UNIT V APPLICATION DEVELOPMENT ENVIRONMENT

- **Overview of MVC architecture**
- **Java Server Faces:** Features - Components - Tags
- **Struts:** Working principle of Struts - Building model components - View components -Controller components - Forms with Struts - Presentation tags - Developing Web applications -
- **Hibernate:** Configuration Settings - Mapping persistent classes - Working with persistent objects - Concurrency - Transactions - Caching - Queries for retrieval of objects –
- **Spring:** Framework - Controllers - Developing simple applications

Presented by,

B.Vijayalakshmi

Computer Centre

MIT Campus

Anna University

APPLICATION DEVELOPMENT ENVIRONMENT

(FRAMEWORKS)

What is a framework?

- **Frameworks** are large bodies (usually many classes) of **prewritten code** to which you add your own code to solve a problem in a specific domain.
- The framework uses your code because it is usually the framework that is in control. You make use of a framework by calling its methods, inheritance, and supplying "callbacks", listeners, or other implementations of the *Observer* pattern.
- **Constrast to library**. Although sometimes large libraries are referred to as frameworks, this is probably not the most common use of the term.
- The difference between a framework and an ordinary programming library is that a framework employs an ***inverted flow of control*** between itself and its clients.
- When using a framework, one usually just implements a few callback functions or specializes a few classes, and then invokes a single method or procedure. At this point, the framework does the rest of the work for you, invoking any necessary client callbacks or methods at the appropriate time and place. For this reason, frameworks are often said to abide by **the Hollywood Principle** ("Don't call us, we'll call you.") or **the Greyhound Principle** ("Leave the driving to us.").

Why to use Frameworks?

- Using frameworks **makes a programmer, code easy**.
- The main advantage of using framework is that, while you are implementing the project, **certain predefined modules are available which you can import by adding just few lines of code**.
  - For example, the persistent objects like Transaction and Query help in direct communication with database as objects and not values
  - Similarly, in Struts Framework, there is ease of communication between data layer (Model), business logic can be easily implemented by using Interceptors (ActionServlets), Validation can be automatically implemented, file download and upload has direct implementation etc.
- Frameworks are used to make the **programmers' focus more on business logic rather than displaying modules or handling databases.**
- Popular Java Web MVC Frameworks are Struts, Struts2, Spring MVC, JSF etc
- Types

  →Web framework (struts,JSF)

  →Application framework (spring)

MVC Architecture

Introduction

- Client server communication takes place in a specific pattern which is known as **application design pattern or architecture.**
- **The application is developed according to different modules specified in architecture**.
- Different types of architecture can be:
  - MVC Architecture
  - Visitor architecture
  - state architecture
  - observer architecture etc.
- J2EE (i.e. Enterprise Java applications such as Spring, Struts, Hibernate, JDBC integrated framework) follow MVC architecture with different tier models
- MVC is implemented in J2EE using following models:
  - Single-tier or one-tier architecture
  - Two-tier architecture
  - Three-tier architecture
  - N-tier architecture

MVC Architecture

- MVC is an abbreviation for **Model-View-Controller**.
- **For ease of development and deployment** of the project, the web-application is divided into **basic 3 modules,**
  - Model - responsible for handling database interaction with the controller (database layer)
  - View - responsible for displaying contents of user (presentation layer)
  - Controller – It acts as an interface between view and model. It intercepts all the requests i.e. receives input and commands to Model / View to change accordingly.

MVC Architecture

- Considering the example of AWT or Java Swing (Drop-down menu/Combobox menu), if the user wants to retrieve the states belonging to the selected country from drop-down menu,
  - **Model:** retrieves the selected names of states from the database
  - **View:** displays the Combobox menu to the user (coordinates with model)
  - **Controller:** sends message to model about the user's selection of country name.
- Considering Struts or Spring or Hibernate frameworks:
  - **Model:** Java Beans (database)
  - **View:** HTML or JSP page
  - **Controller:** back-end servlet (converted JSP's)

MVC Architecture

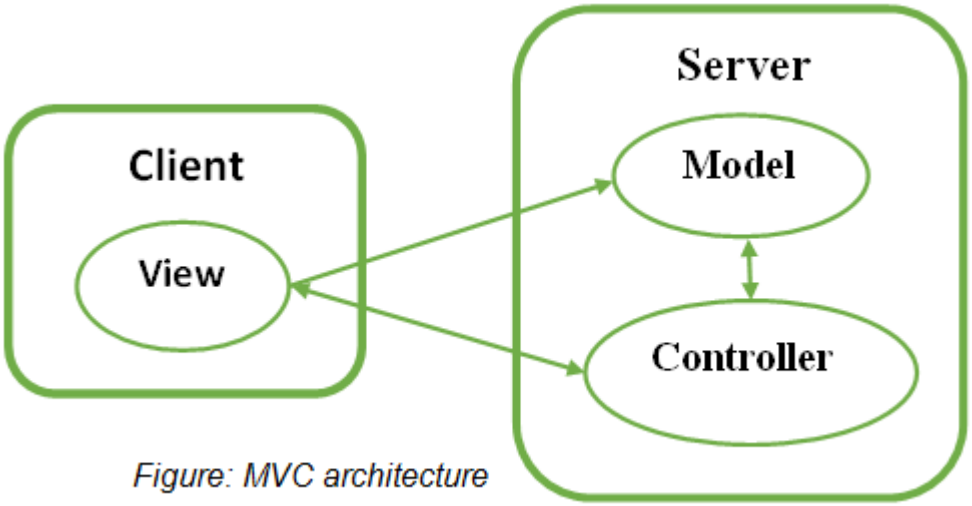


Figure: MVC architecture

- The complete architecture is mainly controlled by the controller.
- The end-user (client) actions on view, based on which the controller delegates the requests to model.
- The model retrieves required data from beans or database (through database queries) and responses back to the controller.
- Based on the response received, the controller changes/modifies the data in view.

MVC Models

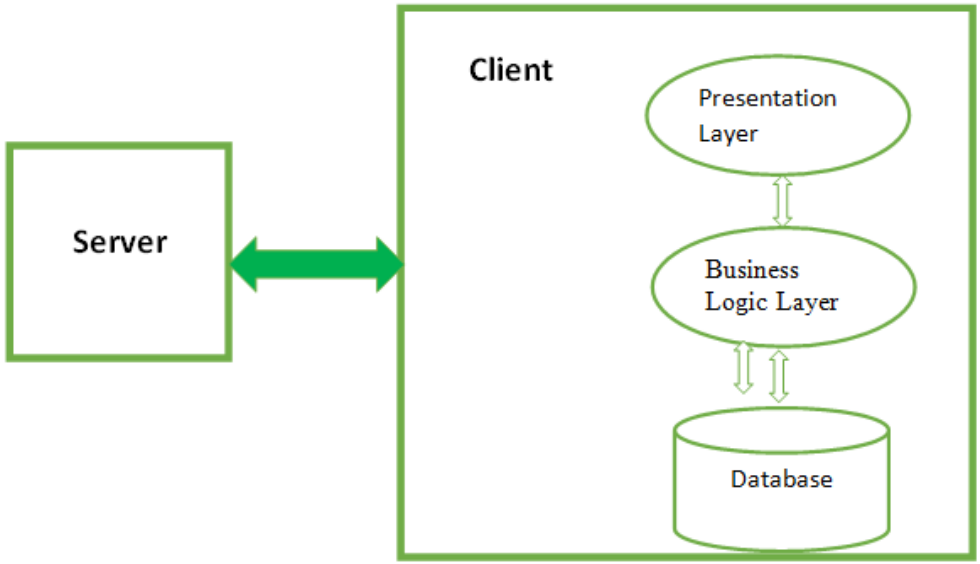


Figure: One-tier MVC architecture

One-tier architecture:

