SBMS Class Notes by ashok -2022

Spring Boot with Cloud & Microservices

-----------------------------------------------------------------------

-> In last session we discussed below topics

Why to learn spring boot & microservices

What are the pre-requisites to attend spring boot & microservices

1) Core Java

2) Advaned Java (jdbc & servlets)

3) SQL Basics

Course Content

Module-1 : Spring Basics, Spring Boot

Module-2 : Spring Data JPA

Module-3 : Spring Web MVC

Module-4 : RESTFul Services

Module-5 : Microservices (Kafka, Redis, Api gateway, Eureka, Config etc)

Module-6 : Spring Cloud & Security (OAuth & JWT)

Module-7 : Realtime Tools(Junit, Docker, Jenkins, Log4j etc)

Module-8 : Mini Project, Angular Integration, AWS Cloud Deployment

Duration, Timings, Fees

-------------------------------------------------------------------------

Course Duration : 3.5 Months

Course Fee : 5000 INR Only (Live Classes + Daily Class Notes)

Course Fee : 10000 INR Only (Live Classes + Daily Notes + Daily Video) - 6 Months

Class Timings

-------------

9:30 AM - 10:30 AM IST For Module-1 & Module-2 (2 weeks)

9:30 PM - 10:45 PM IST for Module-3 to Module-8 (3 Months)

Note: You no need to write running notes because daily i will share class notes

-> 70+ applications we are going to develop from scratch in our classes

-> After this training completed you can claim 3-4 years experience as Spring Boot & Microservices developer

-> Average salary for 3-4 years experienced Spring Boot developer 8 - 16 lakhs

-----------------------------------------------------------------------------------

Module-1 : Spring Basics & Spring Boot

----------------------------------------------------------------------------------

-> Spring is an open source java based framework

-> By using Spring framework we can develop end to end applications

-> Spring framework is Application Development framework

-> Spring framework is loosely coupled framework

-> Spring framework is light weight framework

-> Spring framework is non-invasive framework

-> Spring framework is versatile framework

-> Spring framework came into market in the year of 2004 as Spring 1.0v

-> The current version of Spring Framework is 5.x

-> Spring framework developed in modular fashion

In last session we have started our discussion related to Spring framework

============================================================================================

-> Spring is an open source java based framework

-> Spring framework came into market in 2004 as Spring 1.0v

-> The current version of Spring framework is 5.x version

-> By using spring framework we can develop end to end application

-> As we can develop entire application by using Spring, we can call it as application development framework

-> Spring framework is light weight

-> Spring framework is loosely coupled

-> Spring framework is versatile (It can be integrated with any other java framework available in the market)

-> Spring framework is non-invasive framework (It will not force to use any spring classes or interfaces)

==============================================================================================

=> Spring framework developed in modular fashion

=> In spring framework we have several modules, they are

Spring Core

Spring Context

Spring AOP

Spring DAO/JDBC

Spring ORM

Spring Web MVC

Spring REST

Spring Cloud

Spring Security

Spring Batch etc....

Note: Spring framework is not forcing to use all the above modules. Based on our requirement we can choose and we can use that particular module.

-----------------------------------------------------------------------------------------------

-> Spring Core is base module in Spring Framework. All the other spring modules built on top of Spring Core Module.

-> Spring Core Module providing fundamental concepts of Spring framework they are IOC and Dependency Injection.

-----------------------------------------------------------------------------------------------

-> Spring Context module will deal with configurations required for our application development

----------------------------------------------------------------------------------------------

-> AOP stands for Aspect Oriented Programming.

-> Spring AOP module is used to seperate business logic and supporting logics

-> Supporting logics means logging, transaction, security, auditing etc...

----------------------------------------------------------------------------------------------

-> Spring JDBC/DAO module is used to develop persistence logic

-> The logic which is responsible to communicate with Database is called as Persistence Logic

-> Spring JDBC is resolving the problems of JDBC api

-> Spring JDBC is developed on top of JDBC API

----------------------------------------------------------------------------------------------

-> ORM stands for Object Relational Mapping

-> ORM is used to represent data in the form of objects

-> Spring ORM module is used to persistence layer with ORM principles

----------------------------------------------------------------------------------------------

-> Spring Web MVC module is used to develop web applications

-> Using Spring Web MVC we can develop both Web & distributed applications

----------------------------------------------------------------------------------------------

-> Spring Security is used to secure our web applicaitons with Authorization and Authentication

----------------------------------------------------------------------------------------------

-> Spring Batch is used to perform business operations on bulk data

----------------------------------------------------------------------------------------------

-> Spring Boot is one approach to develop spring applications with less configurations

----------------------------------------------------------------------------------------------

-> Spring REST is used to develop RESTFul services

----------------------------------------------------------------------------------------------

-> Spring cloud is used to develop our application with cloud concepts like

API Gateway

Service Registry

Load balancing etc..

-----------------------------------------------------------------------------------------------

=> In Tomorrow session we will start our discussion with Spring Core Module

---------------------------------------------------------------------------------------------

In last session we discussed about Spring Module

--------------------------------------------------------

-> Spring framework developed in modular fashion

-> The first version of spring framework having 7 modules

Spring Core

Spring Context

Spring AOP

Spring DAO

Spring ORM

Spring Web

Spring WebMVC

-> In Spring second version we have 6 modules because they have combined Spring Web & Spring WebMVC as single module.

-> In Spring 3.x version they released 20+ modules...

-> The current version of Spring is 5.x version

-----------------------------------------------------------------------------------

-> Spring Core Module is base module for Spring framework

-> Core module provided IOC & Dependency Injection concepts

----------------------------------------------------------------------------------

Spring Core Introduction

10 December 2021 09:37

-> If we want to call one java class method in another java class method then we have below 2

options

1) Inheritance

2) Composition

-> Inheritance means extending the properties of one class in another class to re-use the methods

-> Composition means creating one class object in another class

-> If we go for inheritance or composition then our classes will become tightly coupled.

-> To make our classes loosely coupled we will use Spring Core Module.

-> Spring Core Module providing IOC & DI to make classes loosely coupled.

Note: If we use IOC & DI then without extending properties and without creating object we can

access one class method in another class.

Use case - Inheritance

package in.ashokit; public class Engine (

public int start() {

//logic to start return 1;

package in.ashok it;

public class Car extends Engine ( public void drive() (

int start = start(); if (start == 1) {

System.out.printIn("Journey Started. ");

} else (

System.out.printIn("There is a problem with Engine");

New Section 5 Page 1

package in.ashokit; public class Test (

public static void main(String[] args) ( Car car = new Car();

car.drive();

=> In above application Car class extending properties from Engine

=> Without having Engine class we can't compile Car class

=> Car class is dependent on Engine

=> Car class can't extend properties from any other class in future because java doesn't support

multiple inheritance.

