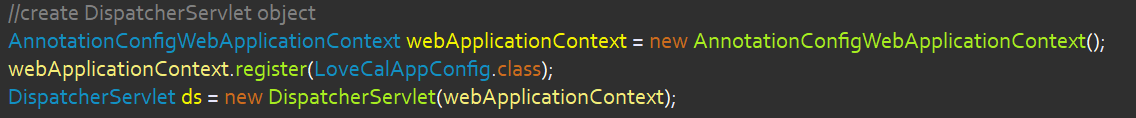
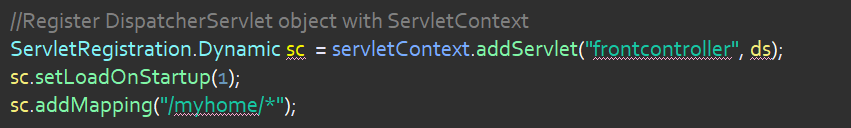
In the last tutorial, we used **LCApplicationInitializer** for initializing our DispatcherServlet which was implementing WebApplicationInitializer

In this class we were doing the following works :

* Implementing the **onStartup()** method
* Creating DispatcherServlet Object inside **onStartup()**



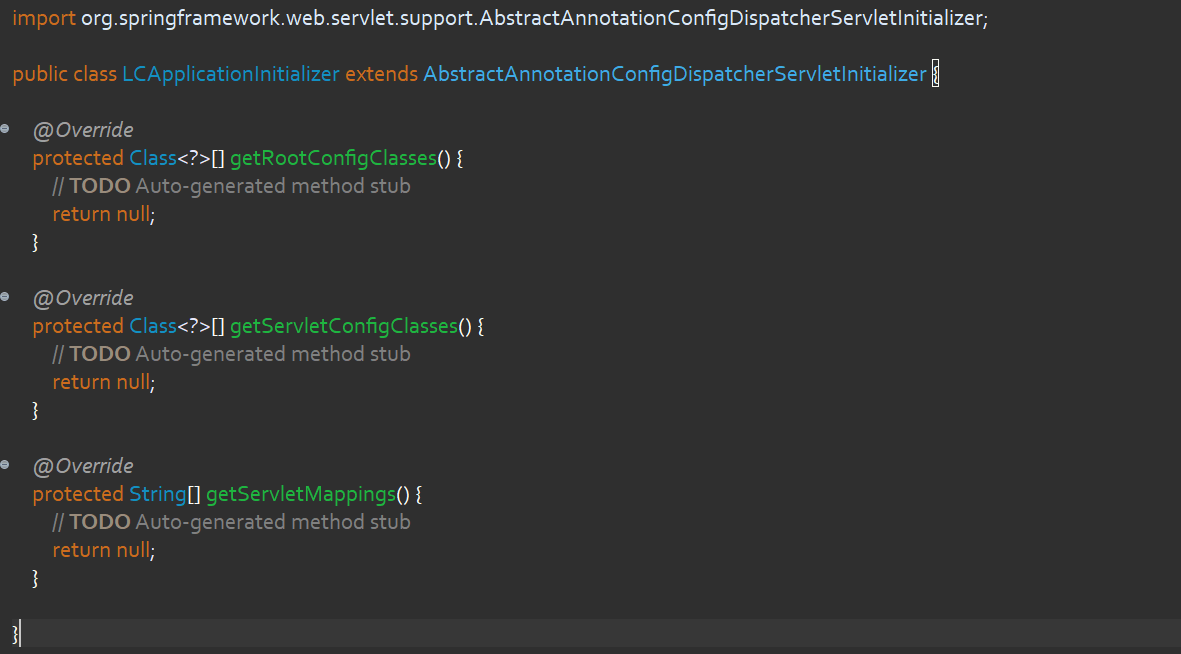
* Registering DispatcherServlet with ServletContext, mapping and setting load on startup.



