Uttara InfoSolutions

www.uttarainfo.com

Steps to build SpringMVC App:

SpringMVC FirstApp Steps:

0) Import SpringWelcomeApp.war in Eclipse->File->Import->Web -> give downloaded SpringWelcomeApp.war file path
1) Eclipse->File->New->Dy web project -> FirstApp
2) Copy paste jar files from SpringWelcomeApp->WEB-INF->lib to FirstApp->WEB-INF->lib.
Copy paste web.xml and spring-servlet.xml from SpringWelcomeApp->WEB-INF to FirstApp->WEB-INF (overwrite).
Create a folder named jsps in WEB-INF of FirstApp.

Observe the contents of the 2 xmls. web.xml must contain
a) mapping of / url pattern to fully q.c.n of DispatcherServlet
b) DS should be <load-on-startup> enabled
c) <servlet-name> should be "spring". This name is used by default to pick up spring-servlet.xml from web-inf.

servlet-spring.xml should contain:
a) component scanning being turned on
b) mvc-annotation being turned on
c) InternalResourceViewResolver bean being configured
with prefix and suffix properties being injected.
Note that the prefix should point to "/WEB-INF/jsps"
Recollect when this will be used.

3) Create HomeController.java. Create a no-arg constructor with SOP

Annotate it with @Controller above the class// what does this do?

Create a method thus:

```
@RequestMapping("/") // what does this do?
public String showHome()
{
	System.out.println("in showHome");
	return "Home";
}
```

4) Create Home.jsp in jsps folder with <h1>Home Page</h1> content.

5) Right click on FirstApp and run on server. Do you see the Home page being served?

Observations:

1)You should see HomeController being instantiated during server startup in console (SOP of constructor)
2)when you send request to FirstApp as URL in browser, you should see showHome() SOP being printed. Recollect the steps of core workflow of Spring MVC (given in this document step by step below)
a) First DispatcherServlet executed
b) DS invokes Handler by passing URI. Handler picks which controller and which method in controller to execute based on annotations pre-configured
c) DS invokes our controllers method identified by handler
d) Our controller returns String to DS
e) DS invokes ViewResolver (IRVR) which using prefix and suffix returns path to DS
f) DS forwards to JSP.

Steps continue…

6) Create Register.jsp with <h1>Register</h1> only. In Home.jsp for hyperlink, submit request to /openRegisterView
7) Create RegBean class with String uname,email,pass,rpass as inst variables. Generate setters/getters and equals(),hashCode().toString().
6) In RAC, add method

```
	@RequestMapping("/openRegisterView")  // What is this?
	public String showRegisterView(Model model) // what is Model?
	{
		System.out.println("in RAC->showRegisterView()");
		RegBean bean = new RegBean();
		model.addAttribute("reg", bean); // why are we doing this?
```

```
            return “Register”;
      }
```

7) Test control flow. Register view should be displayed. If not, there is a problem with control flow. Ask the Lab Instructor. Go through the console and see if there is any exception and try to debug.

8) In Register.jsp:

```
<sp:form action="register" modelAttribute=“reg">
<!- do you understand why modelAttribute is used? ->

            Enter name: <sp:input path="uname"/><sp:errors path="uname"/><br/>
            Enter email: <sp:input path="email"/><sp:errors path="email"/><br/>
            Enter password: <sp:password path="pass"/><sp:errors path="pass"/
><br/>
            Repeat password: <sp:password path="rpass"/><sp:errors path="rpass"/
><br/>
            Enter dob: <sp:input path="dob"/><sp:errors path="dob"/><br/>
            Upload pic: <input type="file" name="pic"/><br/>

            <br/>

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

      </sp:form>
```

Go through the contents of the JSP and verify if you understand it.

9) In Controller, add

```
@RequestMapping("/register")
      public String register(@ModelAttribute("reg") @Valid RegBean
bean,BindingResult result,Model model)
      {
            System.out.println("in RAC->register() bean = "+bean);
            if(result.hasErrors())
            {
                  System.out.println("in RAC->register() result of bean
validation failed! result = "+result);
                  return "Register";
            }
            else
            {
                  //invoke service and ask it to store bean data to db

                  return "Success";
```

```
        }
}
```

Steps that occur when the WS starts:

1) WS reads web.xml of app
2) Because of load-on-startup for DS, WS loads, instantiates and initializes DS.
When DS is initialized (init() is invoked by WS):
    i) starts Spring Container (SpC)
    ii) SpC reads spring-servlet.xml
        a) our package classes are scanned for annotations and instantiated and wired together (Controller class inst only once)
        b) IRVR, etc beans are inst

When a request is sent for a SpringMVC app:

1) WS invokes DS->service in new TOEx cuz of /->DS in web.xml
2) DS consults Handler (DefaultAnnotationHandler). Handler decides on which Controller, which method based on req uri!
3) DS invokes controllers method and passes whatever the method arg list asks for!
4) Controller method returns control string to DS
5) DS consults IRVR and irvr adds prefix + cs + suffix and returns to DS
6) DS forwards to path of view!
7) WS executes JSP (and spring tag backing java code if present) and writes dy.html generated to response and sends it to client
8) client browser renders html to user