ITL 3 PRACTICAL - 10

**AIM**

**Understand working of framework- Struts, Spring – Case study**

**Struts Framework**

**The Struts Framework is an open-source Web application framework used for creating Java Enterprise Edition Web applications quickly and efficiently. It makes use of and further extends the Java Servlet API to promote the model-view-controller (MVC) architecture.  
  
The Struts Framework was originally developed by Craig McClanahan and was then given to the Apache Foundation in May of 2000 under the Apache Jakarta Project and became known as Jakarta Struts. It eventually became a top-level Apache project in 2005 and was eventually replaced by Struts 2, which was released in February 2007.**

**The Struts Framework made exclusive use of the MVC design paradigm, and its goal was to separate the "model," which is the application logic that communicates with the database, from the "view," which is the HTML pages presented to the client/user, and from the "controller," which is the instance that passes information between the model and the client's view. Struts already provides the controller, which is just a Java servlet known as ActionServlet that creates the templates to be displayed by the view. It is then the job of the Web application programmer to create the model code and, by extension, the central configuration file called "struts-config.xml," which binds together the model, view and controller.**

**How Struts Framework works?**

**Struts framework implements MVC architecture, which divides the application components into Model, View and Controller parts. In this tutorial you will learn How Struts works with the help of an easy and simplified Struts Execution Flow Diagram.**

**Struts - Open source MVC Framework**

**Struts is the Open source web application Framework to create enterprise-ready Java web applications and has been designed in such a way that has made creation of web applications in Java quite simple and easy. Struts is based on Model View Framework (MVC) architect, which has been developed using WebWork and Xwork.**

**Here are the details of each components of Struts MVC framework.**

**Controller maps the user request to specific action. In Struts StrutsPrepareAndExecuteFilter acts as Controller. Controller receives the user request and decides which action to invoke.**

**Struts framework creates an instance of this action, invoke interceptors (if any). Then it calls invoke() method of ActionInvocations that executes the action. ActionInvocation then calls the intercept() method. The intercept() method of ActionInvocation class in turn calls the invoke() method of the ActionInvocation till all the interceptors are invoked. Interceptor are invoked before and after the action is executed. Interceptors are executed in the order they are defined in the stack. Finally action itself is invoked and the result is generated. All the interceptors are then invoked again but this time in the reverse order.**

**Model contains the data and the business logic and is implemented by the Action component.**

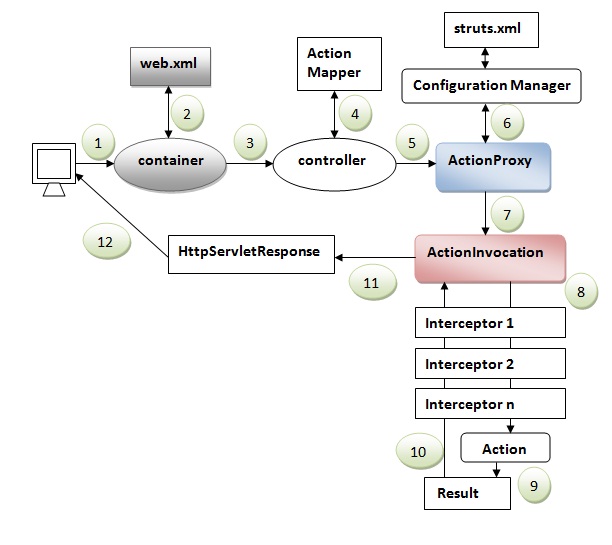
**View displays the result. View can be either JSP page, Velocity templates, XSLT pages, Freemaker or some other presentation-layer technology. Object-Graph Navigation Language (OGNL) is used to reference and manipulate data on the ValueStack.**

**Container reads the WEB-INF/web.xml file, which has all configuration information about out web application. It reads the information from the web.xml file and configures the Struts environment on the startup of the application. In the web.xml file StrutsPrepareAndExecuteFilter is configured to handle the request from client and delegate with the Struts environment.**

**The StrutsPrepareAndExecuteFilter is responsible for preparing as well as executing the various phases of the Struts dispatching process. However, it is**

**recommended to use it, if you don't have other filter, which needs access to action context information, like Sitemesh.**

**Working of Struts framework:**



**Here is the detailed description of how Struts framework works from the time user clicks a URL to the time result is displayed.**

**Request: In the first step the client makes a request for a specific resource using a web browser, which is then received by the web container. After that, the web container  loads web.xml and verifies whether the URL patterns matches or not. Once the verification is ok, the web container transfers the request to Filter Dispatcher.**

**StrutsPrepareAndExecuteFilter: Once the request is transferred to the filter dispatcher evaluates the request and decides an appropriate action as per the mapping of URL (ActionMapper), which then calls the ActionProxy, which then reads the configuration file manager (like the struts.xml file) and then determines an appropriate action for the request. ActionProxy then creates an ActionInvocation, which accounts for the implementation of command pattern implementation and the request is forwarded to appropriate Action Class.**

**Interceptor Stacks: However before calling to the Action Class, the request is passed to Interceptor Stacks, where the list of interceptors are identified that are required to be processed before invoking to the Action class.**

**Action Class: After that, the request is sent to the Action class, which then executes the codes, which finally generates the result of execution as Success or Input or Error.**

**Result: Depending on the result code, Controller then identifies View to be rendered as a result of Action. However, at the time of rendering, Struts tags provided by the framework can be used by the templates.**

**Interceptors Stack: Now, before responding back to the client, the interceptors are executed again and the response returns to the user through the filters configured in the web.xml.**

