Spring Boot

- Wrapping of all spring modules into fewer modules
- It makes application easier to configure & use all the other modules
- It is helpful in speeding up the application development
- It helps in developing production ready application rapidly
- It uses convention over configuration
- Follow opiniated defaults
- Spring boot will scan thru the class path
- Merges different annotations into one or fewer (example @SpringBootApplication – merges of @Configuration, @EnableAutoConfiguration, @ComponentScan, etc)

“Spring Boot Starter” Projects:

- ✓ It helps to build respective projects in the application
- ✓ Add this to Maven Dependency, it automatically brings all jar files that are required to build the project
- ✓ Examples: Spring-boot-starter-web, Spring-boot-starter-data, Spring-boot-starter-data-jpa, etc
- ✓ “Bill of Material” (BOM) – used by Maven internally to pull dependant jar files for each starter project

How Spring Boot works?

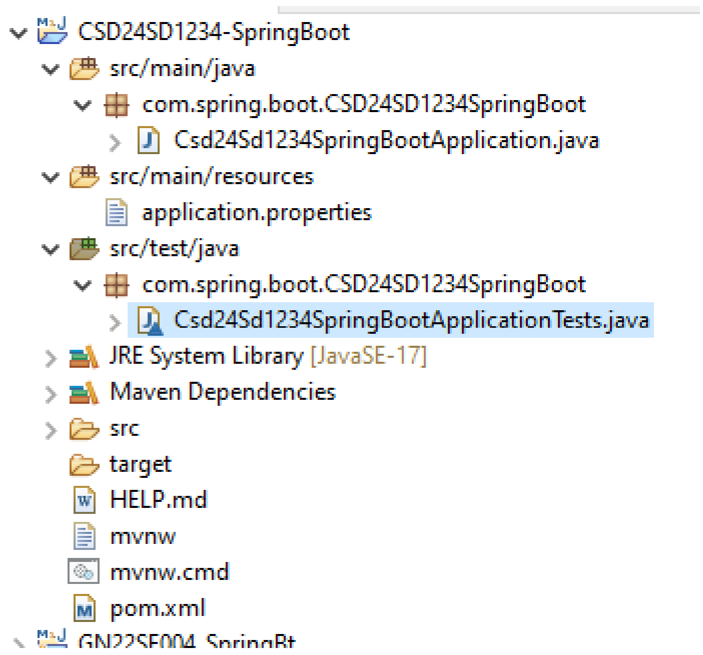
- No code or XML – instead of user creating config, Spring has already config files ready to be used
- Spring Starter POM will add jars
- All configurations are marked with @Configuration

Four ways to create a SpringBoot project:

1. Create a Maven project & add the starter dependencies
2. Use the “Spring intializr” tool
3. Using IDE Support, like STS
4. Using SpringBoot CLI

Example 1 (for Spring Boot)

Folder Structure:


```
CSD24SD1234-SpringBoot
  src/main/java
    com.spring.boot.CSD24SD1234SpringBoot
      Csd24Sd1234SpringBootApplication.java
  src/main/resources
    application.properties
  src/test/java
    com.spring.boot.CSD24SD1234SpringBoot
      Csd24Sd1234SpringBootApplicationTests.java
  JRE System Library [JavaSE-17]
  Maven Dependencies
  src
  target
  HELP.md
  mvnw
  mvnw.cmd
  pom.xml
GN22SE004 SpringBt
```

Pom.xml

```
<?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.o
rg/POM/4.0.0
https://maven.apache.org/xsd/maven-
4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>

    <groupId>org.springframework.boot</groupI
d>
        <artifactId>spring-boot-starter-
parent</artifactId>
        <version>3.2.3</version>
        <relativePath/> <!-- lookup parent
from repository -->
```

```xml
    </parent>
    <groupId>com.spring.boot</groupId>
    <artifactId>CSD24SD1234-
SpringBoot</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>CSD24SD1234-SpringBoot</name>
    <description>Demo project for Spring
Boot</description>
    <properties>
        <java.version>17</java.version>
    </properties>
    <dependencies>
        <dependency>

    <groupId>org.springframework.boot</groupI
d>
            <artifactId>spring-boot-
starter</artifactId>
        </dependency>

        <dependency>

    <groupId>org.springframework.boot</groupI
d>
            <artifactId>spring-boot-starter-
test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
    <groupId>org.springframework.boot</groupI
d>
                <artifactId>spring-boot-maven-
plugin</artifactId>
            </plugin>
        </plugins>
    </build>

</project>
```