- The client-server architecture in which **every communication element** (such as presentation layer, business logic and database layer) **is placed in a single unit** is termed as one-tier architecture or single-tier architecture.
- Every unit responsible for client-server interaction is located in one place in the client. The same architecture can be repeated for the server as well as n-number of clients.
- The clients or servers following this architecture are also known as **thick clients or servers.**

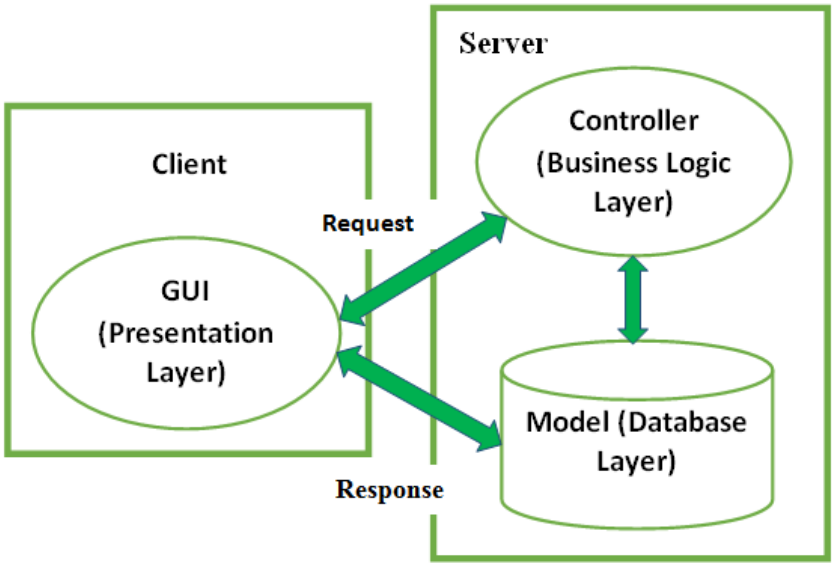


Figure: 2-tier MVC architecture

Two-tier architecture

- In two-tier architecture, the **GUI (view module) is located at client's side while the business logic (controller) and database (model) are located at server side.**
- The client sends request to server through view module and the application at server side directly communicates with the database and retrieved results are sent back to the client.
- **The communication between client-server requires constant internet or intranet connection.**

Three-tier architecture:

- In three-tier architecture, two servers and one client are involved.
- MVC is divided as:
  - Model : Database Server
  - View : Client Machine
  - Controller : Application Server
- The client communicates with application server in the form of a request. These requests are served by the application server in form of responses. The responses include retrieval of data from database by the application server.

