhost:3306/person_ex
ample
spring.datasource.username=personuser
spring.datasource.password=YOUR_PASSWORD
```
- Restart the app and You are ready!

Spring Boot can setup the database for you using Hibernate

Things to note:

- Spring Boot chooses a default value for you based on whether it thinks your database is embedded (default create-drop) or not (default none).
- `spring.jpa.hibernate.ddl-auto` is the setting to perform SchemaManagementTool actions automatically
  - none : No action will be performed.
  - create-only : Database creation will be generated.
  - drop : Database dropping will be generated.
  - create : Database dropping will be generated followed by database creation.
  - validate : Validate the database schema
  - update : Update the database schema

- Reference : https://docs.jboss.org/hibernate/orm/5.2/userguide/html_single/Hibernate_User_Guide.html#configurations-hbmddl

application.properties

```
#none, validate, update, create, create-drop
spring.jpa.hibernate.ddl-auto=create
```

Installing and Setting Up MySQL

- Install MySQL https://dev.mysql.com/doc/en/installing.html
  - More details - http://www.mysqltutorial.org/install-mysql/
  - Trouble Shooting - https://dev.mysql.com/doc/refman/en/problems.html

- Startup the Server (as a service)
- Go to command prompt (or terminal)
  - Execute following commands to create a database and a user

```
mysql --user=user_name --password db_name
create database person_example;
create user 'personuser'@'localhost' identified by
'YOUR_PASSWORD';
grant all on person_example.* to 'personuser'@'localhost';
```

- Execute following sql queries to create the table and insert the data

Table

```
create table person
(
        id integer not null,
        birth_date timestamp,
        location varchar(255),
        name varchar(255),
        primary key (id)
);
```

Data

```
INSERT INTO PERSON (ID, NAME, LOCATION, BIRTH_DATE )
VALUES(10001,  'Ranga', 'Hyderabad',sysdate());
INSERT INTO PERSON (ID, NAME, LOCATION, BIRTH_DATE )
VALUES(10002,  'James', 'New York',sysdate());
INSERT INTO PERSON (ID, NAME, LOCATION, BIRTH_DATE )
VALUES(10003,  'Pieter', 'Amsterdam',sysdate());
```

Notes

JdbcTemplate AutoConfiguration

```
=========================
AUTO-CONFIGURATION REPORT
=========================

DataSourceAutoConfiguration matched:
   - @ConditionalOnClass found required classes
'javax.sql.DataSource',
'org.springframework.jdbc.datasource.embedded.EmbeddedDatab
aseType'; @ConditionalOnMissingClass did not find unwanted
class (OnClassCondition)

DataSourceTransactionManagerAutoConfiguration matched:
   - @ConditionalOnClass found required classes
'org.springframework.jdbc.core.JdbcTemplate',
'org.springframework.transaction.PlatformTransactionManager
';
```

```
 @ConditionalOnMissingClass did not find unwanted class
(OnClassCondition)

H2ConsoleAutoConfiguration matched:
   - @ConditionalOnClass found required class
'org.h2.server.web.WebServlet'; @ConditionalOnMissingClass
did not find unwanted class (OnClassCondition)
   - found ConfigurableWebEnvironment
(OnWebApplicationCondition)
   - @ConditionalOnProperty
(spring.h2.console.enabled=true) matched
(OnPropertyCondition)

JdbcTemplateAutoConfiguration matched:
   - @ConditionalOnClass found required classes
'javax.sql.DataSource',
'org.springframework.jdbc.core.JdbcTemplate';
@ConditionalOnMissingClass did not find unwanted class
(OnClassCondition)
   - @ConditionalOnSingleCandidate (types:
javax.sql.DataSource; SearchStrategy: all) found a primary
bean from beans 'dataSource' (OnBeanCondition)