Main Class Initially:

```
package
com.spring.boot.CSD24SD1234SpringBoot;

import
org.springframework.boot.SpringApplication;
import
org.springframework.boot.autoconfigure.Spring
BootApplication;

@SpringBootApplication
public class Csd24Sd1234SpringBootApplication
{

    public static void main(String[] args) {

    SpringApplication.run(Csd24Sd1234SpringBo
otApplication.class, args);
    }

}
```

Test Class Initially:

```
package
com.spring.boot.CSD24SD1234SpringBoot;

import org.junit.jupiter.api.Test;
import
org.springframework.boot.test.context.SpringB
ootTest;

@SpringBootTest
class Csd24Sd1234SpringBootApplicationTests {

    @Test
    void contextLoads() {
    }

}
```

After making modifications…

Student Class:

```
package
com.spring.boot.CSD24SD1234SpringBoot.DAO;

import
org.springframework.stereotype.Component;

@Component
public class Student {

    public void create() {
        System.out.println("Student Record
created");
```

```
    }
}
```

Student Service class:

```
package
com.spring.boot.CSD24SD1234SpringBoot.Service
;

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

import
com.spring.boot.CSD24SD1234SpringBoot.DAO.Stu
dent;

@Component
public class StudentService {

    Student stu;

    @Autowired
    StudentService(Student stu){
        System.out.println("Student Service
Created");
        this.stu = stu;
    }

    public void save(){
        stu.create();
    }
}
```

Test Class After changes:

```
package
com.spring.boot.CSD24SD1234SpringBoot;

import org.junit.jupiter.api.Test;
import org.junit.runner.RunWith;
import
org.springframework.beans.factory.annotation.
Autowired;
import
org.springframework.boot.test.context.SpringB
ootTest;
import
org.springframework.context.ApplicationContex
t;
import
org.springframework.test.context.junit4.Sprin
gRunner;

import
com.spring.boot.CSD24SD1234SpringBoot.Service
.StudentService;

@RunWith(SpringRunner.class)
@SpringBootTest
class Csd24Sd1234SpringBootApplicationTests {

    @Autowired
    ApplicationContext context;
    @Test
    void Test1() {
        StudentService ss =
context.getBean(StudentService.class);
        ss.save();
    }

}
```

Output:

```
14:08:10.138 [main] INFO
org.springframework.test.context.support.AnnotationConfigContextLoaderUtil
s -- Could not detect default configuration classes for test class
[com.spring.boot.CSD24SD1234SpringBoot.Csd24Sd1234SpringBootApplicationTes
ts]: Csd24Sd1234SpringBootApplicationTests does not declare any static,
non-private, non-final, nested classes annotated with @Configuration.
14:08:10.342 [main] INFO
org.springframework.boot.test.context.SpringBootTestContextBootstrapper --
Found @SpringBootConfiguration
com.spring.boot.CSD24SD1234SpringBoot.Csd24Sd1234SpringBootApplication for
test class
com.spring.boot.CSD24SD1234SpringBoot.Csd24Sd1234SpringBootApplicationTest
s

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v3.2.3)

2024-03-08T14:08:11.344+05:30  INFO 3156 --- [           main]
.C.Csd24Sd1234SpringBootApplicationTests : Starting
Csd24Sd1234SpringBootApplicationTests using Java 17.0.1 with PID 3156
(started by windows in E:\Java\CSD24SD1234-SpringBoot\CSD24SD1234-
SpringBoot)
2024-03-08T14:08:11.346+05:30  INFO 3156 --- [           main]
.C.Csd24Sd1234SpringBootApplicationTests : No active profile set, falling
back to 1 default profile: "default"
Student Service Created
2024-03-08T14:08:12.613+05:30  INFO 3156 --- [           main]
.C.Csd24Sd1234SpringBootApplicationTests : Started
Csd24Sd1234SpringBootApplicationTests in 1.886 seconds (process running
for 4.146)
```

