Spring Framework

Earlier we were using struts -> web development, ejb -> for enterprise development, and jpa -> for database

Spring provides all of this.

Spring epub -> online e-book to learn

Spring context dependency contains spring core also.

Spring boot -> convention over configuration

Before application context came into picture we had BeanFactory to create bean.

BeanFactory factory = XmlBeanFactory(resource);

JVM -> Spring Container -> Spring Bean

By default spring follows singleton design pattern.

Bean scope:

prototype	Scopes a single bean definition to any number of object instances.

Prototype follows lazy initialisation.

<property *name*="age" *value*="10"/>
Does setter injection
So name should be same as name of setter (setAge) and not setAge1. If it is setAge1 then name should be age1.

@Primary as annotation can be mentioned in bean's definition as
<bean id ="" class ="" primary ="true"> </bean>

For spring jdbc -> add jdbc api dependency.

JdbcTemplate belongs to springframework.jdbc.core which helps in setting the datasource.

To work with h2 database,
We have to define the schema for table in application.properties and also to initialise we have to write it in data.sql.
To make spring boot understand where they are place,
Place schema.sql (containing create commands), and data.sql (containing insert command) inside resources folder.

JdbcTemplate has update method which takes sql query such as create, delete, update.
It has query method that takes sql query and RowMapper implemented class.

RowMapper is an interface which has method mapRow(ResultSet rs, int rowNum) throws SQLException.

We have to write the code for one row and jdbctemplate will call for each row.

mapRow returns object of same class.

So first set the properties of the object using resultSet and then return the object.

Real goal of using spring is to create web services (or endpoints).

Spring MVC:
Add dependency of spring web in spring boot.

To work with jsp pages, we have to create it like src/main/webapp/index.jsp

To run jsp page :

Put under src/main/java/webapp/view/index.jsp

In application.properties, put
spring.mvc.view.prefix=/webapp/view/
spring.mvc.view.suffix=.jsp

Putting injdex.jsp under src/main/webapp/WEB-INF/views/index.jsp,
Makes it private as it will only be used by controllers.
Anything inside webapp is public by default.
But WEB-INF is private.

So above method we are getting rid of extension (.jsp) and also we are able to restrict access.

Instead of running tomcat will download the page, it does not have capability to run the page… so we need tomcat jasper.

Jsp gets converted into servlet and then run on tomcat. Here nothing is happening. So here it is not happening.

There is other way to get values from form or post url…

```
@RequestMapping(“/add”, method=RequstMethod.POST)
public String homeController(HttpRequestServlet req) {
        int x = Integer.parseInt(req.getParameter(“num1”));
        int y = Integer.parseInt(req.getParameter(“num2”));
        int num3 = x + y;
        HttpSession session = req.getSession();
        session.setAttribute(“num3”, num3);
        return “result.jsp”
}

@RequestMapping(“/add”, method=RequstMethod.POST)
public String homeController(@RequestParam(“num1”)  int i,
@RequestParam(“num2”) int j, HttpSession session) {
        int num3 = I + j;
        session.setAttribute(“num3”, num3);
        return “result.jsp”
}
```

There is another concept of modelAndView

```
@RequestMapping(“/add”, method=RequstMethod.POST)
public ModelAndView homeController(@RequestParam(“num1”)  int i,
@RequestParam(“num2”) int j) {
        ModelAndView mv = new ModelAndView();
        mv.setViewName(“result.jsp”); // ModelView mv = new
ModelView(“result”);
        int num3 = I + j;
        mv.addObject(“num3”, num3);
        return mv;
}
```

There is another way of using model interface

```
@RequestMapping(“/add”, method=RequstMethod.POST)
public ModelAndView homeController(@RequestParam(“num1”)  int i,
@RequestParam(“num2”) int j, Model model) {
        int num3 = I + j;
        mv.addAttribute(“num3”, num3);
        return ‘result.jsp”;
}
```

If we replace Model with ModelMap then also it will work. ModelMap supports map format, anything added gets added in map format. So first it gets added in map and then filled up in model object.

To put data from a form inside an object, we can use @ModelAttribute over the object, it also adds the data in the model directly.

```
@RequestMapping("/addAlient", method=RequstMethod.POST)
public ModelAndView homeController(@ModelAttribute Alien a) {
        return 'result.jsp";
}
```

For this we have to use same name "a" or "alien" (name used by spring for any class) inside the result.jsp.

To change the convention, we can mention as @ModelAttribute("a1"), so now we can access as a1 inside result.jsp.

But the code can work without @ModelAttribute

```
@RequestMapping("/addAlient", method=RequstMethod.POST)
public ModelAndView homeController(Alien a) {
        return 'result.jsp";
}
```
But we have to use name "alien" in result.jsp

We can use @ModelAttribute over the method which will make spring to call it before any controller.

```
@ModelAttribute
public void modelData(Model m) {
        m.addAttribute("name", "Aliens");
}
```

Creating MVC application without spring boot:

1. First we have to configure the tomcat server.
2. Add dependency of spring web mvc

2 problems now have to be solved, tomcat will directly run index.jsp page under webapp and not under web-inf/views, second we have to divert the request to the controller.

3. Now we have to setup the dispatcher servlet

This web.xml exists under web-inf folder. So whenever request is made it is transferred to dispatcher servlet.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://
xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
        version="4.0">

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

    <servlet-mapping>
        <servlet-name>dispatcher</servlet-name>
        <url-pattern>/</url-pattern> <!—This makes it to listen at every url —>
    </servlet-mapping>

</web-app>
```

If we run now we will get file not found exception (/web-inf/dispatcher-servlet.xml).
This file is used to configure our dispatcher servlet.

4. Now we have to make sure dispatcher servlet transfers control to respective controller.

We have to add the dispatcher-servlet.xml under web-inf folder.

```
<?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:mvc="http://www.springframework.org/schema/mvc"
       xsi:schemaLocation="
           http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans.xsd
           http://www.springframework.org/schema/mvc
           http://www.springframework.org/schema/mvc/spring-mvc.xsd">

    <!-- Configure component scanning -->
    <context:component-scan base-package="com.example.controller" />

    <!-- Configure view resolver -->
    <bean id="viewResolver"
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/views/" /> <!— the views are kept under
```

```
webapp/views —>
        <property name="suffix" value=".jsp" />
    </bean>

    <!-- Enable annotation-driven MVC -->
    <mvc:annotation-driven />

</beans>
```

——————————————————————————————————————

The values filled up in form gets added to the url on clicking post using ?(Character)

But if we do not want it then in form we can specify method=post….