JdbcTemplateAutoConfiguration.JdbcTemplateConfiguration#jdb
cTemplate matched:
   - @ConditionalOnMissingBean (types:
org.springframework.jdbc.core.JdbcOperations;
SearchStrategy: all) did not find any beans
(OnBeanCondition)
```

Bonus Introduction Sections

Other Introduction Sections

Title	Category	Github
Eclipse in 5 Steps	Introduction	Project Folder on Github
Maven in 5 Steps	Introduction	Project Folder on Github
JUnit in 5 Steps	Introduction	Project Folder on Github
Mockito in 5 Steps	Introduction	Project Folder on Github

Bonus Section - Basic Web Application

Bonus Section - Basic Web Application

- Understand Basics of HTTP
- HttpRequest - GET/POST, Request Parameters
- HTTP Response - Response Status - 404,200,500 etc
- Introduction to JSP, Servlets, Scriptlets and EL
- HTML Form - Method, Action & Form Data
- Understand Basics of using Maven, Tomcat and Eclipse
- Using Request Attributes for passing Model between Servlet and View
- Step 11 : Configure application to use Spring MVC
- Step 12 : First Spring MVC Controller, @ResponseBody, @Controller
- Step 13 : Redirect to Login JSP - LoginController, @ResponseBody and View Resolver
- Step 14 : DispatcherServlet and Log4j
- Step 15 : Show userid and password on the welcome page - ModelMap and @RequestParam
- Step 16 : LoginService and Remove all JEE Servlets based code
- Step 17 : Spring Auto-wiring and Dependency Management - @Autowired and @Service

Step 01 : Up and running with a Web Application in Tomcat

In this step, we will quickly setup a running web application.

> *Tip : This is one of the few steps where you copy code in! We would want to ensure that you have a running web application without any mistakes.*

You can run the project using Run as > Maven build > tomcat7:run.

You can copy code from

- Step 01 on Github Repository

\pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
```

```
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0
.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <groupId>com.in28minutes</groupId>
        <artifactId>in28Minutes-first-webapp</artifactId>
        <version>0.0.1-SNAPSHOT</version>
        <packaging>war</packaging>
        <dependencies>
                <dependency>
                        <groupId>javax</groupId>
                        <artifactId>javaee-web-
api</artifactId>
                        <version>6.0</version>
                        <scope>provided</scope>
                </dependency>
        </dependencies>
        <build>
                <pluginManagement>
                        <plugins>
                                <plugin>

<groupId>org.apache.maven.plugins</groupId>
                                        <artifactId>maven-
compiler-plugin</artifactId>

<version>3.2</version>


        <configuration>

<verbose>true</verbose>

<source>1.7</source>
```

```
        <target>1.7</target>

<showWarnings>true</showWarnings>
                                                </configuration>
                                        </plugin>
                                        <plugin>

<groupId>org.apache.tomcat.maven</groupId>

<artifactId>tomcat7-maven-plugin</artifactId>

<version>2.2</version>
                                                <configuration>


<path>/</path>

<contextReloadable>true</contextReloadable>
                                                </configuration>
                                        </plugin>
                                </plugins>
                        </pluginManagement>
        </build>
</project>
```

\src\main\java\webapp\LoginServlet.java

```
package webapp;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/*
```

```
 * Browser sends Http Request to Web Server
 *
 * Code in Web Server => Input:HttpRequest, Output:
HttpResponse
 * JEE with Servlets
 *
 * Web Server responds with Http Response
 */


@WebServlet(urlPatterns = "/login.do")
public class LoginServlet extends HttpServlet {

        @Override
        protected void doGet(HttpServletRequest request,
HttpServletResponse response) throws IOException {
                PrintWriter out = response.getWriter();
                out.println("<html>");
                out.println("<head>");
                out.println("<title>Yahoo!!!!!!!!
</title>");
                out.println("</head>");
                out.println("<body>");
                out.println("My First Servlet");
                out.println("</body>");
                out.println("</html>");

        }

}
```

\src\main\webapp\WEB-INF\web.xml

```
<!-- webapp/WEB-INF/web.xml -->
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
        xsi:schemaLocation="http://java.sun.com/xml/ns/java
ee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
        version="3.0">

        <display-name>To do List</display-name>

        <welcome-file-list>
                <welcome-file>login.do</welcome-file>
        </welcome-file-list>

</web-app>
```

Java Platform, Enterprise Edition (Java EE) JEE6

Servlet is a Java programming language class used to extend the capabilities of servers that host applications accessed by means of a request-response programming model.