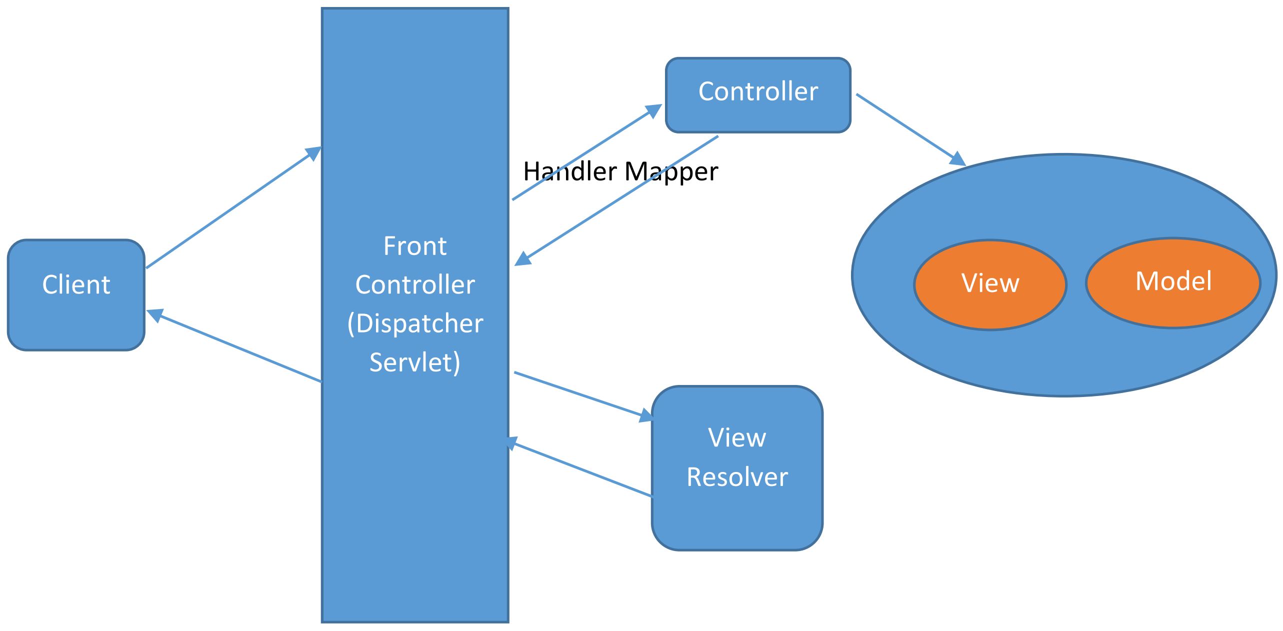
```
Student Record created
```

Spring MVC:

MVC – Model View Controller, where Model is the data & View is the web page

It is used to design dynamic web applications

It internally uses 3 different design patterns: Front Controller, Handler Mapper & View Resolver



Flow & various components involved:

Step 1: When a HTTP request comes from client, the first component to handle the request is "Dispatcher Servlet". DS is an implementation of Front Controller. It is configured in web.xml, which is a deployment descriptor

Step 2: DS will use "Handler Mapper" to invoke a Controller; Controller is a POJO class that we create & mark with @Controller

Step 3: Controller will implement a method that creates a Model & View (Model is optional to keep data; view is the next page to be displayed)

Step 4: At the end, Controller will return the Model & View back to DS; Name of the View along with data will be shared with DS

Step 5: DS will take the view name & will invoke "View Resolver" (a component)

Step 6: VR will append a prefix (location) & suffix (Extension) to the view name and returns a complete view back to DS; This way controller is not coupled with a view; views are stored in webInf pages

Step 7: DS will take that view & render the page back to Client

Pre-requisite:

1. Create Maven Project with SpringWebMVC dependency
2. Tomcat Server has to be installed (Download TomCat server & plug in to Eclipse)

To install "TomCat" server:

Google "Tomcat Server" & download the zip file; Unzip the folder;
Go to Eclipse -> open the "Server" tab -> click "create a new server" -> pick tomcat latest version & click "Next" -> find the folder downloaded & click "Finish"

How to create a Spring Application:

Step 1: Configure the Dispatcher Servlet in web.xml

Step 2: Create Spring Configuration file (under web-inf folder)

Step 3: Configure View Resolver

Step 4: Create any Java classes as needed

Step 5: Create Controller class

Step 6: create a folder structure & views needed (under Web-INF)

Format used is / like below

Domainname/folder/folder/filename

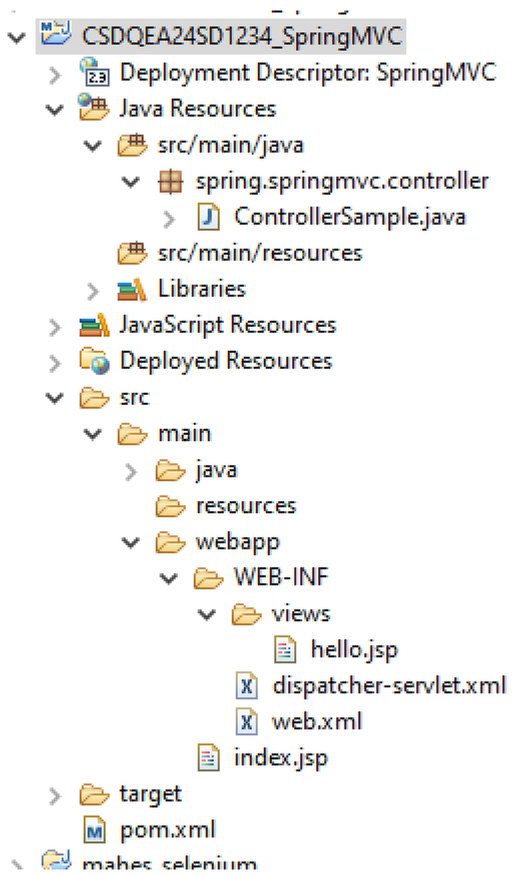
View name: hello

After appending prefix & suffix:

/WEB-INF/views/hello.jsp

<u>Example for SpringMVC:</u>

<u>Folder Structure</u>:


```
CSDQEA24SD1234_SpringMVC
  Deployment Descriptor: SpringMVC
  Java Resources
    src/main/java
      spring.springmvc.controller
        ControllerSample.java
    src/main/resources
    Libraries
  JavaScript Resources
  Deployed Resources
  src
    main
      java
      resources
      webapp
        WEB-INF
          views
            hello.jsp
          dispatcher-servlet.xml
          web.xml
        index.jsp
  target
  pom.xml
mahes_selenium
```

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/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>sample.spring.mvc</groupId>
  <artifactId>CSDQEA24SD1234_SpringMVC</artifactId>
  <packaging>war</packaging>
  <version>0.0.1-SNAPSHOT</version>

  <name>CSDQEA24SD1234_SpringMVC Maven
Webapp</name>
  <url>http://maven.apache.org</url>

  <properties>
```

```
    <project.build.sourceEncoding>UTF-
8</project.build.sourceEncoding>

<maven.compiler.source>1.7</maven.compiler.source>

<maven.compiler.target>1.7</maven.compiler.target>
  </properties>

  <dependencies>
    <dependency>
     <groupId>org.springframework</groupId>
     <artifactId>spring-webmvc</artifactId>
     <version>5.3.18</version>
    </dependency>

    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
  <build>
    <finalName>CSDQEA24SD1234_SpringMVC</finalName>

    <pluginManagement><!-- lock down plugins
versions to avoid using Maven defaults (may be
moved to parent pom) -->
      <plugins>
        <plugin>
          <artifactId>maven-compiler-
plugin</artifactId>
          <version>3.10.1</version>
          <configuration>
        <source>1.8</source>
        <target>1.8</target>
          </configuration>
        </plugin>
      </plugins>
    </pluginManagement>

  </build>
</project>
```

Default jsp file (index.jsp):

```
<html>
<body>
<h2>Hello World!</h2>
</body>
</html>
```

Web.xml

```
<!DOCTYPE web-app PUBLIC
 "-//Sun Microsystems, Inc.//DTD Web Application
2.3//EN"
 "http://java.sun.com/dtd/web-app_2_3.dtd" >

<web-app>
  <display-name>SpringMVC</display-name>

  <servlet>
    <servlet-name>dispatcher</servlet-name>
    <servlet-
class>org.springframework.web.servlet.DispatcherSer
vlet</servlet-class>
  </servlet>

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

Dispatcher-Servlet.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/schem
a/context"


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

http://www.springframework.org/schema/beans/spring-
beans-3.0.xsd

http://www.springframework.org/schema/context

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

    <context:component-scan base-
package="spring.springmvc.controller"></context:com
ponent-scan>

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

```
</beans>
```

Controller class

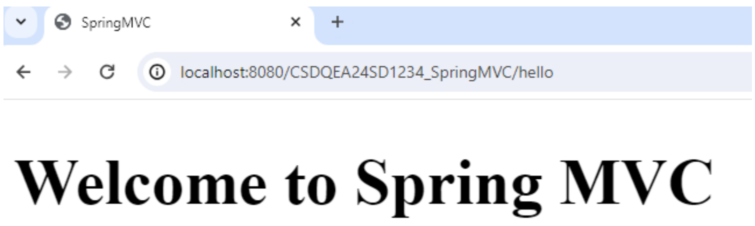
```
package spring.springmvc.controller;

import org.springframework.stereotype.Controller;
import
org.springframework.web.bind.annotation.RequestMapp
ing;
import
org.springframework.web.servlet.ModelAndView;

@Controller
public class ControllerSample {

    @RequestMapping("/hello")
    public ModelAndView hello(){
        ModelAndView mv = new ModelAndView();
        mv.setViewName("hello");
        return mv;
    }
}
```

Hello.jsp

```
<%@ page language="java" contentType="text/html;
charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=ISO-8859-1">
<title>SpringMVC</title>
```

```
</head>
<body>
    <h1>Welcome to Spring MVC</h1>
</body>
</html>
```

Output:


SpringMVC

localhost:8080/CSDQEA24SD1234_SpringMVC/hello

Welcome to Spring MVC

How it works?