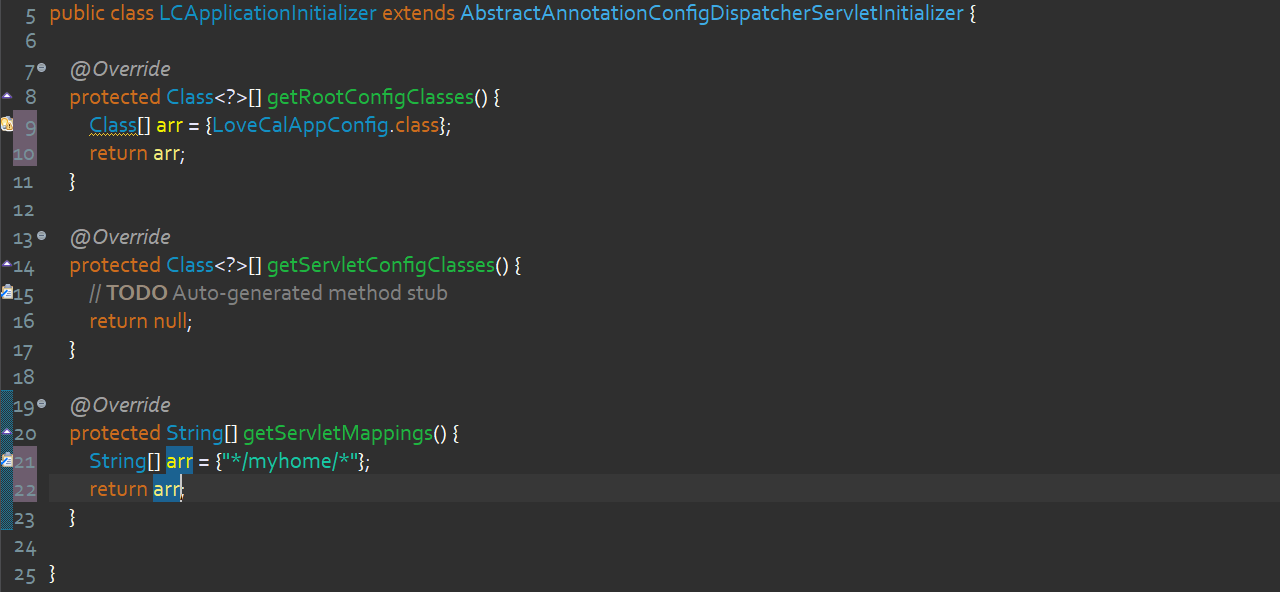
**Isn’t it very long way ?**

Let’s learn a new way of Configuring the DiapatcherServlet.

To minimize the code, You have to extends an abstract class i.e. AbstractAnnotationConfigDispatcherServletInitializer and implement all it’s abstract method like below :



Now make these changes in the above code :



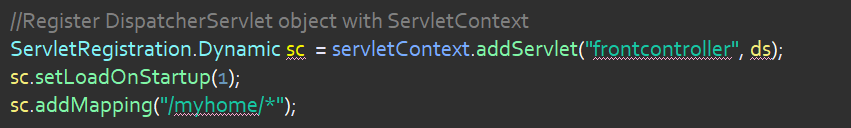
Now a question should arise in your mind. If we have this Abstract Class way then why we studied the long way of using Interface ?

Answer is : Internally AbstractAnnotationConfigDispatcherServletInitializer class does the same thing what we have done by implementing the WebApplicationInitializer Interface.

If you explore this class further then you will find that indirectly we are implementing the WebApplicationInitializer

The only benefit to use interface approach is that we have more control while configuring DispatcherServlet where we don’t have much control in case of Abstract Class approach.

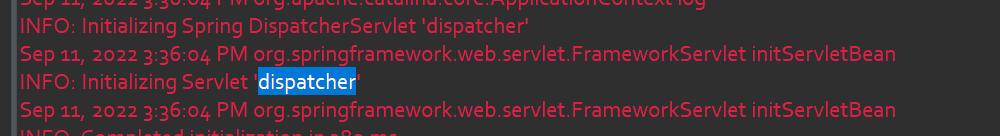
**If you remember while creating and registering DispatcherServlet object we assign it a name i.e. frontcontroller**



And this name was being reflected inside the console while tomcat initialize DispatcherServlet.

So the question is that when we use AbstractAnnotationConfigDispatcherServletInitializer we don’t mention any name. Is it anonymous or what ?

Answer is : Internally, this class uses a default name for it i.e. **dispatcher** and you must find this name into the console.



**Deleting web.xml :**

Until now, we didn’t delete the web.xml while we were not using it at all. But it’s time to delete it.

When you delete this file, your tomcat will still complain you that your project don’t have web.xml

This is because there is default property called **failOnMissingWebXml** set to **true.** We have to override this property by setting it up to **false.** What you need to do is, copy the below xml code and paste into your **pom.xml**

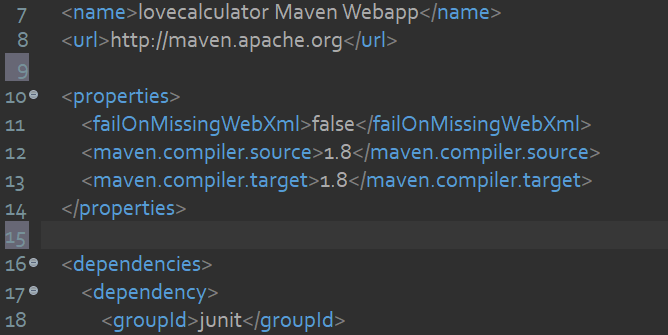
<properties>

<failOnMissingWebXml>false</failOnMissingWebXml>

<maven.compiler.source>1.8</maven.compiler.source>

<maven.compiler.target>1.8</maven.compiler.target>

</properties>



Now before execute the project, go to **Project / Maven / Update Project.**

Still getting any error try to stop the tomcat server and do **clean** then **clean tomcat working directory** by right clicking on your server.