To overcome the problems of Inheritance we are using Composition

package in.ashokit; public class Engine (

public int start() {

// logic to start engine return 1;

package in.ashokit;

public class Car (

public void drive() (

Engine eng -- new Engine(); int start -- eng.start{};

if (start == 1) (

System.out.printIn("Journey Started...");

) else (

System.out.printIn("Failed to start");

package in.ashokit; public class Test (

public static void main(String[] args) { Car car = new Car();

car.drive();

New Section 5 Page 2

=> In the above approach we are creating Engine class object in Car class

=> Car class is directly dependent on Engine class

-> As we are creating object we should know object creation process

=> Without Engine class , Car class can't be compiled

=> In this approach also classes are tightly coupled

=> If we use either Inheritance or Composition then our classes are becoming tightly coupled.

=> To overcome this tightly coupling problem, we can use IOC container provided by Spring Core

Module.

-> loc container will take care of Dependency Injection

What is Dependency Injection?

-> Injecting one class object into another class object is called as Dependency Injection.

-> In Spring Framework IOC container is responsible to perform Dependency Injection

Usecase : Engine class object and Car class object will be created by IOC and IOC will inject

Engine obj into Car obj.

What is IOC Container?

--> IOC is a principle which is used to manage and collaborate dependencies among the classes in

the application.

-> IOC will resolve tightly coupling problem using DI (dependency injection)

Configuration/Instructions

Classes

IOC

New Section 5 Page 3

Produce

Beans Ready To Use

eveloping First Application using Spring Core Module

11 December 2021 09:42

1) Create Maven Project Using IDE (Eclipse/ STS / Intelli J)

2) Add Spring Framework dependency in Maven project porn.xml file

3) Create Engine class with start ( ) method

4) Create Car class with drive( ) method

5) Create Spring Bean Configuration File to give instruction to ioc to perform Dependency Injection

(src/main/resources/Beans.xml)

6) Test the application

<project xmIns="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</modeIVersion>

<groupId>in.ashokit</groupId>

<artifactId>03-Spring-App-1</artifactId>

<version>0.0.1-SNAPSHOT</version>

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.13</version>

</dependency>

</dependencies>

</project>

package in.ashokit.beans;

public class Engine (

public Engine() (

System.out.println("\*\* Engine :: Constructor \*\*");

public int start() (

return 1;

package in.ashokit.beans;

public class Car (

private Engine eng;

New Section 6 Page 1

public Car(Engine eng) {

System.out.println("\*\* Car :: Constructor \*\*"); this.eng = eng;

public void drive() (

int start = eng.start(); if (start == 1) (

System.out.println("\*\*Journey Started\*\*\*");

) else (

System.out.println("\*\* Engine Having Problem \*\*");

<?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/beans

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

<bean id="engine" class="in.ashokit.beans.Engine" />

<bean id="car" class="in.ashokit.beans.Car">

<constructor-arg name="eng" ref="engine" />

</bean>

</beans>

package in.ashokit.beans;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext; public class Test (

public static void main(String[] args) {

ApplicationContext context = new CIassPathXmlAppIicationContext("Beans.xml"); Car car =

context.getBean(Car.class);

car.drive();

Developed First App Using Spring Framework

13 December 2021 09:36

-> What is IOC Container ?

-> What is Dependency Injection ?

-> IOC is a principle which is used to manage dependencies among the classes in the application

-> IOC will perform Dependency Injection

-> Injecting one class object into another class object is called as Dependency Injection

class User {

private int age;

\_

cla s s Demo (

public static void main(String... args){

User user - new User ( );

user. age = ze;

This is invalid because we are trying to initialize User class private variable in Demo class.

=> If we want to 1n1t1a11ze age varlab1e then we have 2 options

1) setter method

2) parameterized constructor

New Section 7 Page 1

In III a11zat1on with setter method

class User {

private Int age,

public void setAge(Int age){ this .age = age;

class Demo {

psvm(Str1ng... args){

User user = new User ( );

user. setAge(ze);

-> In the above program we are initializing age variable using setter method hence it is called as

Setter Injection.

In1t1a11zat1on with Constructor

class User (

private Int age;

public User(int age)(

this . age = age;

class Demo {

public static void main (St ring ... args){ User user = new User (2€I) ;

-> In the above program we are initializing age variable using constructor hence it is called as

Constructor Injection.

package in.ashokit.beans; public class Engine (

public Engine() (

System.out.println("’ Engine :: Constructor \*’");

public int start() (

// logic to start return 1;

package in.ashokit.beans; public class Car {

public Car() (

System.out.println("’ Car :: Constructor \*\*"); private Engine eng;

New Section 7 Page 2

public void setEng(Engine eng) ( System.out.println("\*’setEng( ) method called \*\*"); this.eng =

eng;

public void drive() (

int start = eng.start(); if (start == 1) (

System.out.println(" \*\* Journey Started \*\*");

} else (

System.out.println("\*\* Problem with Engine \*\*");

<?xmI 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/beans

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

<bean id="dieseIEng" cIass="in.ashokit.beans.Engine" />

<bean id="car" class="in.ashokit.beans.Car">

<property name="eng" ref="dieseIEng" />

</bean>

</beans>

package in.ashokit.test;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext; import

in.ashokit.beans.Car;

public class Test {

public static void main(String[] args) (

ApplicationContext context = new CIassPathXmIAppIicationContext("Beans.xml"); Car car = context.get

Bean(Car.class);

car.drive();

---Assignment------------------------------------------------------------------

IPr‘1nter

void print( )

SonyPrinter

AtmMachine

IPrinter printer,

void withdraw(long amt)

-> When we withdraw amt from ATM, then it will give money & it will print receipt Note: AtmMachine

withdraw ( ) method should call printer print ( ) method

14 December 2021 09:24

=> Setter Injection means injecting value to variable using Setter method

=> We will represent setter injection using <property/> tag

=> Constructor means injecting value to variable using Constructor

=> We will represent Constructor Injection using <constructor-arg/> tag

package in.ashokit.beans;

public interface IPrinter {

public void print();

package in.ashokit.beans;

public class HpPrinter implements IPrinter {

public HpPrinter() (

System.out.println("\*\*\* HP :: Constructor \*\*\*");

public void print() (

System.out.printIn("Statement printed by HP Printer. ");

package in.ashokit.beans;

public class SonyPrinter implements IPrinter (

public SonyPrinter() {

System.out.println("\*\* SonyPrinter :: Constructor \*’");

public void print() {

System.out.printIn("Statement printing by Sony Printer. ");

package in.ashokit.beans;

public class AtmMachine { private IPrinter printer;

public AtmMachine(I Printer printer) (

System.out.println("\*\* AtmMachine :: Constructor \*\*"); this.printer = printer;

public void withdraw(int amt) { System.out.printIn("Amount withdraw success. ");

printer.print();

New Section 9 Page 1

<?xmI version="1.0" encoding="UTF-8"?>

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

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

xsi:schemaLocation="http://www.springframework.org/schema/beans

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

<bean id="hpPrinter" class="in.ashokit.beans.HpPrinter" />

<bean id="sonyPrinter" cIass="in.ashokit.beans.SonyPrinter" />

<bean id="atm" class="in.ashokit.beans.AtmMachine">

<constructor-arg name="printer" ref="hpPrinter" />

</bean>

</beans>

package in.ashokit.test;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext; import

in.ashokit.beans.AtmMachine;

public class Test {

public static void main(String[] args) (

ApplicationContext context = new CIassPathXmIAppIicationContext("Beans.xml"); AtmMachine atm =

context.getBean(AtmMachine.class);

atm.withdraw(2500);

=> If we do both Setter & Constructor injection for same variable then Setter Injection will

override Constructor Injection because Constructor will execute first and will initialize that

variable after that setter method will execute and it will re-initialize same variable hence setter

injection value will be the final value.