Notes

- `extends javax.servlet.http.HttpServlet` - All servlets should extend HttpServlet class
- `@WebServlet(urlPatterns = "/login.do")` - Provide the url pattern to access the servlet
- `doGet(HttpServletRequest request, HttpServletResponse response)` - To handle the RequestMethod GET we need to implement doGet method.

Configuring welcome-file-list in web.xml will ensure that url http://localhost:8080/redirects to http://localhost:8080/login.do

```
<welcome-file-list>
        <welcome-file>login.do</welcome-file>
</welcome-file-list>
```

Step 02 : First JSP

Complete code

Notes

- Create LoginServlet again
- Redirect to a view - JSP

Code Snippets and Examples

Redirect to a view - JSP

\src\main\java\webapp\LoginServlet.java

```
request
 .getRequestDispatcher("/WEB-INF/views/login.jsp")
 .forward(request, response);
```

\src\main\webapp\WEB-INF\views\login.jsp

```
<html>
<head>
<title>Yahoo!!</title>
</head>
<body>
My First JSP!!!
</body>
</html>
```

Step 03 : Adding a Get Parameter name

Complete code

Notes

- Passing a Request Parameter Name

Code Snippets and Examples

We read the request parameter and set it as a request attribute. Request attributes can be accessed from the view (jsp).

\src\main\java\webapp\LoginServlet.java

```
request.setAttribute("name",
                request.getParameter("name"));
```

\src\main\webapp\WEB-INF\views\login.jsp

```
My First JSP!!! My name is ${name}
```

Step 04 : Adding another Get Parameter Password

Complete code

Code Snippets and Examples
\src\main\java\webapp\LoginServlet.java

```
request.setAttribute("password",
                  request.getParameter("password"));
```

\src\main\webapp\WEB-INF\views\login.jsp

```
My First JSP!!! My name is ${name} and password is
${password}
```

Step 05 : Let’s add a form

Complete code

Code Snippets and Examples
\src\main\java\webapp\LoginServlet.java

```
@Override
protected void doGet(HttpServletRequest request,
HttpServletResponse response)
                throws IOException, ServletException {
        request


        .getRequestDispatcher("/WEB-INF/views/login.jsp")
                .forward(request, response);
}
```

\src\main\webapp\WEB-INF\views\login.jsp

```
<html>
<head>
<title>Yahoo!!</title>
</head>
<body>
        <form action="/login.do" method="POST">
                Name : <input type="text" /> <input
type="submit" />
        </form>
</body>
</html>
```

\src\main\webapp\WEB-INF\views\welcome.jsp

```
<html>
<head>
<title>Yahoo!!</title>
</head>
<body>
Welcome ${name}
</body>
</html>
```

Step 06 : New Form and doPost

Complete code

\src\main\java\webapp\LoginServlet.java

```
@Override
protected void doPost(HttpServletRequest request,
HttpServletResponse
```

```
 response)
                throws IOException, ServletException {
        request.setAttribute("name",
request.getParameter("name"));
        request.getRequestDispatcher("/WEB-
INF/views/welcome.jsp").forward(request, response);
}
```

\src\main\webapp\WEB-INF\views\welcome.jsp

```
<html>
<head>
<title>Yahoo!!</title>
</head>
<body>
Welcome ${name}
</body>
</html>
```

Step 07 : Adding Password and Validation of User Id

Complete code

Code Snippets and Examples
\src\main\java\webapp\LoginService.java

```
public class LoginService {
        public boolean validateUser(String user, String
password) {
                return user.equalsIgnoreCase("in28Minutes")
&& password.equals("dummy");
        }


}
```