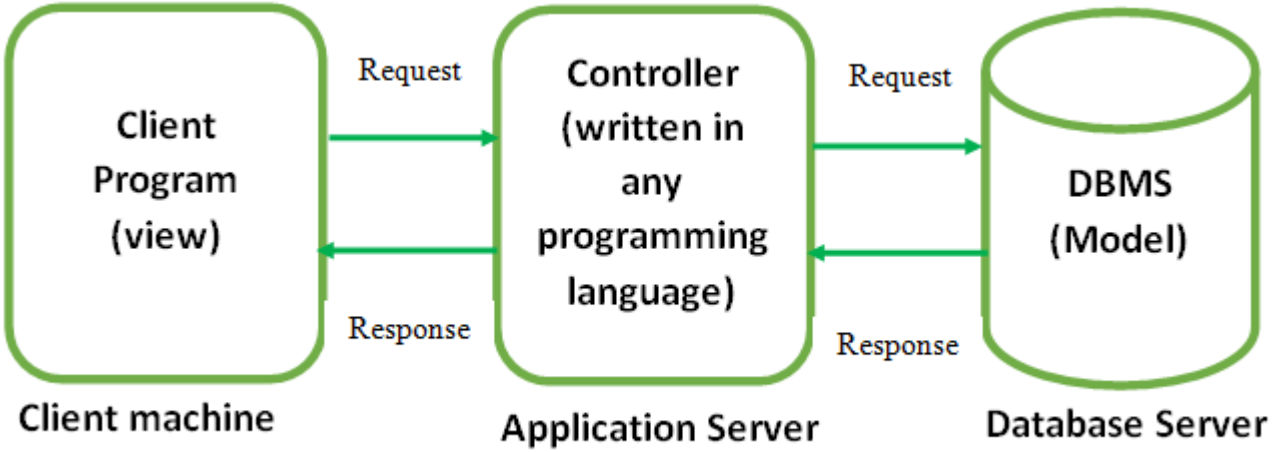


Figure: 3-tier MVC architecture

Advantage of MVC Architecture

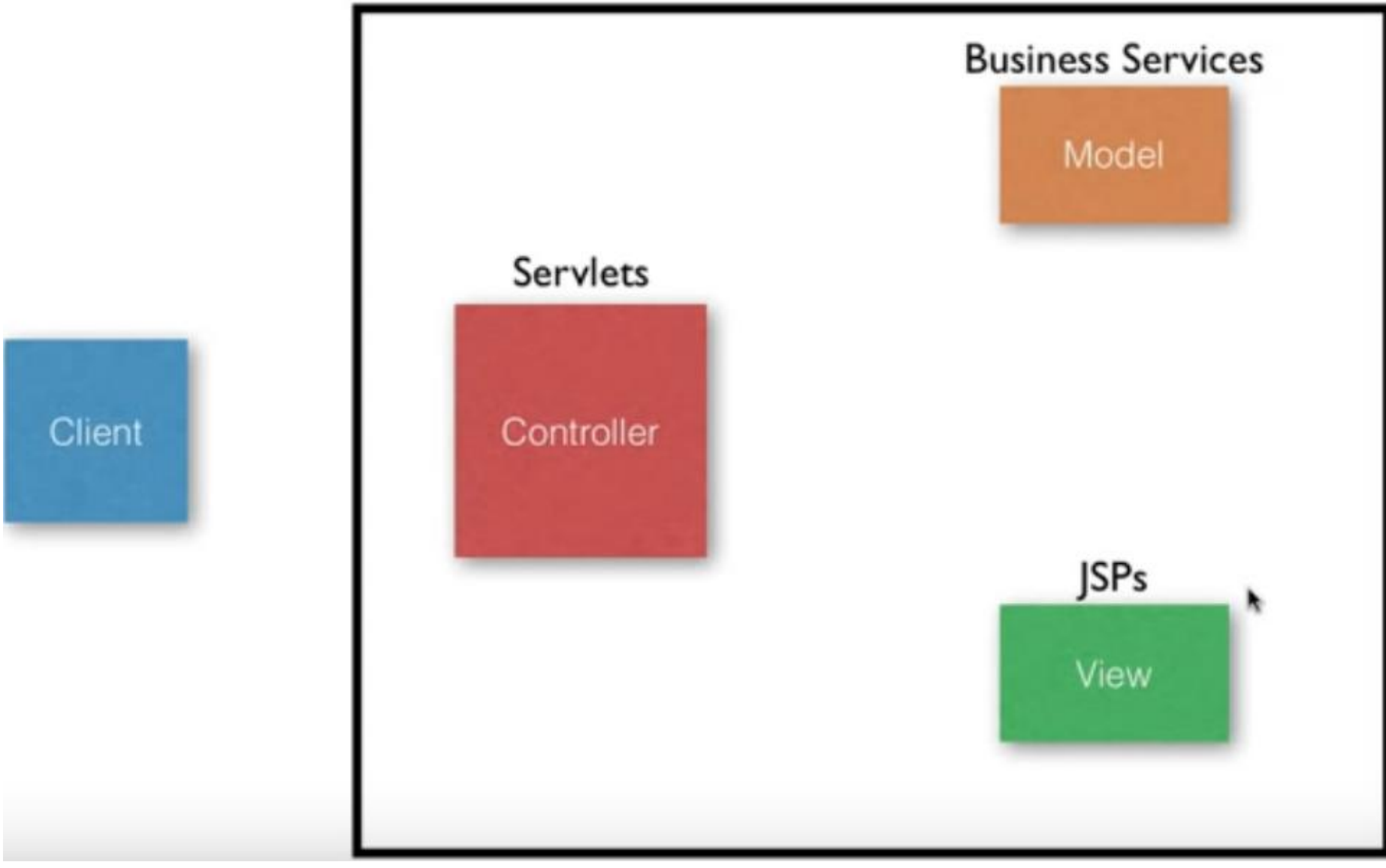
- Navigation control is centralized Now only controller contains the logic to determine the next page.
- Easy to maintain
- Easy to extend
- Easy to test
- Better separation of concerns
- Separate model and presentation enabling developers to focus on their core skills and collaborate more clearly.
- Web designers have to concentrate only on view layer rather than model and controller layer. Developers can change the code for model and typically need not change view layer. Controllers are used to process user actions. In this process, layer model and views may be changed.

STRUTS

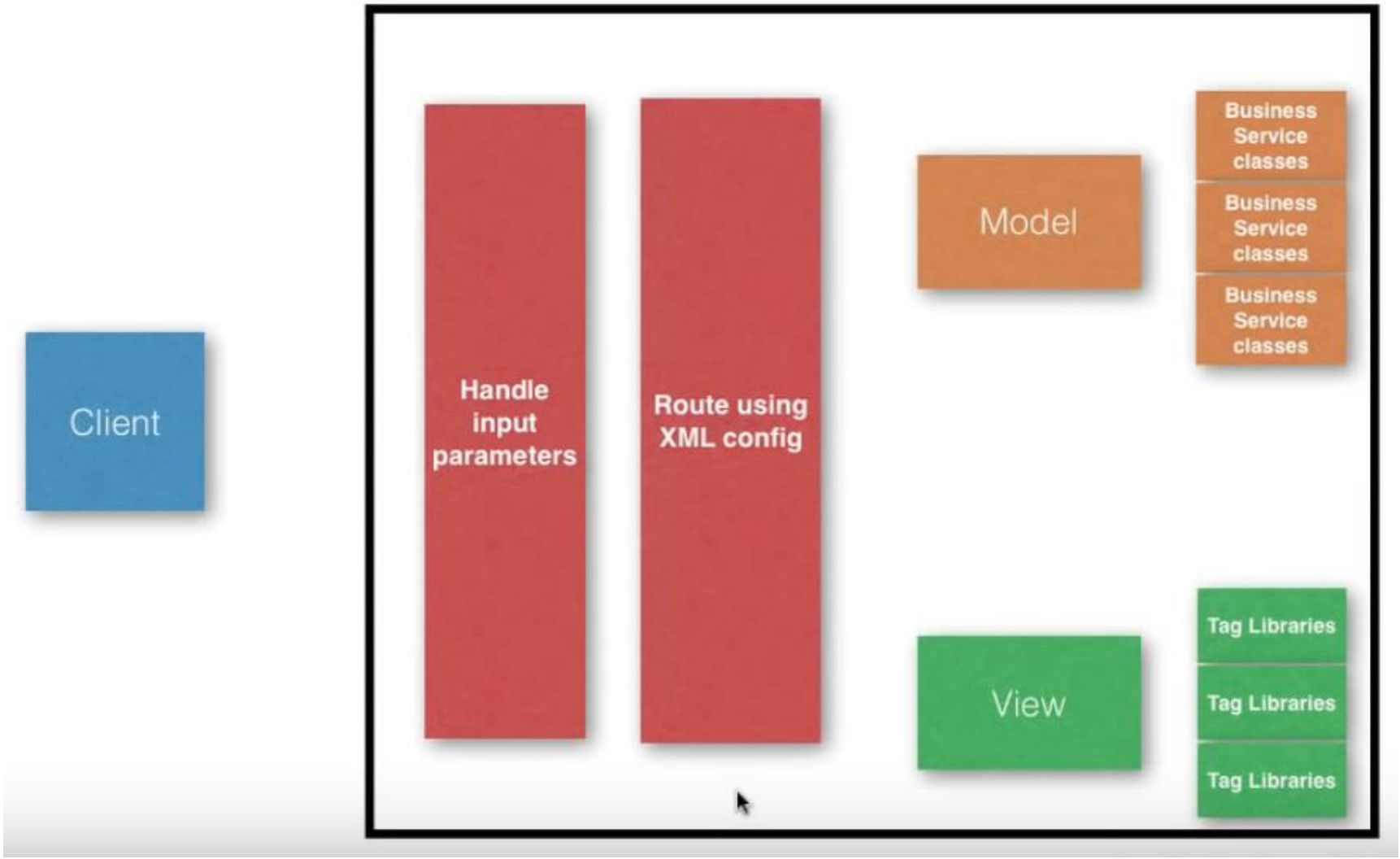
Struts Framework

- It is the product of the Apache software foundation which is basically used for developing web applications in Java.
- As this framework is provided by Apache, it is also known as Apache Struts or Jakarta Struts.
- It was originally known as WebWork framework.
- Struts 2 is the combination of **webwork framework** of opensymphony and **struts 1**.
- **Struts is an invasive, extensible as well as portable framework that is mainly used for software development, web-application development, application maintenance, application deployment etc.**
- The Struts 2 provides supports to ,
  - POJO (Plain Old Java Object) based actions
  - Validation Support
  - AJAX Support
  - Integration support to various frameworks such as Hibernate, Spring, Tiles etc,
  - support to various result types such as Freemarker, Velocity, JSP etc
- **Note:** ***Invasive frameworks*** *– The framework that provides a set of interfaces and classes, which the application has to implement or extend for its development, are termed as invasive frameworks. This provides a way to provide own implementation which makes the application extensible*.