In last session we have implemented an application using setter Injection & constructor injection

-------------------------------------------------------------------------

-> IOC container is responsible to perform Dependency Injection in spring based applications

-> With the help of IOC we can develop our classes with loosely coupling

-> Setter Injection means injecting value to variable with setter method

-> Constructor Injection means injecting value to variable with constructor

-> Setter Injection will be represented by using <property/> tag

-> Constructor Injection will be represented by using <constructor-arg/> tag

-> In setter injection dependencies are optional

-> In constructor injection dependencies are mandatory

-> If we do both setter & constructor injection for same variable then first constructor injection will happen then setter injection will happen so final value will be setter injection value.

Note: Setter Injection will override constructor injection value

-----------------------------------------------------------------------------------

Bean Scopes

----------------------------------------------------------------------------------

-> Bean scope will represent how many objects should be created for a bean

-> In Spring framework we have 4 scopes for a bean

1) singleton

2) prototype

3) request

4) session

-> Singleton scope means for spring bean only one object will be created by IOC. This is default scope.

-> Prototype scope means for spring bean new object will be created everytime

-> request & session scopes are related to Spring Web MVC module (will discuss in that module)

//in this bean def no scope mentioned hence it is singleton

<bean id="robot" class="in.ashokit.beans.Robot"/>

//in this bean def scope mentioned as prototype

<bean id="robot" class="in.ashokit.beans.Robot" scope="prototype"/>

Note: When IOC container started objects will be created for singleton scoped beans. For prototype scoped beans objects will be created on-demand basis.

-> If we make all the beans as prototype un-necessarily too many objects will be created and it leads to out of memory issues.

----------------------------------------------------------------------------------

Usecase

---------------------------------------------------------------------------------

We have to generate a new ticket for every customer using TicketGenerator class

TicketGenerator class

Ticket class

-> For every customer we have to generate new ticket

-> To generate multiple Tickets one TicketGenerator class obj is sufficient

----------------------------------------------------------------------------------

What is Spring Framework

Spring Framework Advantages

Spring Modules

Spring Core Module Introduction

IOC Container

Dependency Injection

Setter Injection

Constructor Injection

SI vs CI

Bean Scopes (Singleton & Prototype)

Bean Configuration

Spring Applications Development

----------------------------------------------------------------------------------

Last session we completed our discusssion related to spring based applications with xml approach

-> If we develop applications using spring framework then as a programmer we have to take care of configurations.

-> To overcome the problems of spring framework spring team released Spring Boot

--------------------------------------------------------------------------

Spring Boot

-------------------------------------------------------------------------

-> Spring Boot is an approach to develop spring based applications with minimal configurations.

-> The main advantage of going for Spring boot is Auto configuration

-> Auto Configuration means it will identify the configuration required for our application and it will provide that configuration

-> Whatever we can develop using spring framework, same things can be developed by using spring boot also with less configurations.

Advantages of Spring Boot

--------------------------

1) Starter POM

2) Dependency Version Management

3) Auto Configuration

4) Embedded Servers

5) Actuators etc...

-> Starter poms simplifying maven/gradle configuration

spring-boot-starter

spring-boot-starter-data-jpa

spring-boot-starter-web

spring-boot-starter-security

spring-boot-starter-mail etc..

-> When we are adding dependencies in spring boot application we no need to mention version number for that dependency. Spring Boot Parent Starter will take care of version for child starter/dependencies.

-> Auto Configuration means Spring Boot will take care of configuration required for our application execution (No need of any xml files for configurations)

-> When we run web application boot will deploy our application into embedded server by default. We no need to install and run server to run boot web application.

-> Actuators are used to monitor and manage our application

how many beans loaded by our app

what are the url-patterns available in our app

what is env of our application

how many objects created

how many threads are running

----------------------------------------------------------------------------------

Spring Boot Application Creation

---------------------------------------------------------------------------------

-> We can create Spring Boot application in 2 ways

1) Spring Initializr website ( start.spring.io )

2) Using STS IDE

Note: If we use STS ide to create spring-starter-application internally it will communicate with Initializr website only to create it.

Note: Internet connection is mandatory to create spring boot applications.

> In Last session we have started our discussion related to Spring Boot

-> Spring Boot is an approach to develop spring based applications with less configurations.

-> Spring Boot is not an alternative for spring... spring boot internally using spring only

-> Whatever we can develop using spring framework same things can be developed by using spring boot also

Advantages of spring boot

--------------------------

1) starter poms

2) dependency version conflicts resolution

3) auto configuration

4) embedded servers

5) actuators

-> Spring Boot application we can create in 2 ways

1) initializer website (start.spring.io)

2) STS IDE

Note: When we create spring boot application internally it will connect with initializer website only to create the project

Note: To create spring boot application internet connection is mandatory

-------------------------------------------------------------------------

-> When we create spring boot application we will get one start class by default.

-> That start class is entry point for spring boot application execution

-----------------------------------------------------------------------

@SpringBootApplication

public class Application {

public static void main(String[] args) {

SpringApplication.run(Application.class, args);

}

}

------------------------------------------------------------------------

-> Weather it is a standalone/web/rest-api/micro-service it will start from start class only. Start class is also called as main class in springboot.

-------------------------------------------------------------------------

-> Spring Boot run ( ) method will start IOC container based on starter available in pom.xml

spring-boot-starter ==> AnnotationConfigApplicationContext

spring-boot-starter-web => AnnotationConfigServletWebServerApplicationContext

spring-boot-starter-webflux => AnnotationConfigReactiveWebServerApplicationContext

----------------------------------------------------------------------------------

Note: If we have all the above 3 starters in pom.xml file then boot using below class to start IOC

AnnotationConfigServletWebServerApplicationContext

----------------------------------------------------------------------------------

-> Spring Boot application start class having one annotation i.e

@SpringBootApplication

-> This one annotation is equal to below 3 annotations

1) @SpringBootConfiguration

2) @EnableAutoConfiguration

3) @ComponentScan

-----------------------------------------------------------------------------------

Component Scanning

-----------------------------------------------------------------------------------

-> Component Scanning is the process of identifying spring bean classes available in the application

-> Component Scanning will start from base package

-> The package which contains start class is called as base package

-> After base package scanning got completed it will scan sub packages of base package.