**Execution flow of Struts**

* **When a client request is given, a web container will receive request**
* **Web container loads web.xml and verifies whether the url-patterns are verified or not, if matches web-container transfer the request to FilterDispatcher**
* **FilterDispatcher hand overs the request to ActionProxy, it is a proxy class which is responsible to apply before and after services to original business logic**
* **ActionProxy contacts ConfiguraionManager class, to know the suitable Action for the request and the needed services for the request**
* **ConfigurationManager class loads structs.xml and provides the required information back to ActionProxy**
* **ActionPorxy delegates the request along with its information to ActionInvocation**
* **ActionInvocation executes the interceptors added to an Action from 1 – N, after that it will call the business logic implemented from N – 1 in reverse order**
* **ActionInvocation receives finally result produced by an action aclass**
* **ActionProxy transfers the result back to FilterDispatcher**
* **FilterDispatcher selects an appropriate view, basing on the result**
* **Finally FilterDispatcher uses RequestDispatchers forwarding mechanism and forward a view as a response back to the client**

**Spring Framework**

**The Spring Framework is an**[**application framework**](https://en.wikipedia.org/wiki/Application_framework)**and**[**inversion of control**](https://en.wikipedia.org/wiki/Inversion_of_control)[**container**](https://en.wikipedia.org/wiki/Servlet_container)**for the**[**Java platform**](https://en.wikipedia.org/wiki/Java_platform)**. The framework's core features can be used by any Java application, but there are extensions for building web applications on top of the**[**Java EE**](https://en.wikipedia.org/wiki/Java_EE)**(Enterprise Edition) platform. Although the framework does not impose any specific**[**programming model**](https://en.wikipedia.org/wiki/Programming_model)**, it has become popular in the Java community as an addition to, or even replacement for the**[**Enterprise JavaBeans**](https://en.wikipedia.org/wiki/Enterprise_JavaBeans)**(EJB) model. The Spring Framework is**[**open source**](https://en.wikipedia.org/wiki/Open-source_software)**.**

**How does spring work ?**

**In spring everything is a single**[**POJO**](https://hu.wikipedia.org/wiki/Plain_Old_Java_Object)**, and it works almost like a glorified**[**HashMap.**](https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html)

**The spring framework itself is built on a straightforward architecture and aren’t going on too much behind the scenes.**

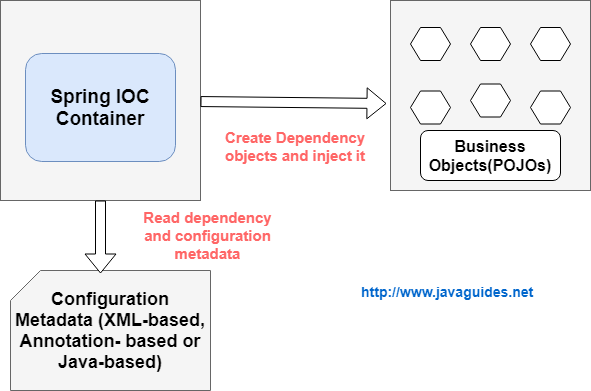
**It’s effortless to try it; you can write a simple main method and just run it.**

**The Spring IoC container**

**It is the heart of the Spring Framework. The IoC container receives metadata from either an XML file, Java annotations, or Java code. The container gets its instructions on what objects to instantiate, configure, and assemble from simple Plain Old Java Objects (POJO) by reading the configuration metadata provided. These created objects through this process called Spring Beans.**

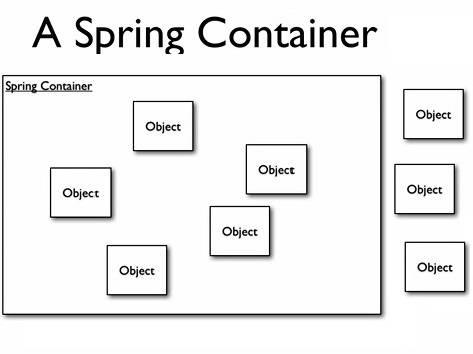
**The responsibilities of IoC container are:**

1. **Instantiating the bean**
2. **Wiring the beans together**
3. **Configuring the beans**
4. **Managing the bean’s entire life-cycle**



**This diagram represents an abstract view of the working of the Spring Framework. It shows how Spring makes use of Java POJO classes and configuration metadata.**

**The real benefits of Spring are showing up through its wiring constructs and using auto-wiring.**



**If you look at the picture above, you can see that there is a lot of squares in there.**

**That’s how the spring container would look like if you graphed it all out.  
The little squares are the Spring beans, and you can see their references. Some of them are standalone, some of them are referencing other beans.  
Just how they wired up and how it makes all these objects that we’re using altogether.**

**The JDBC Example**

**According to our example of a JDBC connection that we**[**were looking at**](https://www.zoltanraffai.com/blog/spring-in-java-what-the-business-needs/)**, one of these might be the entity manager, one of them might be a statement or a prepared statement.  
The other one can be a connection behind the scenes.**

**All of our beans are get stored in this container, and then we access them out of that. This is where the HashMap or Registry metaphor comes into the play. We can simply access them in appropriate ways.**

**Execution flow of Spring**

**1) Client sends an HTTP request to a specific URL  
2) DispatcherServlet of Spring MVC receives the request  
3) It passes the request to a specific controller depending on the URL requested using @Controller and @RequestMapping annotations.  
4) Spring MVC Controller then returns a logical view name and model to DispatcherServlet.  
5) DispatcherServlet consults view resolvers until actual View is determined to render the output  
6) DispatcherServlet contacts the chosen view (e.g. Thymeleaf, Freemarker, JSP) with model data and it renders the output depending on the model data**