1. From browser, request goes to web application
2. In web.xml, we have configured the Dispatcher Servlet, to handle everything that comes to web application
3. DS will take the request & dispatches it to @Controller via Handler Mapper
4. Controller returns the ModelandView where it is configured with view (example: hello)
5. DS will then invoke ViewResolver & will pass on the view name
6. VS will append prefix & suffix and return the complete view back to DS
7. DS will render the jsp page back to web browser as a response

URI (uniform resource indicator) => Request Mapping name / id

Where URI is part of URL

<u>Data Exchange:</u>

1. Controller to UI
2. UI to Controller

<u>Controller to UI:</u>

- Data is passed to ModelandView object
- It uses addObject(key,value) method to set data (key- java string, value-object)
- Data is accessed in jsp page using request.getAttribute("key") method – here "request" is http servlet request object; value will be accessed using "key"
- Object – may be of Primitive / String, Object or Collection

a) **sending Primitive & String based data**

Update the previous Controller class:

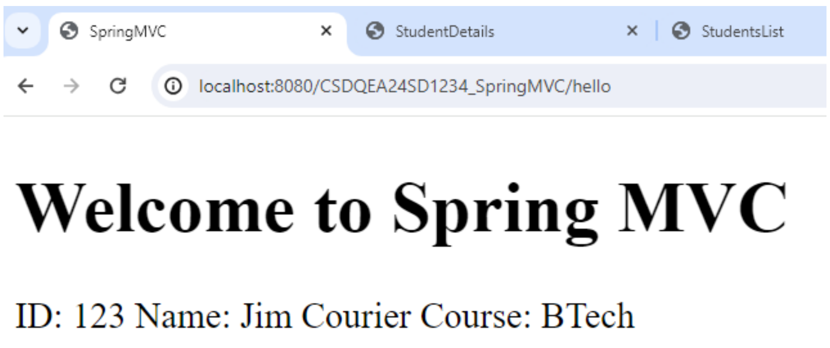
```
package spring.springmvc.controller;

import org.springframework.stereotype.Controller;
import
org.springframework.web.bind.annotation.RequestMapp
ing;
import
org.springframework.web.servlet.ModelAndView;

@Controller
public class ControllerSample {

    @RequestMapping("/hello")
    public ModelAndView hello(){
        ModelAndView mv = new ModelAndView();
        mv.setViewName("hello");
        mv.addObject("id", 123);
        mv.addObject("name", "Jim Courier");
        mv.addObject("course", "BTech");
        return mv;
    }
}
```

Update previous hello.jsp:

```
<%@ page language="java" contentType="text/html;
charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
```

```
<meta http-equiv="Content-Type" content="text/html;
charset=ISO-8859-1">
<title>SpringMVC</title>
</head>
<body>
    <h1>Welcome to Spring MVC</h1>
    <%
        Integer id =
(Integer)request.getAttribute("id");
        String name =
(String)request.getAttribute("name");
        String course =
(String)request.getAttribute("course");
        out.println("ID: " + id);
        out.println("Name: " + name);
        out.println("Course: " + course);
    %>
</body>
</html>
```

Restart the server & check Output:



Welcome to Spring MVC

ID: 123 Name: Jim Courier Course: BTech

b) sending Object data type

Create a new Model class:

```
package spring.springmvc.data.entities;
```

```
public class Student {

	private int id;
	private String name;
	private String course;

	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 getCourse() {
		return course;
	}
	public void setCourse(String course) {
		this.course = course;
	}

	@Override
	public String toString() {
		return "Student [id=" + id + ", name=" +
name + ", course=" + course + "]";
	}
}
```

Create new controller class:

```
package spring.springmvc.controller;
```

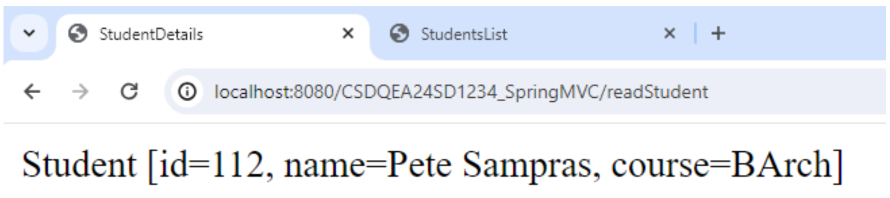
```
import org.springframework.stereotype.Controller;
import
org.springframework.web.bind.annotation.RequestMapp
ing;
import
org.springframework.web.servlet.ModelAndView;

import spring.springmvc.data.entities.Student;

@Controller
public class StudentController {

    @RequestMapping("/readStudent")
    public ModelAndView sendStudent(){
        ModelAndView mv1 = new ModelAndView();
        mv1.setViewName("displayStudent");

        Student stu = new Student();
        stu.setId(112);
        stu.setName("Pete Sampras");
        stu.setCourse("BArch");
        mv1.addObject("student", stu);
        return mv1;
    }
}
```