-> Sub package means the package name which is starting with base package name

in.ashokit ------- base package

in.ashokit.dao

in.ashokit.service

in.ashokit.util

in.ashokit.config

com.ashokitech---------> This is not sub package

In Last session we dicussed about Spring Boot Start class

-> When we create Spring boot application, Application.java will be created by default that is called as start class of spring boot

-> Start class is the entry point for spring boot application execution

-> In start class we will have a main method which will call SpringApplication.run ( .. ) method

-> SpringApplication.run (..) method will perform below operations

-> Create Bootstrap Context

-> Run Listener classes

-> Load App Args

-> Prepare Env

-> Print Banner

-> Create IOC

-> Prepare & Refresh IOC

-> Call Runners

-> Return IOC obj

-> In Spring Boot programmer no need to create/start IOC because run ( ) method will start/create IOC based on starter available in pom.xml

-> Based on starter available in pom.xml ioc will be created

-> In spring boot start class we have one annotation i.e @SpringBootApplication

-> This one annotation is equal to below 3 annotations

1) @SpringBootConfiguration

2) @EnableAutoConfiguration

3) @ComponentScan

--------------------------------------------------------------------------

-> Component Scanning is the process of identifying spring bean classes available in the application.

-> Component Scanning will start from base package

-> The package which contains start class will be considered as base package by default

-> After base package scanning got completed it will scan sub packages of base package.

-------------------------------------------------------------------------

Q) Can we configure more than one base package in spring boot ?

Ans) Yes, it is possible using @ComponentScan annotation in start class like below

-------------------------------------------------------------------------

package in.ashokit;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.ConfigurableApplicationContext;

import org.springframework.context.annotation.ComponentScan;

@SpringBootApplication

@ComponentScan(basePackages = { "in.ashokit", "com.ashokitech" })

public class Application {

public static void main(String[] args) {

ConfigurableApplicationContext ctxt = SpringApplication.run(Application.class, args);

System.out.println(ctxt.getClass().getName());

}

}

----------------------------------------------------------------------------------

=> @EnableAutoConfiguration annotation will provide auto-configuration feature for our application

=> Auto-Configuration means it will identify the configuration which is required for our application and it will provide that configuration in run time.

Ex: providing embedded server for web application

creating connection pool

providing security features

loading beans and creating objects etc..

-----------------------------------------------------------------------------------

-> @SpringBootConfiguration annotation represents our class as Configuration class. Configuration class is the replacement for xml file based configuration

-> As we have this annotation for Spring Boot Start Class we can consider that as Configuration class and we can perfom configuration customization in the start class.

-----------------------------------------------------------------------------------

Interview Questions From Spring Boot Start Class

----------------------------------------------------------------------------------

1) What is start class in spring boot ?

2) what is @SpringBootApplication annotation?

3) What is Auto Configuration ?

4) What is Component Scanning ?

5) What is base package naming convention ?

6) Can we configure multiple base packages in spring boot ?

7) What is @ComponentScan annotation ?

8) What is the purpose of SpringApplication.run ( ) method in spring boot ?

9) Can u exaplin spring boot application execution flow ?

10) What is the return type of SpringApplication.run ( ) method ?

11) Who will create IOC in spring boot application ?

12) Which class is used to create IOC in spring boot ?

13) What is banner in spring boot ?

14) What is Runner in the spring boot ?

15) Is it mandatory to have start class for web application ?

Last session we discussed Spring Boot & Its advantages & internals

=> We discussed FAQ from Spring Boot start class

----------------------------------------------------------------------------------

Spring Boot Annotation

----------------------------------------------------------------------------------

@Component

@Service

@Repository

@Configurtion

@Bean

----------------------------------------------------------------------------------

-> @Component, @Service, @Repository these annotations are belongs to same family and they are used to represent java class as Spring Bean.

@Component : Used for any utility class

@Service : Used for service/business layer classes

@Repository : Used for persistence/dao layer classes

Note: Above annotations are class level annotations. When we use above annotations IOC container will create object for those classes by considering them as spring beans.

----------------------------------------------------------------------------------

-> @Configuration annotation is used to represent java class as configuration class

-> To perform any customized configurations we will use Config classes

-> If we want to customize object creation for a class then we will @Bean annotation at the method level.

-> @Bean annotation is method level annotation.

---------------------------------------------------------------------------------

Autowiring

---------------------------------------------------------------------------------

-> The process of injecting one class object into another class object is called as dependency injection.

-> When we are discussing about Spring basics we learnt Setter Injection & Constructor Injection

-> SI & CI we implemented manually using XML configuration

---------------------------------------------------------------------------------

<bean id="dieselEng" class="in.ashokit.beans.Engine" />

<bean id="car" class="in.ashokit.beans.Car">

<property name="eng" ref="dieselEng" />

</bean>

----------------------------------------------------------------------------------

-> In the above configuration we are informing to IOC which object should be injected into Car using "ref" attribute. This is called as manual wiring.

----------------------------------------------------------------------------------

-> Autowiring means IOC will identify dependent object and it will inject into target object (we no need to specify which object should be injected)

-> To enable auto-wiring we will use @Autowired annotation

-> @Autowired annotation we can use at below 3 places

a) setter method level (SI)

b) constructor level (CI)

c) field level (FI)

----------------------------------------------------------------------------------

public interface IEngine {

public int start();

}

---------------------------------------------------------------------------------

@Component

public class DieselEngine implements IEngine {

@Override

public int start() {

System.out.println("DieselEngine :: started");

return 1;

}

}

--------------------------------------------------------------------------------

@Component

public class Car {

private IEngine eng;

@Autowired

public void setEng(IEngine eng) {

this.eng = eng;

}

public void drive() {

int start = eng.start();

if (start == 1) {

System.out.println("journey started....");

}

}

}

----------------------------------------------------------------------------------

@SpringBootApplication

public class Application {

public static void main(String[] args) {

ConfigurableApplicationContext ctxt = SpringApplication.run(Application.class, args);

Car car = ctxt.getBean(Car.class);

car.drive();

}

}

---------------------------------------------------------------------------------

When we developed spring based applications we used "ref" attributed to specifiy what is dependent object that should be injected into target object

-> If we use Auto-wiring spring ioc will become and it will identify dependent object and it will inject that into target object

-> To enable Autowiring we will use @Autowired annotation

-> @Autowired annotation we can use at 3 places

a) setter method level (SI)

b) constructor level (CI)

c) field level (FI)

----------------------------------------------------------------------------------

-> Autowiring is the process of identifying dependent object and injecting that into target object.