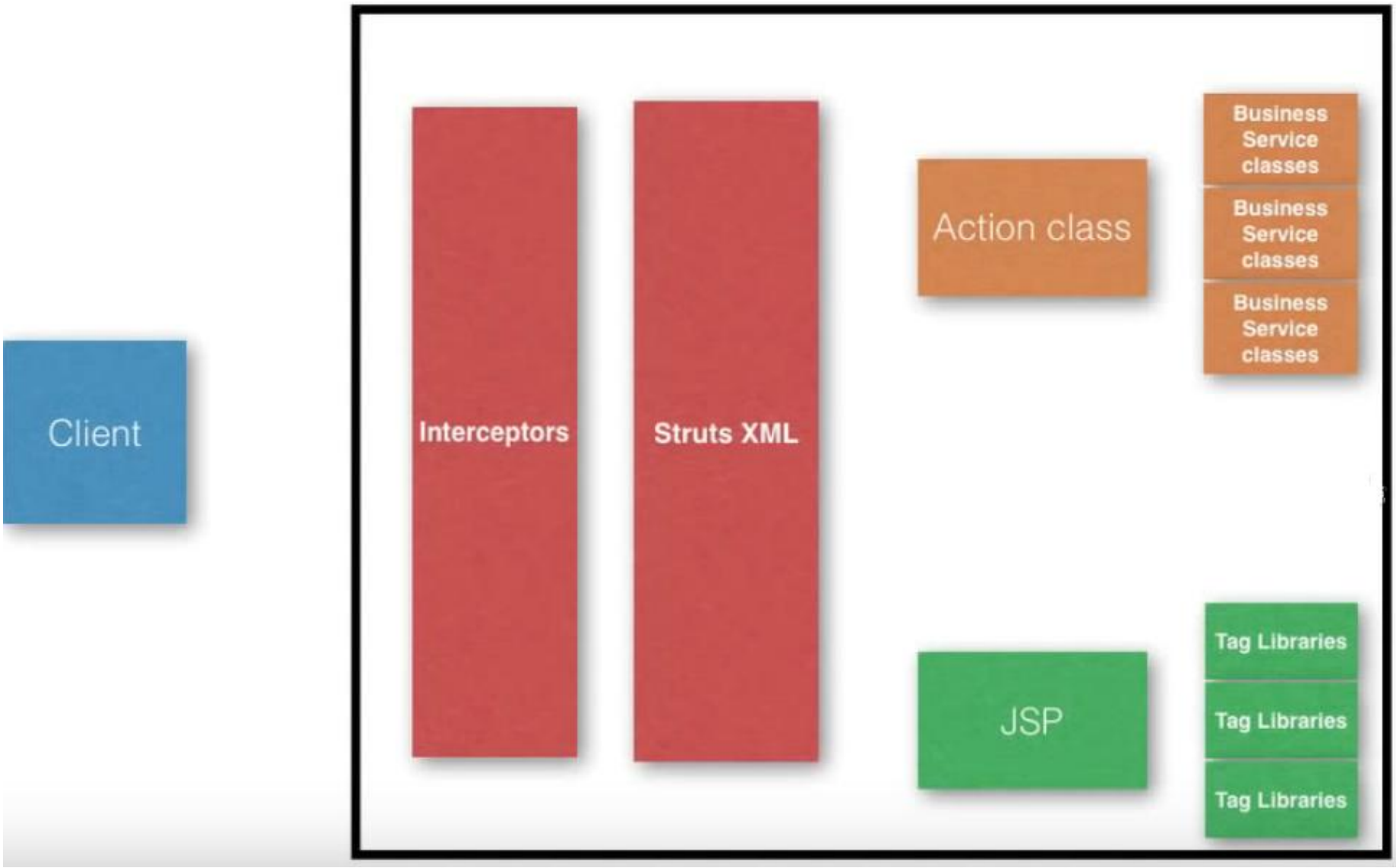
To build our own MVC



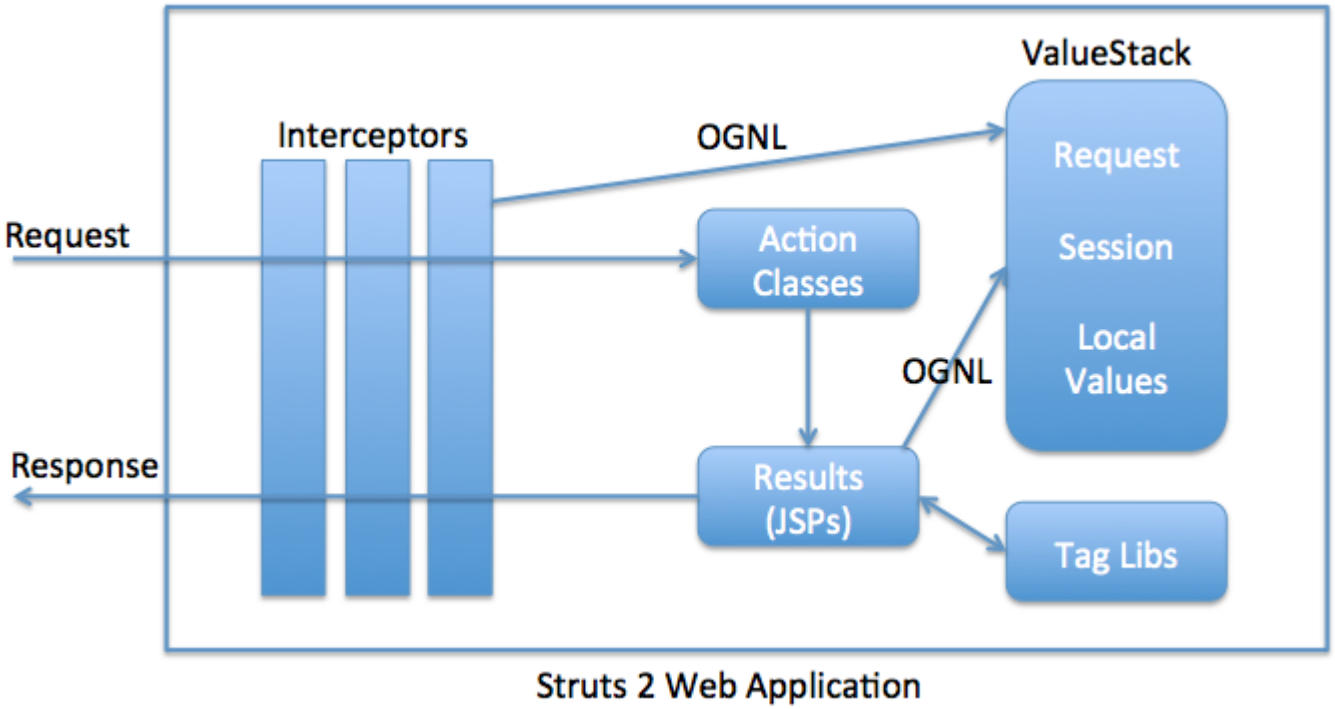
To build our own MVC



STRUTS 2


Client
Interceptors
Struts XML
Action class
Business Service classes
Business Service classes
Business Service classes
JSP
Tag Libraries
Tag Libraries
Tag Libraries

Components of Struts 2



Interceptors

- It is an object that is invoked at the preprocessing and postprocessing of a request
- In Struts 2, interceptor is used to perform operations such as validation, exception handling, internationalization, displaying intermediate result etc.