Create new jsp file:

```
<%@ page language="java" contentType="text/html;
charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
```

```
<meta http-equiv="Content-Type" content="text/html;
charset=ISO-8859-1">
<title>StudentDetails</title>
</head>
<body>
    <%=request.getAttribute("student")%>
</body>
</html>
```

Restart the server & check Output:



```
Student [id=112, name=Pete Sampras, course=BArch]
```

c) sending Collection Data Type

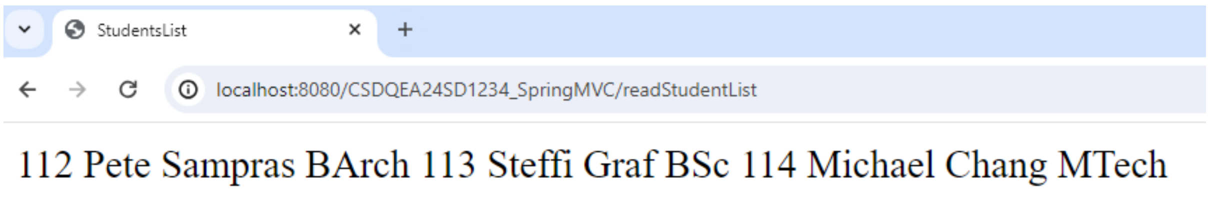
Create new controller class:

```
package spring.springmvc.controller;

import java.util.ArrayList;

import org.springframework.stereotype.Controller;
import
org.springframework.web.bind.annotation.RequestMapp
ing;
import
org.springframework.web.servlet.ModelAndView;

import spring.springmvc.data.entities.Student;

@Controller
public class StudentListController {
    @RequestMapping("/readStudentList")
    public ModelAndView sendStudentList(){

        ModelAndView mv2 = new ModelAndView();
        mv2.setViewName("displayStudentList");

        Student stu1 = new Student();
        stu1.setId(112);
        stu1.setName("Pete Sampras");
        stu1.setCourse("BArch");

        Student stu2 = new Student();
        stu2.setId(113);
        stu2.setName("Steffi Graf");
        stu2.setCourse("BSc");

        Student stu3 = new Student();
        stu3.setId(114);
        stu3.setName("Michael Chang");
        stu3.setCourse("MTech");

        ArrayList<Student> studList = new
ArrayList<Student>();
        studList.add(stu1);
        studList.add(stu2);
        studList.add(stu3);

        mv2.addObject("students", studList);
        return mv2;
    }
}
```

Create new jsp file:

```
<%@ page language="java" contentType="text/html;
charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"
    import="java.util.List,
spring.springmvc.data.entities.Student"
     %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=ISO-8859-1">
<title>StudentsList</title>
</head>
<body>
    <%
        List<Student> students =
(List<Student>)request.getAttribute("students");
        for(Student s:students){
            out.println(s.getId());
            out.println(s.getName());
            out.println(s.getCourse());
        }
    %>
</body>
</html>
```

Restart the server & check Output:



```
112 Pete Sampras BArch 113 Steffi Graf BSc 114 Michael Chang MTech
```

Sending Data from UI to Controller:

Done in 2 ways:

a) HTML form
b) Query Parameters

HTML Form:

1. Form data is submitted from web browser
2. The Spring Container does 4 activities:
    a. Read – Reads all the data using Request.getParameter method
    b. Convert – Convers into appropriate java data type using integer.parseInt, etc
    c. CreatesObject – create object of the model class
    d. SetsValues - sets the values that comes into the object
    e. Return – Handover the object after setting values to controller
3. Data is accessed in Controller class as an object

How to action it?