-> Autowiring is working based on modes

a) byName

b) byType

-> byName means it will identify dependent object based on name of the bean

-> byType means it will identify dependent object based on data type

----------------------------------------------------------------------------------

Last session we discussed @Autowired annotation with setter & constructor injection.

-> When we use @Autowired annotation iOC container will identify dependent object and it will inject into target object

-> @Autowired annotation we can use at 3 places

1) setter method

2) constructor

3) field

-----------------------------------------------------------------------

-> When we use @Autowired annotation at setter method level then IOC will perform dependeny injection by calling setter method.

-> When we use @Autowired annotation at constructor then IOC will perform dependeny injection using constructor.

-> When we use @Autowired annotation at variable level then IOC will perform Field Injection.

------------------------------------------------------------------------

Q) How Field Injection will work internally ?

------------------------------------------------------------------------

-> IOC will use Reflection api to inject value to private varibale.

-------------------------------------------------------------------------

public class User {

private Integer age;

}

-------------------------------------------------------------------------

public class Demo {

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

Class<?> clz = Class.forName("in.ashokit.beans.User");

Field ageField = clz.getDeclaredField("age");

ageField.setAccessible(true);

Object clzObj = clz.newInstance();

ageField.set(clzObj, 20);

System.out.println(ageField.get(clzObj));

}

}

--------------------------------------------------------------------------

Runners In Spring Boot

--------------------------------------------------------------------------

-> Spring Boot Runners will execute once application got started

-> Runners will be called from SpringApplication.run (..) method

Note: If we have any requirement to execute our logic only one time when application got stated then we can use Runners for that.

-> In spring boot we have 2 types of Runners

1) ApplicationRunner

2) CommandLineRunner

Note : The above 2 are interfaces which contains run( ) method

Usecase : Load static from db tables to cache memory

-------------------------------------------------------------------------

Working with Banner in Spring Boot

-------------------------------------------------------------------------

-> When we run spring boot application, spring logo will be printed on the console that is called as Spring Boot banner.

-> For banner we have 3 modes

1) console

2) log

3) off

-> The default banner mode is console which will print banner in console

-> log mode means it will print banner in log file

-> off mode means banner will not be printed

-> We can set banner mode in application.properties file like below

spring.main.banner-mode=off

-> We can customize banner text also

-> Create banner.txt file inside src/main/resources folder and keep ascii text to customize banner text.

ASCII Text Generation URL : https://patorjk.com/software/taag/#p=display&f=Graffiti&t=Ashok%20IT

In yesterday's session we developed 2 web apps using Spring Boot

=> Today's session : Web app development

-------------------------------------------------------------------------

-> To create web application with spring boot we are using "spring-boot-starter-web" dependency

-> Web Starter supports for

1) web app development

2) rest api development

3) tomcat embedded server

-------------------------------------------------------------------------

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

<scope>runtime</scope>

<optional>true</optional>

</dependency>

<dependency>

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

<artifactId>tomcat-embed-jasper</artifactId>

</dependency>

-------------------------------------------------------------------------

-> spring-boot-devtools dependency is used for re-starting the server when changes happend in the project.

--------------------------------------------------------------------------

-> In every web application mainly we will have 2 types of functionalities

1) sending data from controller to ui

2) sending data from ui to controller

-------------------------------------------------------------------------

-> In Spring Web MVC we have Model interface to send data from controller to UI

-> Model will represent the data in the form of key-value pair

Key - String type

Value - Object type

----------------------------------------------------------------------------------

@Controller

public class WelcomeController {

@GetMapping("/welcome")

public String welcomeMsg(Model model) {

model.addAttribute("msg", "Welcome to Ashok IT Family...!!");

return "index";

}

}

---------------------------------------------------------------------------------

-> Model is an interface which is used to send data to ui in key-value pair

-> Model interace having BindingAwareModelMap as the implementation class

-> BindingAwareModelMap internally uses LinkedHashMap to store the data

--------------------------------------------------------------------------------

public class User {

private Integer userId;

private String username;

private String userEmail;

public User() {

// TODO Auto-generated constructor stub

}

public User(Integer userId, String username, String userEmail) {

super();

this.userId = userId;

this.username = username;

this.userEmail = userEmail;

}

-----------------------------------------------------------------------------------

@Controller

public class UserController {

@GetMapping("/user")

public String getUserData(Model model) {

User userObj = new User(101, "Ashok", "ashokitschool@gmail.com");

model.addAttribute("user", userObj);

return "user-data";

}

}

-----------------------------------------------------------------------------------

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"

pageEncoding="ISO-8859-1"%>

<!DOCTYPE html>

<html>

<head>

<meta charset="ISO-8859-1">

<title>Insert title here</title>

</head>

<body>

User Id : ${user.userId} <br/>

User Name : ${user.username} <br/>

User Email : ${user.userEmail} <br/>

</body>

</html>

-----------------------------------------------------------------------------------

Assignment : Develop a spring boot web application which will display list of users data in table format.

> In the last session, we discussed about how to send data from Controller to UI

=> To send data from Controller to UI we are using Model object.

=> Model will represent data in the form of key-value pair

------------------------------------------------------------------------------------------

Today's topic : Sending data from UI to controller

------------------------------------------------------------------------------------------

-> We can send data from UI to controller in below 3 ways

1) Query Parameters

2) Path Parameters

3) Request Body

---------------------------------------------------------------------------------------------

Query Parameters

----------------------------------------------------------------------------------------------

-> Query Parameters will represent data in the form of key-value pair

www.ashokitech.com/courses?name=sbms&trainer=ashok

-> Query Parameters will start with question mark (?) and will be seperated by and (&) symbol

-> Query Parameters always will present at end of the URL

-> To read query parameters from URL we will use @RequestParam annotation

---------------------------------------------------------------------------------------------

@Controller

public class WelcomeController {

@GetMapping("/welcome")

public String welcomeMsg(@RequestParam("name") String name, Model model) {

String msgTxt = name + ", Welcome to Spring Web MVC";

model.addAttribute("msg", msgTxt);

return "index";

}

}

-----------------------------------------------------------------------------------------------

Path Parameters (or) Path variables (or) URI Parameters

---------------------------------------------------------------------------------------------

-> Path Parameters are also used to send data to server in URL

-> Path Parameters will represent data directly in URL (No key)

-> Path Parameters can present anywhere in the URL

www.ashokitech.com/course/java

-> Path Parameters should be represented in URL pattern

-> To read path parameters from URL we will use @PathVariable annotation

-------------------------------------------------------------------------------------------

@Controller

public class WelcomeController {

@GetMapping("/welcome/{name}")

public String welcomeMsg(@PathVariable("name") String name, Model model) {

String msgTxt = name + ", Welcome to Spring Web MVC";

model.addAttribute("msg", msgTxt);

return "index";

}

}

----------------------------------------------------------------------------------------------

-> Both Query Params & Path Params data is visible in URL

-> It is not recommended to send sensitive data in URL like pwd, pin, otp etc...