Advantage of interceptors:

- **Pluggable** If we need to remove any concern such as validation, exception handling, logging etc. from the application, we don't need to redeploy the application. We only need to remove the entry from the struts.xml file.

ValueStack

- It is simply a stack that contains application specific objects such as action objects and other model object.
- We can put objects in the valuestack, query it and delete it.
- Methods in ValueStack interface.,
  - **public String findString(String expr)** finds the string by evaluating the given expression.
  - **public Object findValue(String expr)** finds the value by evaluating the specified expression.
  - **public Object findValue(String expr, Class c)** finds the value by evaluating the specified expression.
  - **public Object peek()** It returns the object located on the top of the stack.
  - **public Object pop()** It returns the object located on the top of the stack and removes it.
  - **public void push(Object o)** It puts the object on the top of the stack.
  - **public void set(String key, Object value)** It sets the object on the stack with the given key. It can be get by calling the findValue(key) method.
  - **public int size()** It returns the number of objects from the stack.

ActionContext

- It is a container of objects in which action is executed.
- The values stored in the ActionContext are unique per thread (i.e. ThreadLocal). So we don't need to make our action thread safe.
- We can get the reference of ActionContext by calling the getContext() method of ActionContext class. It is a static factory method.
- For example:

```
ActionContext context = ActionContext.getContext();
```

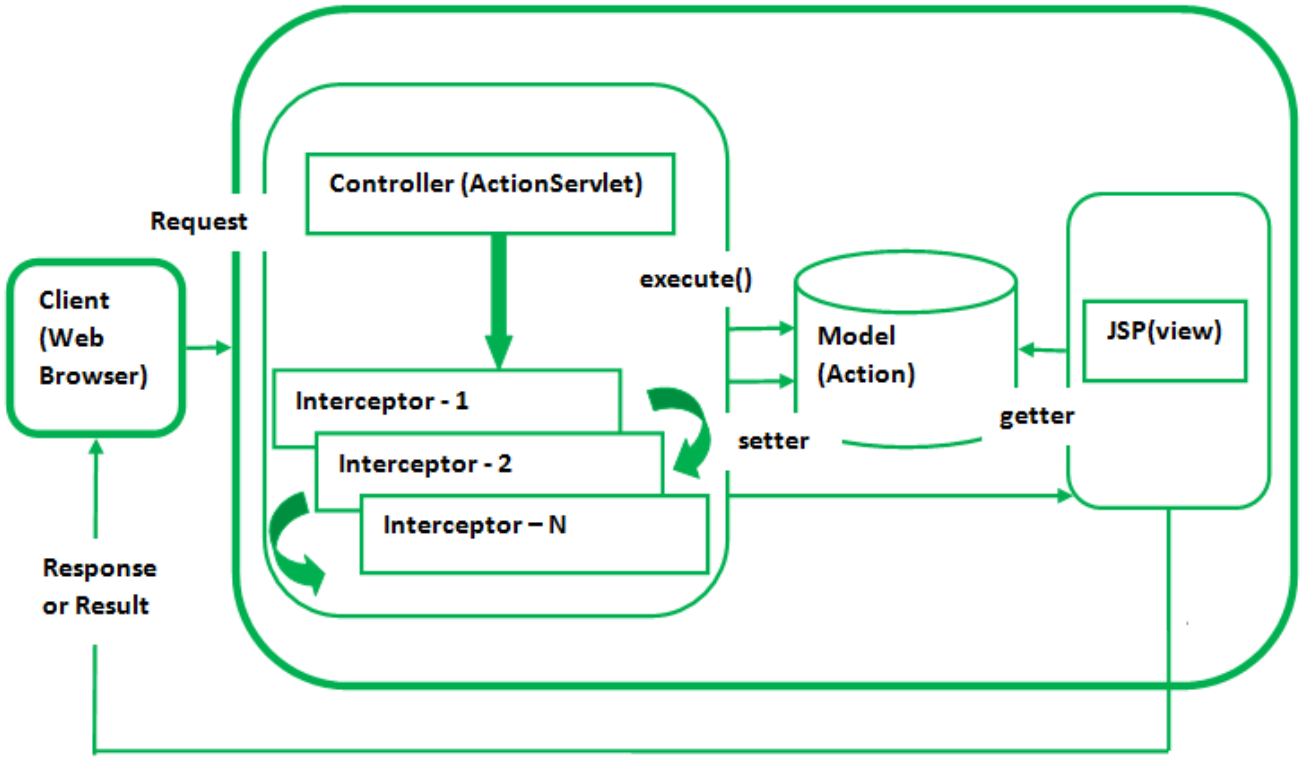
ActionInvocation

- It represents the execution state of an action.
- It holds the action and interceptors objects.
- The struts framework provides **ActionInvocation interface** to deal with ActionInvocation.
- It provides many methods, some of them can be used to get the instance of ValueStack, ActionProxy, ActionContext, Result etc.

OGNL
(Object-Graph Navigation Language)