\src\main\java\webapp\LoginServlet.java

```
@Override
```

```
protected void doPost(HttpServletRequest request,
HttpServletResponse response)
                throws IOException, ServletException {
        String name = request.getParameter("name");
        String password = request.getParameter("password");

        boolean isValidUser = service.validateUser(name,
password);

        if (isValidUser) {
                request.setAttribute("name", name);
                request.getRequestDispatcher("/WEB-
INF/views/welcome.jsp")
                                .forward(request,
response);
        } else {
                request.setAttribute("errorMessage",
"Invalid Credentials!!");
                request.getRequestDispatcher("/WEB-
INF/views/login.jsp")
                                .forward(request,
response);
        }
}
```

\src\main\webapp\WEB-INF\views\login.jsp

```
<html>
<head>
<title>Yahoo!!</title>
</head>
<body>
        <p><font color="red">${errorMessage}</font></p>
        <form action="/login.do" method="POST">
                Name : <input name="name" type="text" />
Password : <input name="password" type="password" /> <input
```

```
 type="submit" />

        </form>
</body>
</html>
```

Step 11 : Configure application to use Spring MVC

What we will do
Before we start with the Flows, we need to configure application to use Spring MVC

- Lets do a little bit of Refactoring. Mini Step 1: Rename package webapp to com.in28minutes.jee
- We need Spring MVC Framework and its dependencies. Mini Step 2 : Add required jars to the project
- Spring MVC uses Front Controller Pattern -> Dispatcher Servlet. Mini Step 3 : Add Dispatcher Servlet to web.xml
- DispatcherServlet needs an Spring Application Context to launch. We will create an xml (/WEB-INF/todo-servlet.xml). Mini Step 4: Add Spring Context

Useful Snippets
pom.xml

```
                <dependency>

<groupId>org.springframework</groupId>
                        <artifactId>spring-
webmvc</artifactId>
                        <version>4.2.2.RELEASE</version>
                </dependency>
```

web.xml

```
            <servlet>
                <servlet-name>dispatcher</servlet-name>
                <servlet-class>
```

```
 org.springframework.web.servlet.DispatcherServlet


 </servlet-class>
               <init-param>
                   <param-
name>contextConfigLocation</param-name>
                   <param-value>/WEB-INF/todo-
servlet.xml</param-value>
               </init-param>
               <load-on-startup>1</load-on-startup>
           </servlet>

           <servlet-mapping>
               <servlet-name>dispatcher</servlet-name>
               <url-pattern>/spring-mvc/*</url-pattern>
           </servlet-mapping>
```

todo-servlet.xml

```
        <beans
xmlns="http://www.springframework.org/schema/beans"

xmlns:context="http://www.springframework.org/schema/contex
t"

xmlns:mvc="http://www.springframework.org/schema/mvc"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"

xsi:schemaLocation="http://www.springframework.org/schema/b
eans http://www.springframework.org/schema/beans/spring-
beans.xsd
            http://www.springframework.org/schema/mvc
http://www.springframework.org/schema/mvc/spring-mvc.xsd
            http://www.springframework.org/schema/context
```

```
 http://www.springframework.org/schema/context/spring-
context.xsd">

            <context:component-scan base-
package="com.in28minutes"

 />

            <mvc:annotation-driven />

        </beans>
```

Flows:

- Flow 1. Login Servlet -> GET -> login.jsp
- Flow 2. Login Servlet -> POST (Success) -> welcome.jsp
- Flow 3. Login Servlet -> POST (Failure) -> login.jsp (with error message)

Files List

\src\main\webapp\WEB-INF\views\login.jsp Deleted

\pom.xml Deleted

\src\main\java\webapp\LoginService.java Deleted

\src\main\java\webapp\LoginServlet.java Deleted

\src\main\webapp\WEB-INF\views\welcome.jsp Deleted

\src\main\webapp\WEB-INF\web.xml Deleted

/pom.xml New

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0
.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <groupId>com.in28minutes</groupId>
        <artifactId>in28Minutes-springmvc</artifactId>
        <version>0.0.1-SNAPSHOT</version>
        <packaging>war</packaging>