-> Query Params & Path Params supports only text data (no support for binary data)

-> If we want to send any sensitive data then it is recommended to send using Request Body.

----------------------------------------------------------------------------------------------

Form Based Applications

-----------------------------------------------------------------------------------------------

-> Spring Web MVC Module provided Form Tag Library To develop the forms easily

-> Spring MVC form tag library supports for Form Binding

-> Form Binding means binding form data to object and object data to form.

object <======> form fields

-----------------------------------------------------------------------------------------------

Login Form Example

-----------------------------------------------------------------------------------------------

public class LoginForm {

private String email;

private String pwd;

//setters & getters

}

-----------------------------------------------------------------------------------------------

@Controller

public class LoginController {

@GetMapping("/login")

public String loginPage(Model model) {

LoginForm loginFormObj = new LoginForm();

model.addAttribute("loginForm", loginFormObj);

return "index";

}

@PostMapping("/checkLogin")

public String checkLogin(LoginForm formObj, Model model) {

if (formObj.getEmail().equals("ashok@gmail.com")

&& formObj.getPwd().equals("abc@123")) {

model.addAttribute("msg", "Welcome to Ashok IT application...");

return "dashboard";

} else {

model.addAttribute("msg", "Invalid Credentials");

return "index";

}

}

}

-----------------------------------------------------------------------------------------------

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"

pageEncoding="ISO-8859-1"%>

<%@ taglib uri="http://www.springframework.org/tags/form" prefix="form"%>

<!DOCTYPE html>

<html>

<head>

<meta charset="ISO-8859-1">

<title>Insert title here</title>

</head>

<body>

${msg}

<form:form action="checkLogin" modelAttribute="loginForm" method="POST">

<table>

<tr>

<td>Email :</td>

<td><form:input path="email" /></td>

</tr>

<tr>

<td>Password :</td>

<td><form:password path="pwd" /></td>

</tr>

<tr>

<td><input type="submit" value="Login" />

</tr>

</table>

</form:form>

</body>

</html>

----------------------------------------------------------------------------------------------

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"

pageEncoding="ISO-8859-1"%>

<!DOCTYPE html>

<html>

<head>

<meta charset="ISO-8859-1">

<title>Insert title here</title>

</head>

<body>

${msg}

<br />

<a href="login">Logout</a>

</body>

</html>

----------------------------------------------------------------------------------------------

server.port=9090

spring.mvc.view.prefix=/pages/

spring.mvc.view.suffix=.jsp

Last session : Login Form Development

Today's session : Registration Form Development

--------------------------------------------------------------------------

-> Spring Web MVC module provided Form Tag Library to simplify forms development

-> Below are the tags available in spring mvc form tag library

<form:form/> : To display a form

<form:input/> : To display a text field

<form:password/> : To display a password field

<form:radiobutton/> : To display single radio button

<form:radiobuttons/> : To display multiple radio buttons

<form:select/> : To display a dropdown/listbox

<form:option/> : To display one option for the dropdown

<form:options/> : To display multiple options for the dropdown

<form:checkbox/> : To display single checkbox

<form:checkboxes/> : To display mutliple checkboxes

<form:button/> : To display a button

<form:hidden/> : To represent hidden field

<form:error/> : To display error message (validation error)

-> To use these tags we have to specifiy taglib of spring mvc form tag library

--------------------------------------------------------------------------

Requirement: Develop Student Registration Form Using Spring MVC Form Tag Library

-------------------------------------------------------------------------

1) Create Spring Starter project with below dependencies

a)spring-boot-starter-web

b)tomcat-embed-jasper

c)spring-boot-devtools

2) Configure Server Port & View Resolver in application.properties file

3) Create Form Binding Class to represent form data

4) Create Controller class with below methods

a) method to load empty form (HTTP GET Request)

b) method to handle Register button (HTTP POST Request)

5) Create View Files

a)index.jsp (form display)

b)success.jsp (success msg display)

6) Run the application and test it

---------------------------------------------------------------------------------

public class Student {

private String name;

private String email;

private String gender;

private String course;

private String[] timings;

//setters & getters

}

----------------------------------------------------------------------------------

package in.ashokit.controller;

import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

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

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

import in.ashokit.binding.Student;

@Controller

public class StudentController {

@GetMapping("/student")

public String loadForm(Model model) {

model.addAttribute("student", new Student());

return "index";

}

@PostMapping("/saveStudent")

public String handleRegisterBtn(Student student, Model model) {

System.out.println(student);

String name = student.getName();

String msgTxt = name + ", Your Registration Completed Successfully..!!";

model.addAttribute("msg", msgTxt);

return "success";

}

}

-----------------------------------------------------------------------------------

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"

pageEncoding="ISO-8859-1"%>

<%@ taglib uri="http://www.springframework.org/tags/form" prefix="form"%>

<!DOCTYPE html>

<html>

<head>

<meta charset="ISO-8859-1">

<title>Insert title here</title>

</head>

<body>

<form:form action="saveStudent" modelAttribute="student" method="POST">

<table>

<tr>

<td>Name :</td>

<td><form:input path="name" /></td>

</tr>

<tr>

<td>Email :</td>

<td><form:input path="email" /></td>

</tr>

<tr>

<td>Gender :</td>

<td><form:radiobutton path="gender" value="Male"/>Male <form:radiobutton

path="gender" value="Fe-Male"/>Fe-Male</td>

</tr>

<tr>

<td>Course :</td>

<td><form:select path="course">

<form:option value="">-Select-</form:option>

<form:option value="java">Java</form:option>

<form:option value="python">Python</form:option>

<form:option value="devops">DevOps</form:option>

</form:select></td>

</tr>

<tr>

<td>Timings :</td>

<td>

<form:checkbox path="timings" value="morning"/> Morning

<form:checkbox path="timings" value="Evening"/> Evening

</td>

</tr>

<tr>

<td><input type="submit" value="Register"/></td>

</tr>

</table>

</form:form>

</body>

</html>

----------------------------------------------------------------------------------

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"

pageEncoding="ISO-8859-1"%>

<!DOCTYPE html>

<html>

<head>

<meta charset="ISO-8859-1">

<title>Insert title here</title>

</head>

<body>

${msg} <br/>

<a href="student">Go Back</a>

</body>

</html>

----------------------------------------------------------------------------------

In Last session we developed Registration Form Using Spring MVC Form Tag Library

-------------------------------------------------------------------------

Today's session : Layered Architecture Based App Development

-------------------------------------------------------------------------

-> In Realtime application will be divided into multiple layers

1) Presentation Layer (UI)

2) Web Layer

3) Business/Service Layer

4) Persistence Layer

-> Presentation layer contains User Interface Logic (UI)

-> Web Layer contains logic to handle request & response

-> Business/Service layer contains business logic & integration logic

-> Persistence Layer contains database communication logic

-------------------------------------------------------------------------

Requirement : Develop a web application to store and retrieve books details using Layered Architecture

Form Fields

------------

Book Name (text field)

Book Price (text field)

Book Author (text field)

Submit (Button)

View All Books (Hyperlink)

=> After entering the data in form, user will click on "Submit" button then book data should be stored into database table and success message should be displayed on the page.