1. Define a Model class with properties needed (the number of properties & their name should match the # of fields & name in the HTML form)
2. Create a form (jsp) file matching the data members needed
3. Container will read the data comes in, creates object of the model class, set the values & hands over back to controller
4. To read the values, use @ModelAttribute in the method parameters

Example:

Case Study:

(i) Application will have a Student Form
(ii) Student will access the form & fill data and submit the form
(iii) Student will get success message or will get the submitted data back displayed in the web page

New Jsp page:

```
<%@ page language="java" contentType="text/html;
charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
```

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=ISO-8859-1">
<title>Student Form</title>
</head>
<body>
    <form action="registerStudent" method="post">
        <pre>
            ID : <input type="text" name="id"/>
            Name : <input type="text" name="name"/>
            Course : <input type="text"
name="course"/>
            <input type="submit" name="register">
        </pre>
    </form>
</body>
</html>
```

New Controller class:

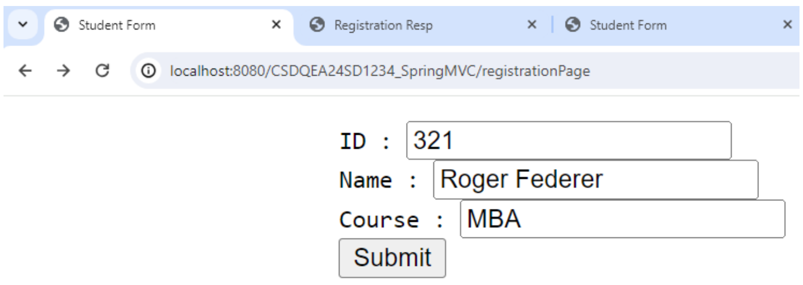
```
package spring.springmvc.controller;

import org.springframework.stereotype.Controller;
import
org.springframework.web.bind.annotation.ModelAttrib
ute;
import
org.springframework.web.bind.annotation.RequestMapp
ing;
import
org.springframework.web.bind.annotation.RequestMeth
od;
import
org.springframework.web.servlet.ModelAndView;

import spring.springmvc.data.entities.Student;
```

```
@Controller
public class StudentRegController {

	@RequestMapping("/registrationPage")
	public ModelAndView showRegPage(){

		ModelAndView mv3 = new ModelAndView();
		mv3.setViewName("studentRegistration");
		return mv3;
	}

	@RequestMapping(value="/registerStudent",
method=RequestMethod.POST)
	public ModelAndView
registerStudent(@ModelAttribute("student") Student
student){

		System.out.println(student);
		ModelAndView mv4 = new ModelAndView();
		mv4.setViewName("studentRegistration");
		return mv4;
	}
}
```

Restart the server & check Output:

Output 1:

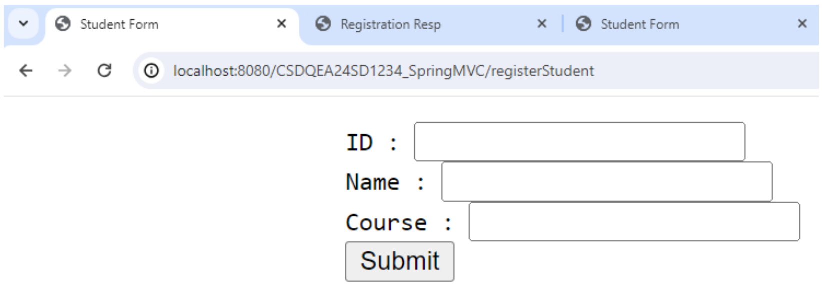
(showing the form & data filled)


Student Form | Registration Resp | Student Form

localhost:8080/CSDQEA24SD1234_SpringMVC/registrationPage

ID : 321
Name : Roger Federer
Course : MBA
Submit

(with same page as response)



ID :
Name :
Course :
Submit

Output in console:

```
INFO: Completed initialization in 2372 ms
Student [id=321, name=Roger Federer, course=MBA]
```

New Jsp page for response:

```
<%@ page language="java" contentType="text/html;
charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=ISO-8859-1">
<title>Registration Resp</title>
</head>
<body>
    Student Registered Successfully !!! Student
Details are ...
    <%=request.getAttribute("student") %>
</body>
</html>
```

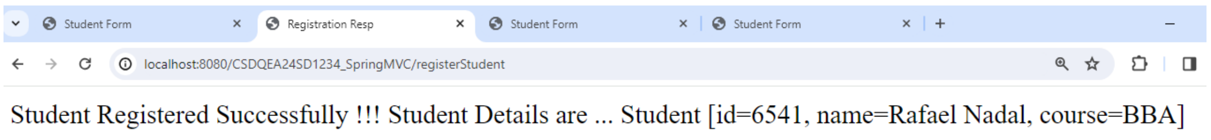
Make change in Controller class to include the new jsp page:

```
@RequestMapping(value="/registerStudent",
method=RequestMethod.POST)
```

```
    public ModelAndView
registerStudent(@ModelAttribute("student") Student
student){

        System.out.println(student);
        ModelAndView mv4 = new ModelAndView();
//      mv4.setViewName("studentRegistration");
        mv4.setViewName("studentRegResp");
        mv4.addObject("student", student);
        return mv4;
```

Restart the server & check Output:

Output 2: (with a new response page)