        <dependencies>
                <dependency>
                        <groupId>javax</groupId>
                        <artifactId>javaee-web-
api</artifactId>
                        <version>6.0</version>
                        <scope>provided</scope>
                </dependency>

                <dependency>

<groupId>org.springframework</groupId>
                        <artifactId>spring-
webmvc</artifactId>
                        <version>4.2.2.RELEASE</version>
                </dependency>
        </dependencies>

        <build>
                <pluginManagement>
                        <plugins>
                                <plugin>

<groupId>org.apache.maven.plugins</groupId>
                                        <artifactId>maven-
compiler-plugin</artifactId>

<version>3.2</version>
                                        <configuration>
        <verbose>true</verbose>

<source>1.8</source>

<target>1.8</target>


        <showWarnings>true</showWarnings>
                                                </configuration>
                                        </plugin>
                                        <plugin>

<groupId>org.apache.tomcat.maven</groupId>

<artifactId>tomcat7-maven-plugin</artifactId>

<version>2.2</version>
                                                <configuration>

<path>/</path>

<contextReloadable>true</contextReloadable>
                                                </configuration>
                                        </plugin>
                                </plugins>
                        </pluginManagement>
        </build>
</project>
```

/src/main/java/com/in28minutes/jee/LoginService.java New

```
package com.in28minutes.jee;

public class LoginService {
        public boolean validateUser(String user, String
password) {
                return user.equalsIgnoreCase("in28Minutes")
```

```
 && password.equals("dummy");
        }

}
```

/src/main/java/com/in28minutes/jee/LoginServlet.java New

```
package com.in28minutes.jee;

import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet(urlPatterns = "/login.do")
public class LoginServlet extends HttpServlet {

        private LoginService service = new LoginService();

        @Override
        protected void doGet(HttpServletRequest request,
                        HttpServletResponse response)
throws IOException, ServletException {
                request.getRequestDispatcher("/WEB-
INF/views/login.jsp").forward(
                                request, response);
        }

        @Override
        protected void doPost(HttpServletRequest request,
                        HttpServletResponse response)
throws IOException, ServletException {
                String name = request.getParameter("name");
                String password =
```

```
 request.getParameter("password");

                boolean isValidUser =
service.validateUser(name, password);

                if (isValidUser) {
                        request.setAttribute("name", name);


        request.getRequestDispatcher("/WEB-
INF/views/welcome.jsp").forward(
                                        request, response);
                } else {

request.setAttribute("errorMessage", "Invalid
Credentials!!");
                        request.getRequestDispatcher("/WEB-
INF/views/login.jsp").forward(
                                        request, response);
                }
        }

}
```

/src/main/webapp/WEB-INF/todo-servlet.xml New

```
<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:context="http://www.springframework.org/schema/contex
t"
    xmlns:mvc="http://www.springframework.org/schema/mvc"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/b
eans http://www.springframework.org/schema/beans/spring-
beans.xsd
    http://www.springframework.org/schema/mvc
http://www.springframework.org/schema/mvc/spring-mvc.xsd
    http://www.springframework.org/schema/context
```

```
 http://www.springframework.org/schema/context/spring-
context.xsd">

    <context:component-scan base-package="com.in28minutes"
/>

    <mvc:annotation-driven />

</beans>
```

/src/main/webapp/WEB-INF/views/login.jsp New

```
<html>
<head>
<title>Yahoo!!</title>
</head>
<body>
    <p><font color="red">${errorMessage}</font></p>
    <form action="/login.do" method="POST">
        Name : <input name="name" type="text" /> Password :
<input name="password" type="password" /> <input
type="submit" />
    </form>
</body>
</html>
```

/src/main/webapp/WEB-INF/views/welcome.jsp New