=> When user click on "View All Books" hyperlink, retrieve books info from database table and display in web page in table format.

Steps to develop Book Store application with layered architecture

---------------------------------------------------------------------

1) Create spring starter project with below dependencies

a) spring-boot-starter-web

b) spring-boot-starter-datajpa

c) H2 (Embedded Database)

d) spring-boot-devtools

e) tomact-embed-jasper

f) JSTL

2) Configure below properties in application.properties file

a) embedded server port number

b) datsource properties

c) view resolver

3) Create Entity & Repository for DB communication

4) Create Service interface & implementation

5) Create Controller with required methods

6) Create view files

-------------------------------------------------------------------------

server.port=9090

spring.mvc.view.prefix=/pages/

spring.mvc.view.suffix=.jsp

spring.datasource.username=sa

spring.datasource.password=sa

spring.datasource.url=jdbc:h2:mem:testdb

spring.datasource.driver-class-name=org.h2.Driver

-------------------------------------------------------------------------

@Entity

@Table(name = "BOOK\_DTLS")

public class Book {

@Id

@GeneratedValue

@Column(name = "BOOK\_ID")

private Integer id;

@Column(name = "BOOK\_NAME")

private String name;

@Column(name = "BOOK\_PRICE")

private String price;

@Column(name = "BOOK\_AUTHOR")

private String author;

//setters & getters

-------------------------------------------------------------------------

public interface BookRepository extends JpaRepository<Book, Integer> {

}

-------------------------------------------------------------------------

public interface BookService {

public boolean saveBook(Book book);

public List<Book> getAllBooks();

}

--------------------------------------------------------------------------@Service

public class BookServiceImpl implements BookService {

@Autowired

private BookRepository bookRepo;

@Override

public boolean saveBook(Book book) {

Book save = bookRepo.save(book);

if (save.getId() != null) {

return true;

}

return false;

}

@Override

public List<Book> getAllBooks() {

return bookRepo.findAll();

}

}

--------------------------------------------------------------------------

@Controller

public class BookController {

@Autowired

public BookService bookService;

@GetMapping("/book")

public String loadForm(Model model) {

model.addAttribute("book", new Book());

return "index";

}

@PostMapping("/saveBook")

public String saveBook(Book book, Model model) {

boolean status = bookService.saveBook(book);

if (status) {

model.addAttribute("success", "Book Saved Successfully");

} else {

model.addAttribute("failure", "Failed to save book");

}

return "index";

}

@GetMapping("/viewBooks")

public String viewBooks(Model model) {

List<Book> allBooks = bookService.getAllBooks();

model.addAttribute("books", allBooks);

return "data";

}

}

--------------------------------------------------------------------------

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"

pageEncoding="ISO-8859-1"%>

<%@ taglib uri="http://www.springframework.org/tags/form" prefix="form"%>

<!DOCTYPE html>

<html>

<head>

<meta charset="ISO-8859-1">

<title>Insert title here</title>

</head>

<body>

<font color='green'>${success}</font>

<font color='red'>${failure}</font>

<form:form action="saveBook" modelAttribute="book" method="POST">

<table>

<tr>

<td>Book Name :</td>

<td><form:input path="name" /></td>

</tr>

<tr>

<td>Book Price :</td>

<td><form:input path="price" /></td>

</tr>

<tr>

<td>Book Author :</td>

<td><form:input path="author" /></td>

</tr>

<tr>

<td></td>

<td><input type="submit" value="Save" /></td>

</tr>

</table>

</form:form>

<a href="viewBooks">View All Books</a>

</body>

</html>

--------------------------------------------------------------------------

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"

pageEncoding="ISO-8859-1"%>

<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>

<!DOCTYPE html>

<html>

<head>

<meta charset="ISO-8859-1">

<title>Insert title here</title>

</head>

<body>

<a href="book">+Add New Book</a>

<table border="1">

<thead>

<tr>

<th>Name</th>

<th>Price</th>

<th>Author Name</th>

</tr>

</thead>

<tbody>

<c:forEach items="${books}" var="book">

<tr>

<td>${book.name}</td>

<td>${book.price}</td>

<td>${book.author}</td>

</tr>

</c:forEach>

</tbody>

</table>

</body>

</html>

-------------------------------------------------------------------------

Last session : Developed Layered Architecture Based Application

--------------------------------------------------------------------------

-> In realtime every application will be developed using layered architecture only

-> If we use layered architecture then our application will be loosely coupled

-> In Layered Architecture we can see below layers/tiers in the application

1) Presentation layer (UI)

2) Web Layer (Controllers)

3) Business/Service Layer (business logic)

4) Persistence Layer (DB communication logic)

--------------------------------------------------------------------------

-> Spring Web MVC

-> Advantages Of Spring Web MVC

-> Spring Web MVC Architecture

-> DispatcherServlet (FrontController)

-> HandlerMapper

-> Controller/RequestHandler

-> ModelAndView (data & view file name)

-> Model (I)

-> ViewResolver

-> View

-> Query Parameter (@RequestParam)

-> Path Parameter (@PathVariable)

-> Query Param Vs Path Param

-> Form Tag Library

-> Form Binding

-> Login Form App developement

-> Registration Form App development

-> Student Registration Form development

-> Layered Application development (Book Store App)

-----------------------------------------------------------------------------------

Students Doubts clarified

In Last Session : We revised Web MVC Concepts & Doubts cclarified

---------------------------------------------------------------------------------------------

Today's session : Thymleaf

--------------------------------------------------------------------------------------------

-> We used JSP as a presentation technology in our spring web mvc based applications

-> JSP can't be executed in browser directley

-> When the request comes to JSP then internally JSP will be converted to Servlet and that servlet will send response to browser

-> When we use JSP for presentation then burden will be increased on server because every JSP should be converted into Servlet to produce the response.

---------------------------------------------------------------------------------------------

-> To overcome problems of JSP we can use Thymeleaf as a presentation technology

-> Thymleaf is a template engine that can be used in HTML pages directley

-> HTML pages can be executed in browser directley

-> In general, HTML pages are used for static data. If we use thymleaf in HTML then we can add dynamic nature to HTML pages.