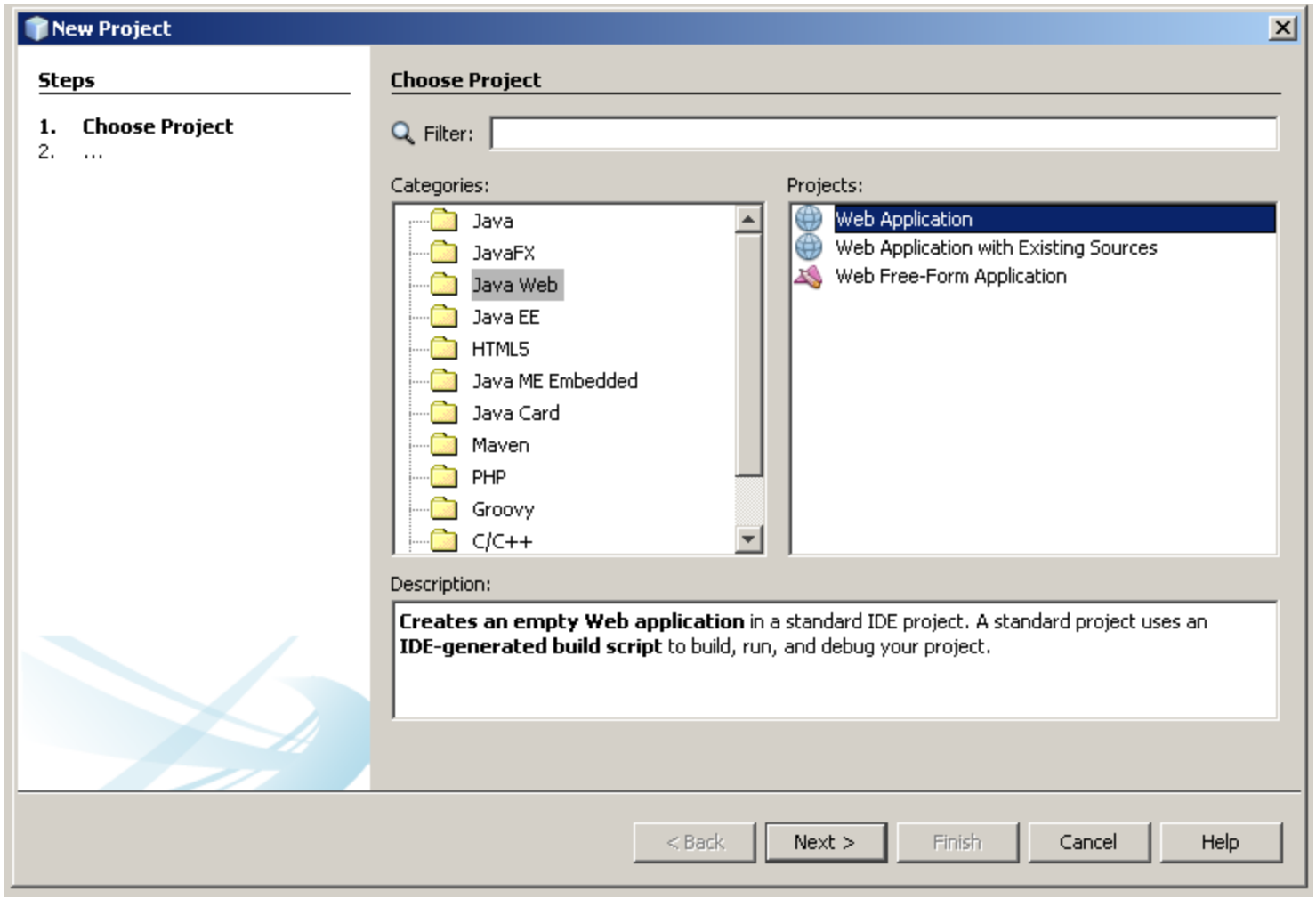
- It is a powerful Expression Language that is used to manipulate data stored on the ValueStack.
- Both interceptors and result pages can access data stored on ValueStack using OGNL.