```
Student Registered Successfully !!! Student Details are ... Student [id=6541, name=Rafael Nadal, course=BBA]
```

Query Parameters:

- By using Request Parameters or Query Parameters
- Data is sent by appending at the end of URL using query string (?key=value; more than one key-value pair separated with & symbol)
- Data is retrieved in the controller class by using @RequestParam -> give the key & spring container will retrieve the value and set it in controller method parameter (@RequestParam(“key”)DT methodParametername)
- Controller usually receives data as String & will then have to be converted into respective data type
- If invalid data received, spring will throw error message (400 error)

New Controller class:

```
package spring.springmvc.controller;

import org.springframework.stereotype.Controller;
import
org.springframework.web.bind.annotation.RequestMapp
ing;
```

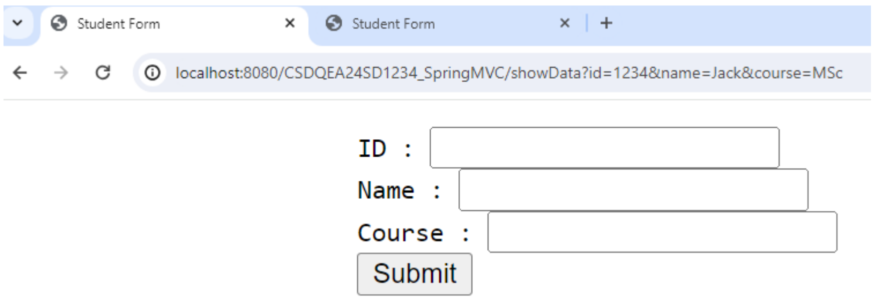
```
import
org.springframework.web.bind.annotation.RequestPara
m;
import
org.springframework.web.servlet.ModelAndView;

@Controller
public class RequestParamController {

    @RequestMapping("/showData")
    public ModelAndView
showData(@RequestParam("id") int id,
            @RequestParam("name") String name,
            @RequestParam("course") String course){

        System.out.println("Id : " + id);
        System.out.println("Name : " + name);
        System.out.println("Course : " + course);
        return new
ModelAndView("studentRegistration");
    }
}
```

Build the Query Parameter:

http://localhost:8080/CSDQEA24SD1234_SpringMVC/showData?id=1234&name=Jack&course=MSc

Run the URL & query in the browser

Output in the browser:


Student Form
Student Form
localhost:8080/CSDQEA24SD1234_SpringMVC/showData?id=1234&name=Jack&course=MSc
ID :
Name :
Course :
Submit

Output in the Console:

```
INFO: Completed initialization in 2372 ms
Id : 1234
Name : Jack
Course : MSc
```

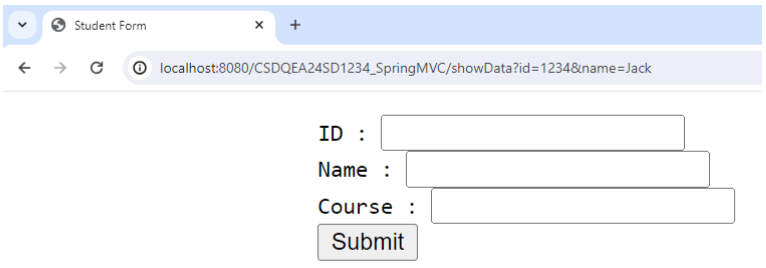
Make change in controller class to keep a data optional:

```
package spring.springmvc.controller;

import org.springframework.stereotype.Controller;
import
org.springframework.web.bind.annotation.RequestMapp
ing;
import
org.springframework.web.bind.annotation.RequestPara
m;
import
org.springframework.web.servlet.ModelAndView;

@Controller
public class RequestParamController {

    @RequestMapping("/showData")
    public ModelAndView
showData(@RequestParam("id") int id,
            @RequestParam("name") String name,
            @RequestParam(value="course",
required=false, defaultValue="BTech") String
course){

        System.out.println("Id : " + id);
        System.out.println("Name : " + name);
        System.out.println("Course : " + course);
        return new
ModelAndView("studentRegistration");
    }
}
```

Run the URL & query in the browser

http://localhost:8080/CSDQEA24SD1234_SpringMVC/showData?id=1234&name=Jack

Output in the browser:


Student Form
localhost:8080/CSDQEA24SD1234_SpringMVC/showData?id=1234&name=Jack
ID :
Name :
Course :
Submit

Output in the Console:

```
INFO: Completed initialization in 2372 ms
Id : 1234
Name : Jack
Course : BTech
```