---------------------------------------------------------------------------------------------

-> We can develop spring boot application with thymleaf as a presentation technology

-> To use Thymleaf in spring boot we have a starter

spring-boot-starter-thymleaf

----------------------------------------------------------------------------------------------

Procedure to develop spring boot application with thymeleaf

---------------------------------------------------------------------------------------------

1) Create Spring Starter Project with below dependencies

a) web-starter

b) thymeleaf-starter

c) devtools

2) Create Controller with required methods

3) Create Theymeleaf templates in src/main/resources/templates folder (file extension .html)

4) Configure Embedded Server Port number in application.properties file

5) Run the application and test it

Note: No need to configure view resolver because Spring Boot will detect theymeleaf template files and will process them

-----------------------------------------------------------------------------------------------

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-thymeleaf</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

<scope>runtime</scope>

<optional>true</optional>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

---------------------------------------------------------------------------------------------

@Controller

public class WelcomeController {

@GetMapping("/welcome")

public String welcomeMsg(@RequestParam("name") String name, Model model) {

String msgTxt = name + ", Welcome to Ashok IT..!!";

model.addAttribute("msg", msgTxt);

return "index";

}

}

-----------------------------------------------------------------------------------------------

<!DOCTYPE html>

<html xmlns:th="http://www.thymeleaf.org">

<head>

<meta charset="ISO-8859-1">

<title>Insert title here</title>

</head>

<body>

<p th:text=${msg} />

</body>

</html>

----------------------------------------------------------------------------------------------

server.port=9090

--------------------------------------------------------------------------------------------

Today's Assignment: Develop a form based application using thymeleaf.

Form Fields

-----------

Product ID, Product Name, Product price

-> When user submit the form capture form data and display that in another page

----------------------------------------------------------------------------------------------

Today's session : Form based application development using Spring Boot + Thymeleaf

===============================================================================================

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

<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 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>2.6.2</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<groupId>in.ashokit</groupId>

<artifactId>24-SB-Web-MVC-Thymeleaf-Form-App</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>24-SB-Web-MVC-Thymeleaf-Form-App</name>

<description>Demo project for Spring Boot</description>

<properties>

<java.version>1.8</java.version>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-thymeleaf</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

<scope>runtime</scope>

<optional>true</optional>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

-----------------------------------------------------------------------------------------------

package in.ashokit.binding;

public class Product {

private Integer pid;

private String pname;

private Double price;

public Integer getPid() {

return pid;

}

public void setPid(Integer pid) {

this.pid = pid;

}

public String getPname() {

return pname;

}

public void setPname(String pname) {

this.pname = pname;

}

public Double getPrice() {

return price;

}

public void setPrice(Double price) {

this.price = price;

}

@Override

public String toString() {

return "Product [pid=" + pid + ", pname=" + pname + ", price=" + price + "]";

}

}

-----------------------------------------------------------------------------------------------

package in.ashokit.controller;

import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

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

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

import in.ashokit.binding.Product;

@Controller

public class ProductController {

@GetMapping("/product")

public String loadForm(Model model) {

Product productObj = new Product();

model.addAttribute("product", productObj);

return "index";

}

@PostMapping("/saveProduct")

public String saveProduct(Product product, Model model) {

System.out.println(product);

model.addAttribute("product", product);

return "data";

}

}

-----------------------------------------------------------------------------------------------

<!DOCTYPE html>

<html xmlns:th="https://www.thymeleaf.org">

<head>

<meta charset="ISO-8859-1">

<title>Insert title here</title>

</head>

<body>

<form th:action="@{/saveProduct}" th:object="${product}" method="POST">

<table>

<tr>

<td>Product Id:</td>

<td><input type="text" th:field="\*{pid}" /></td>

</tr>

<tr>

<td>Product Name:</td>

<td><input type="text" th:field="\*{pname}" /></td>

</tr>

<tr>

<td>Product Price:</td>

<td><input type="text" th:field="\*{price}" /></td>

</tr>

<tr>

<td></td>

<td><input type="submit" value="Save" /></td>

</tr>

</table>

</form>

</body>

</html>

-----------------------------------------------------------------------------------------------

<!DOCTYPE html>

<html xmlns:th="https://www.thymeleaf.org">

<head>

<meta charset="ISO-8859-1">

<title>Insert title here</title>

</head>

<body>

Product Id :

<p th:text="${product.pid}" />

Product Name :

<p th:text="${product.pname}" />

Product Price:

<p th:text="${product.price}" />

<a href="product">Go Back</a>

</body>

</html>

-----------------------------------------------------------------------------------------------

Spring Boot + Thymeleaf + Form validations - Example

-----------------------------------------------------------------------------------------------

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

<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 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>2.6.2</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<groupId>in.ashokit</groupId>

<artifactId>25-SB-Web-MVC-Form-Validations</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>25-SB-Web-MVC-Form-Validations</name>

<description>Demo project for Spring Boot</description>

<properties>

<java.version>1.8</java.version>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-thymeleaf</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-validation</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

<scope>runtime</scope>

<optional>true</optional>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

-----------------------------------------------------------------------------------------------

package in.ashokit.binding;

import javax.validation.constraints.Min;

import javax.validation.constraints.NotNull;

import javax.validation.constraints.Size;

public class Person {

@NotNull

@Size(min = 3, max = 8)

private String name;

@NotNull

@Min(18)

private Integer age;

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public Integer getAge() {

return age;

}

public void setAge(Integer age) {

this.age = age;

}

@Override

public String toString() {

return "Person [name=" + name + ", age=" + age + "]";

}

}

-----------------------------------------------------------------------------------------------

package in.ashokit.controller;

import javax.validation.Valid;

import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

import org.springframework.validation.BindingResult;

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

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

import in.ashokit.binding.Person;

@Controller

public class PersonController {

@GetMapping("/person")

public String displayForm(Model model) {

Person personObj = new Person();

model.addAttribute("person", personObj);

return "index";

}

@PostMapping("/savePerson")

public String savePerson(@Valid Person person, BindingResult result, Model model) {

System.out.println(person);

if (result.hasErrors()) {

return "index";

}

model.addAttribute("msg", person.getName() + " record saved successfully");

return "data";

}

}

-----------------------------------------------------------------------------------------------

<!DOCTYPE html>

<html xmlns:th="https://www.thymeleaf.org">

<head>

<meta charset="ISO-8859-1">

<title>Insert title here</title>

</head>

<body>

<form th:action="@{/savePerson}" th:object="${person}" method="POST">

<table>

<tr>

<td>Name :</td>

<td><input type="text" th:field="\*{name}" /></td>

<td th:if="${#fields.hasErrors('name')}" th:errors="\*{name}">Name

Error</td>

</tr>

<tr>

<td>Age :</td>

<td><input type="text" th:field="\*{age}" /></td>

<td th:if="${#fields.hasErrors('age')}" th:errors="\*{age}">Age

Error</td>

</tr>

<tr>

<td></td>

<td><input type="submit" value="Save" /></td>

</tr>

</table>

</form>

</body>

</html>

-----------------------------------------------------------------------------------------------<!DOCTYPE html>

<html xmlns:th="https://www.thymeleaf.org">

<head>

<meta charset="ISO-8859-1">

<title>Insert title here</title>

</head>

<body>

<p th:text="${msg}" />

<a href="person">Go Back</a>

</body>

</html>

----------------------------------------------------------------------------------------------

4-1-2022 notes completed