Creating the maven Project: where it contains webapp(is the root folder for all UI related things)

creating the groupId and ArtifactId in application level:

groupId: is nothing but package ( com.vtalent.springmvc)

ArtifactId: VtalentSpringMVC

On the WebApp maven structure: It should contains four source folder

src/main/java -> we are creating all java classes

src/main/resource -> we are creating the confg files like .xml

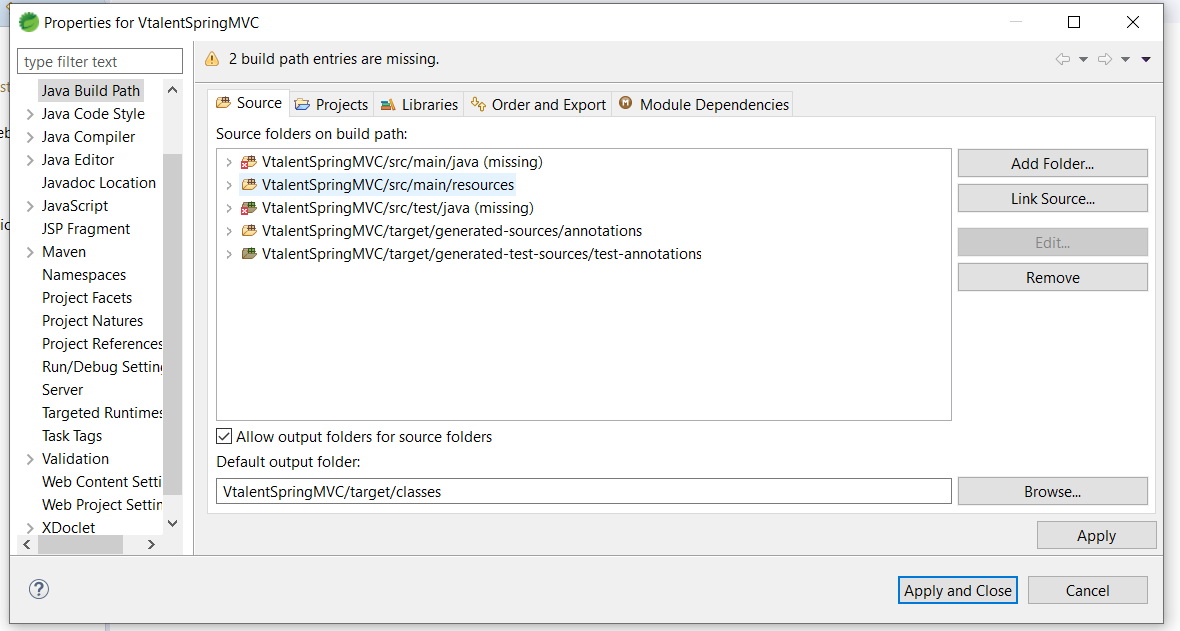
src/test/java -> test classes nothing JUnit

src/test/resource -> test class related confg files like .xml (levl of JUnit)

Note: because of running test classes on your main method we need a seperate config files.

**When the source folders are missing in the Application:**

* Go to the build path, see the below screen shot where your in source tab.
* Remove the missing folder in source tab.
* Add the new source folder in your java resource.



Add the dependency jar’s in pom.xml

* Add all spring related IOC, JDBC, MVC dependency jar’s from maven repository.
* POM : Project Object Model
* After adding all the dependencies, you need to update and clean the maven structure.

**SetUp the workspace:**

* Please check in build path where it contains any unbounded jar are their, please add or modify your local workspace related jars.
* For the maven first we need to update the project.
* Clean the maven project, runAs -> maven clean
* Install the maven project , runas -> maven install
* After the clean and install, mavev will download all dependency jars in your local drive, the path will be C:users/.m2/repository/
* Please make sure that the maven dependency is added in your project or application leval.

@) in Build path

@) In Deployment Assembly

* Maven install error:

No compiler is provided in this environment. Perhaps you are running on a JRE rather than a JDK?

[INFO] 1 error

The error indicates to change the environment JDK.

* Because of the above point if you miss you’ll get the class not found exception.

**Setp by Step procedure for spring mvc application configuration:**

* Created the Home.jsp page, to open this we need to go for Spring Artech level.
* We need to open the Home.jsp page on the intial of project running.
* Intial URL: <http://localhost:8080/VtalentSpringMVC/>

Localhost: local IP machine 127.0.0.1

8080: Tomcat port number

VtalentSpringMVC : context name or application name

/: config name or request name

/ is also called as default request

* Configure the spring mvc application.
* Configuring the DispatcherServlet in web.xml and dispatcher cfg file also we need to load in web.xml
* Make configuration of dispatcher with controller and view rresolver and all class related injections, UI related injections, Hibernate or springJDBC connection related injections etc.,
* In configuration first we need to import all the schema related to the frameworks.
* We need to inject controller scan with contextComponentScan tag.
* Let me create a controller package and create the controller class to accept the request from the DispatcherServlet.
* @Controller is used to make our class as controller.
* @RequestMapping is used to handle the request or config value from the URL and it should used in method level.
* Please make sure that all the @RequestMapping values are handling the unique request.
* We need to configure the ViewResolver in spring config file to display the view page on the browser.
* Let me create a registration page where I inject on Hom.jsp to open my registration form.
* <http://localhost:8080/VtalentSpringMVC/openingAnEmployeeReg>
* **openingAnEmployeeReg** is an config name or request name for that form
* To Read the form values and that values to be insert into the DB, we will configure Hibernate.
* **Hibernate:**

Hibernate is the framework for internal JDBC and SpringJDBC.

There will be two cache levels on hibernate:

1. SessionFactory
2. Session

**SessionFactory:**

* This is an application sessionFactory object.
* We are creating a DB connection with DriverManagerDataSource object.
* We are injecting Entity classes.
* Entity class: it is nothing but an encapsulation class were we can make class as an Entity with the help of @Entity annotation.
* Hibernate queries are not dependent on your DB (MySql).
* Hibernate will depend on Entity classes.
* Configuration of Entity classes.
* Entity class we will map to DB table with the help of @Table annotation.
* What are all the fields we have in entity classes all we are mapping to table columns with help of @column annotation, this one mainly used when the class properties and table fields are different.
* This queries will return in as HQL queries, Hibernate Query Language.
* Now all the entity classes will injected in SessionFactory.

1. **Session:**

* Session object will be opened from sessionfactory and this reference will be used in module level.
* We will close session once the module execution is done.

**Note:**  SessionFactory object will destroy once application is in down or stop.

**Hibernate Core steps:**

1. injecting the DB in your applicationContext.xml
2. to load the configuration file we need to use **Configuration** class.
3. With reference of configuration we need to create the sessionfactory object.
4. With the reference of sessionfactory we are creating the session object.

Ex: Session session = sessionFactory.openSession.

1. with the help of session we will begin the transaction.

@AutoWired: this annotation is used to get a reference from your container and it will inject into the class whatever you mapped.

Ex: @AutoWired Employee emp;

Spring form tags:

Contains a additional attribute to read the form values to server level.

modelAttribute: it is an attribute in your spring form where we will give the ref, so that all the form values will scan the inputs to inject into my modelAttribute object. It will done by internally calling the setter methods of that bean class.

How the setter methods will be called: if your employee class property and form level input name attribute should be same.

* To read the modelAttribute object from your form level, we need to use @ModelAttribute annotation in your method level.

ModelAndView is the object were I can combine the view page and the object value to pass to that page.