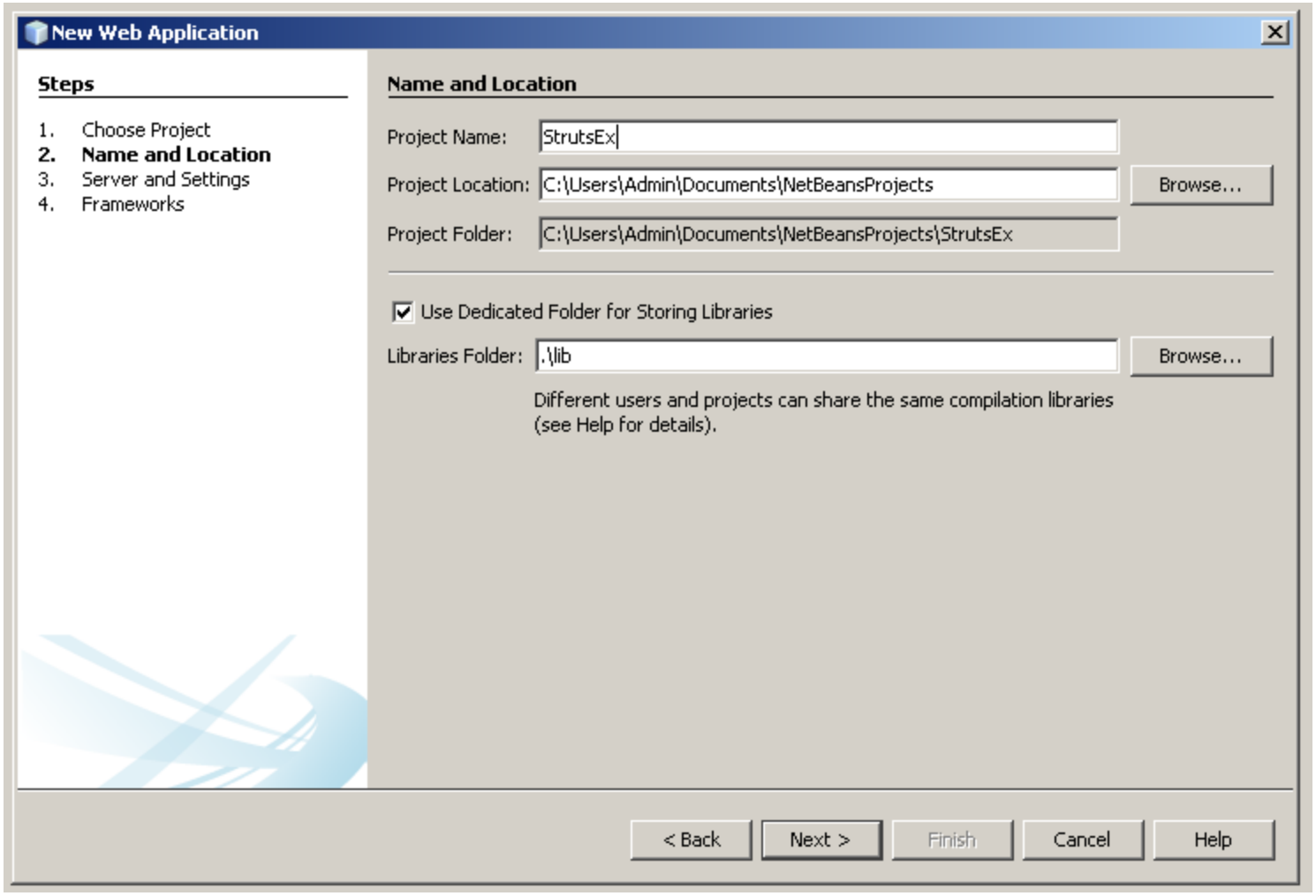
Struts Request Response Scenario

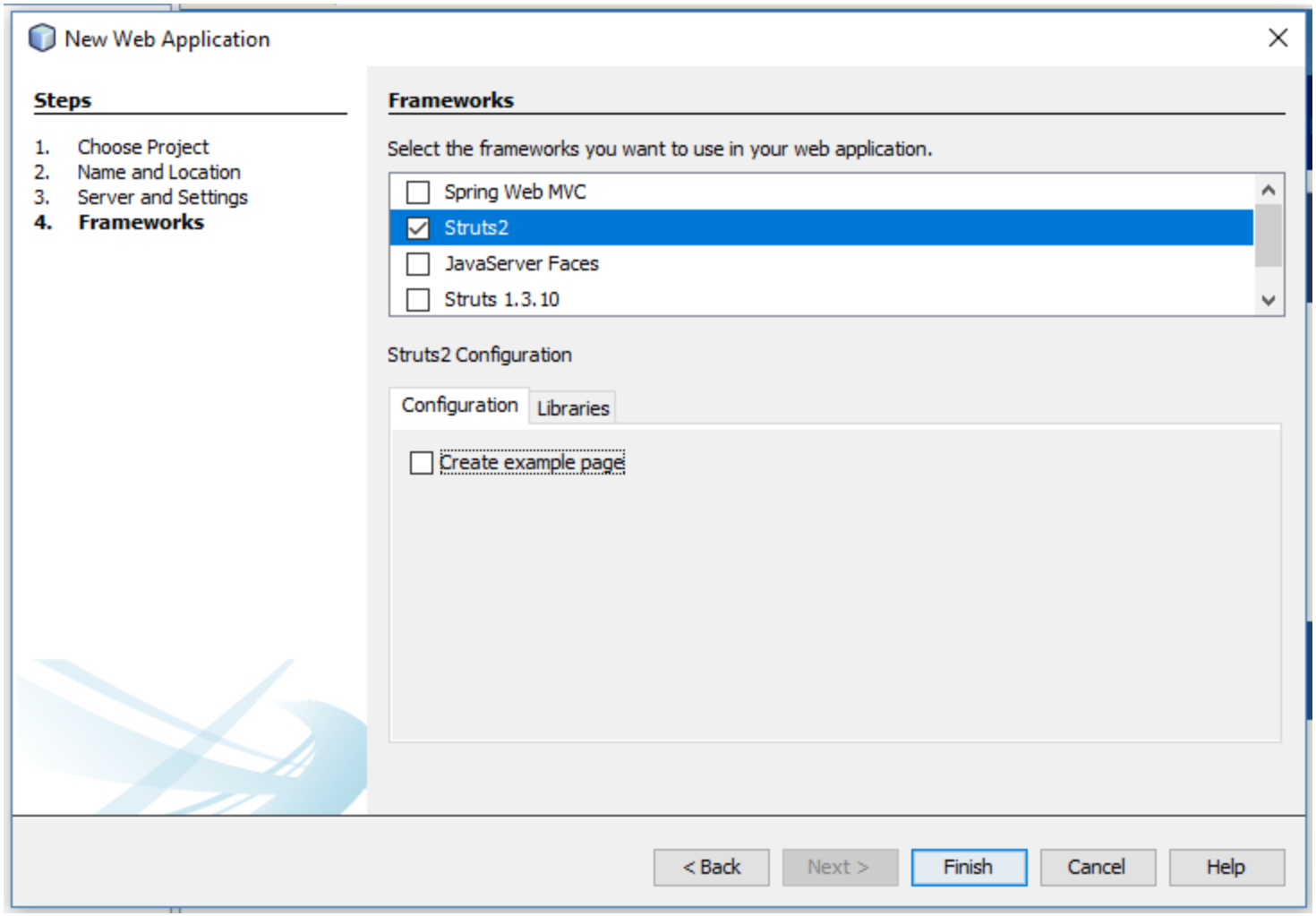


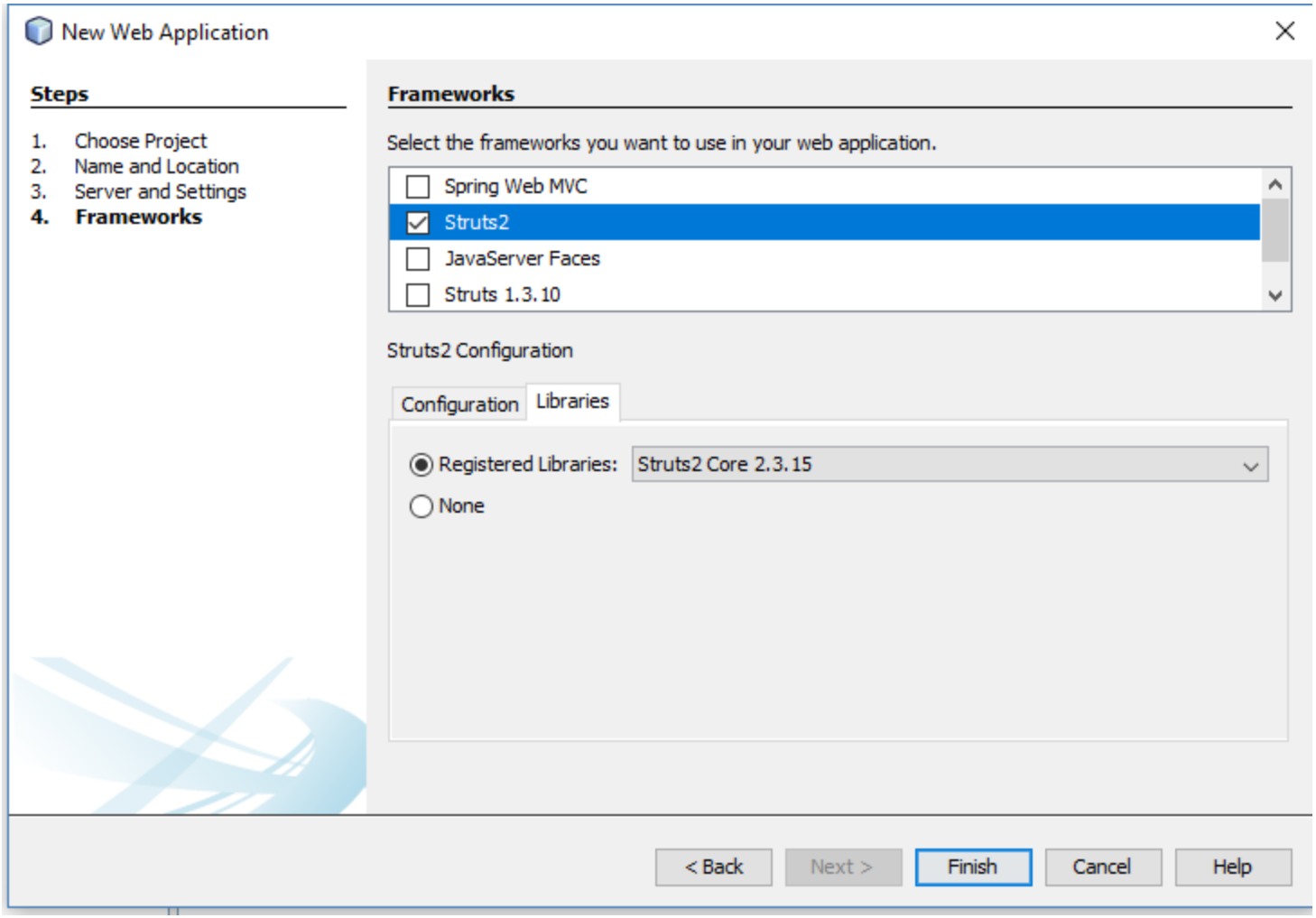
- The basic flow of request to response is:The client sends a request from the browser
- The controller (ActionServlet) invokes the execute() method and function responsible for intended action is called. (in turn calls interceptors)
- If required, the data is retrieved from the database by using getter and setter methods.
- A response is created by the service () method which then forwards it to intended JSP.
- The client receives the response via JSP in the browser.