```
<html>
<head>
<title>Yahoo!!</title>
</head>
<body>
Welcome ${name}
</body>
</html>
```

/src/main/webapp/WEB-INF/web.xml New

```
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
    version="3.0">

    <display-name>To do List</display-name>

    <servlet>
        <servlet-name>dispatcher</servlet-name>


 <servlet-class>

org.springframework.web.servlet.DispatcherServlet
        </servlet-class>
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>/WEB-INF/todo-servlet.xml</param-
value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>dispatcher</servlet-name>
        <url-pattern>/spring-mvc/*</url-pattern>
    </servlet-mapping>
</web-app>
```

Step 12 : First Spring MVC Controller, @ResponseBody, @Controller

#First Spring MVC Controller

- @RequestMapping(value = "/login", method = RequestMethod.GET)
- http://localhost:8080/spring-mvc/login

- web.xml - /spring-mvc/*
- Why @ResponseBody?
- Importance of RequestMapping method
- Can I have multiple urls rendered from Same Controller?

#Snippets

```
package com.in28minutes.springmvc;

import org.springframework.stereotype.Controller;
import 
org.springframework.web.bind.annotation.RequestMapping;


import 
org.springframework.web.bind.annotation.ResponseBody;

@Controller
public class LoginController {

        @RequestMapping(value = "/login")
        @ResponseBody
        public String sayHello() {
                return "Hello World dummy";
        }
}
```

Step 13 : Redirect to Login JSP - LoginController, @ResponseBody - and View Resolver

/src/main/java/com/in28minutes/springmvc/login/LoginController.java New

```
package com.in28minutes.springmvc.login;

import org.springframework.stereotype.Controller;
import
```

```
 org.springframework.web.bind.annotation.RequestMapping;
import 
org.springframework.web.bind.annotation.RequestMethod;

@Controller
public class LoginController {
        @RequestMapping(value = "/login", method = 
RequestMethod.GET)
        public String showLoginPage() {
                return "login";
        }
}
```
/src/main/webapp/WEB-INF/todo-servlet.xml Modified

New Lines

```
    <bean
        
class="org.springframework.web.servlet.view.InternalResourc
eViewResolver">
        <property name="prefix">
            <value>/WEB-INF/views/</value>
        </property>
        <property name="suffix">
            <value>.jsp</value>
        </property>
    </bean>
```
Redirect to Login JSP

- View Resolver in todo-servlet.xml
- Update LoginController
- Remove @ResponseBody
- More about View Resolver

Snippets

```
  <bean
       
```

```
 class="org.springframework.web.servlet.view.InternalResour
ceViewResolver">
        <property name="prefix">
            <value>/WEB-INF/views/</value>
        </property>
        <property name="suffix">
            <value>.jsp</value>
        </property>
    </bean>
```

Step 14 : DispatcherServlet and Log4j

/pom.xml Modified

New Lines

```
<dependency>
        <groupId>log4j</groupId>
        <artifactId>log4j</artifactId>
        <version>1.2.17</version>
</dependency>
```

/src/main/resources/log4j.properties New

```
log4j.rootLogger=TRACE, Appender1, Appender2

log4j.appender.Appender1=org.apache.log4j.ConsoleAppender
log4j.appender.Appender1.layout=org.apache.log4j.PatternLay
out
log4j.appender.Appender1.layout.ConversionPattern=%-7p %d
[%t] %c %x - %m%n

```

/src/main/webapp/WEB-INF/views/login.jsp Modified

New Lines

```
    <form action="/spring-mvc/login" method="POST">
```

/src/main/webapp/WEB-INF/views/welcome.jsp Modified

New Lines

```
Welcome ${name}.
```

What we want to do:

- Understand importance of DispatcherServlet.
- Add Logging Framework Log4j to understand the flow much more.

Spring MVC Request Flow

- DispatcherServlet receives HTTP Request.
- DispatcherServlet identifies the right Controller based on the URL.
- Controller executes Business Logic.
- Controller returns a) Model b) View Name Back to DispatcherServlet.
- DispatcherServlet identifies the correct view (ViewResolver).
- DispatcherServlet makes the model available to view and executes it.

- DispatcherServlet returns HTTP Response Back.
- Flow : http://docs.spring.io/spring-framework/docs/2.0.8/reference/images/mvc.png

Step 15 : Show userid and password on the welcome page - ModelMap and @RequestParam

- Show userid and password on the welcome page.
- We will not use Spring Security for now.
- ModelMap model
- @RequestParam String name

/src/main/java/com/in28minutes/springmvc/login/LoginController.java Modified

New Lines

```
        @RequestMapping(value = "/login", method = 
RequestMethod.POST)
        public String handleUserLogin(ModelMap model, 
@RequestParam String name,
                        @RequestParam String password) {
                model.put("name",
 name);
                model.put("password", password);
                return "welcome";
        }
}
```

/src/main/webapp/WEB-INF/views/welcome.jsp Modified

New Lines

```
Welcome ${name}. You entered ${password}
```

Step 16 : LoginService and Remove all JEE Servlets based code

- Use LoginService to validate userid and password.
- Remove all the old controller code and lets use only Spring MVC here on.
- For now : We are not using Spring Autowiring for LoginService.

- Change URL to http://localhost:8080/login

/src/main/java/com/in28minutes/jee/LoginService.java Deleted

/src/main/java/com/in28minutes/jee/LoginServlet.java Deleted

/src/main/java/com/in28minutes/springmvc/login/LoginController.java Deleted

/src/main/java/com/in28minutes/login/LoginController.java New

```
package com.in28minutes.login;

import org.springframework.stereotype.Controller;
import org.springframework.ui.ModelMap;
import 
org.springframework.web.bind.annotation.RequestMapping;
import 
org.springframework.web.bind.annotation.RequestMethod;
import 
org.springframework.web.bind.annotation.RequestParam;
```

```
import com.in28minutes.login.LoginService;

@Controller
public class LoginController {

        private LoginService loginService = new 
LoginService();

        @RequestMapping(value = "/login", method = 
RequestMethod.GET)
        public String showLoginPage() {
                return "login";
        }

        @RequestMapping(value = "/login", method = 
RequestMethod.POST)
        public String handleUserLogin(ModelMap model,
 @RequestParam String name,
                        @RequestParam String password) {

                if (!loginService.validateUser(name, 
password)) {
                        model.put("errorMessage", "Invalid 
Credentials");
                        return "login";
                }

                model.put("name", name);
                return "welcome";
        }
}
```

/src/main/java/com/in28minutes/login/LoginService.java New

```
package com.in28minutes.login;
```

```
public class LoginService {
        public boolean validateUser(String user, String 
password) {
                return user.equalsIgnoreCase("in28Minutes") 
&& password.equals("dummy");
        }

}
```

/src/main/webapp/WEB-INF/views/login.jsp Modified

New Lines

```
    <form action="/login" method="POST">
```

/src/main/webapp/WEB-INF/views/welcome.jsp Modified

New Lines

```
Welcome ${name}. You are now authenticated.
```

/src/main/webapp/WEB-INF/web.xml Modified

New Lines

```
        <url-pattern>/</url-pattern>
```

Step 17 : Spring Auto-wiring and Dependency Management - @Autowired and @Service

- Learn about Spring Auto-wiring and Dependency Management.
- Use Auto-wiring to wire LoginService.
- @Autowired, @Service

/src/main/java/com/in28minutes/login/LoginController.java Modified

New Lines

```
import 
org.springframework.beans.factory.annotation.Autowired;
        @Autowired
        private LoginService loginService;
```

/src/main/java/com/in28minutes/login/LoginService.java Modified

New Lines

```
import org.springframework.stereotype.Service;
@Service
public class LoginService {
```