STRUTS in NetBeans

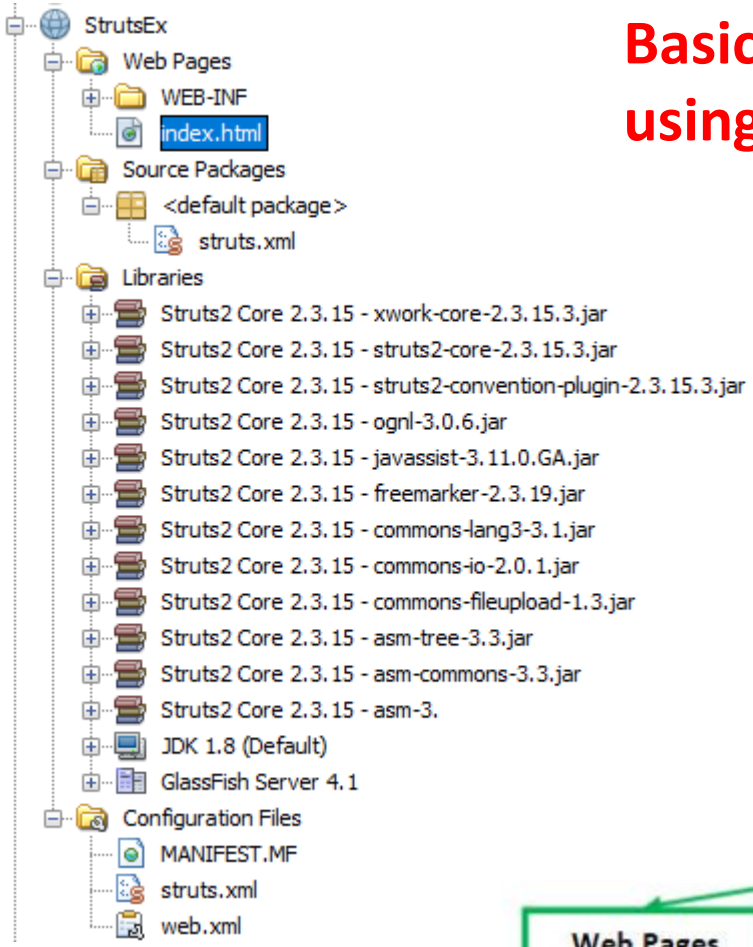


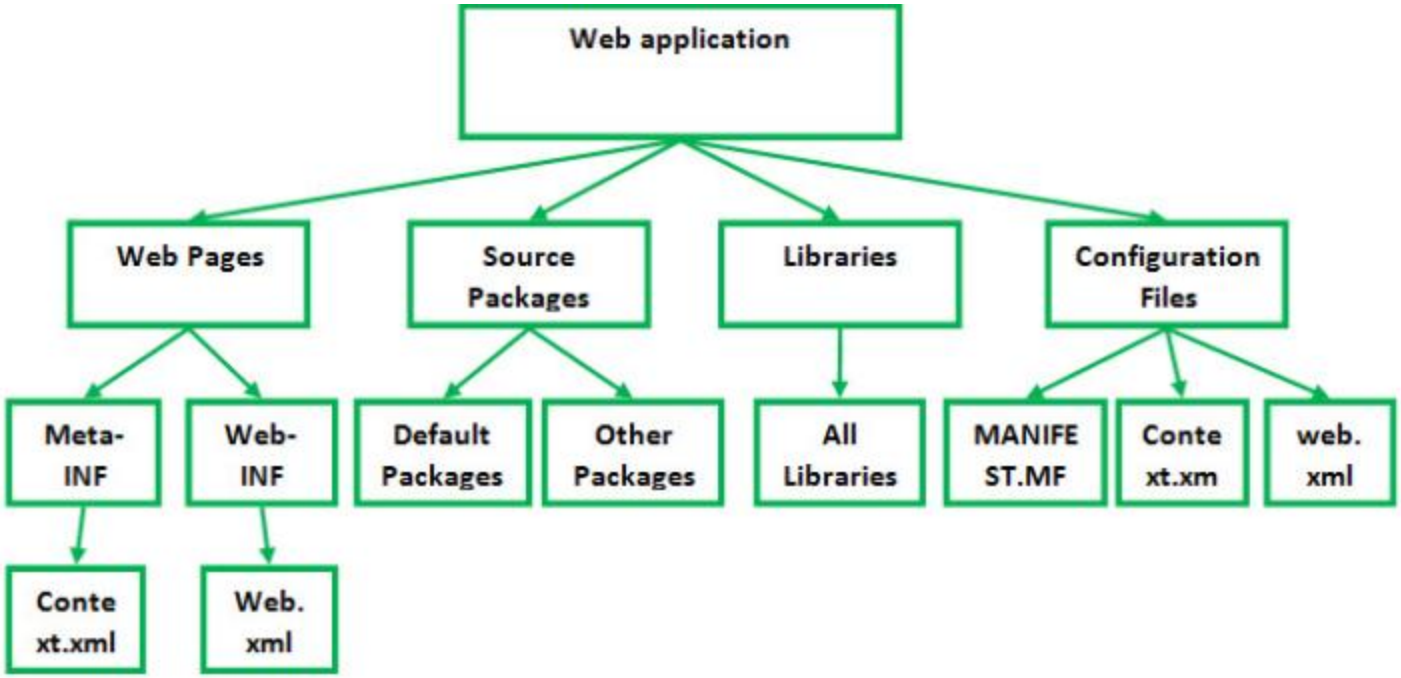

New Web Application
Steps
1. Choose Project
2. Name and Location
3. Server and Settings
4. Frameworks
Name and Location
Project Name: StrutsEx
Project Location: C:\Users\Admin\Documents\NetBeansProjects
Browse...
Project Folder: C:\Users\Admin\Documents\NetBeansProjects\StrutsEx
Use Dedicated Folder for Storing Libraries
Libraries Folder: .\lib
Browse...
Different users and projects can share the same compilation libraries
(see Help for details).
< Back
Next >
Finish
Cancel
Help


New Web Application
Steps
1. Choose Project
2. Name and Location
3. Server and Settings
4. Frameworks
Frameworks
Select the frameworks you want to use in your web application.
Spring Web MVC
Struts2
JavaServer Faces
Struts 1.3.10
Struts2 Configuration
Configuration
Libraries
Create example page
< Back
Next >
Finish
Cancel
Help


New Web Application
Steps
1. Choose Project
2. Name and Location
3. Server and Settings
4. Frameworks
Frameworks
Select the frameworks you want to use in your web application.
Spring Web MVC
Struts2
JavaServer Faces
Struts 1.3.10
Struts2 Configuration
Configuration
Libraries
Registered Libraries:
Struts2 Core 2.3.15
None
< Back
Next >
Finish
Cancel
Help

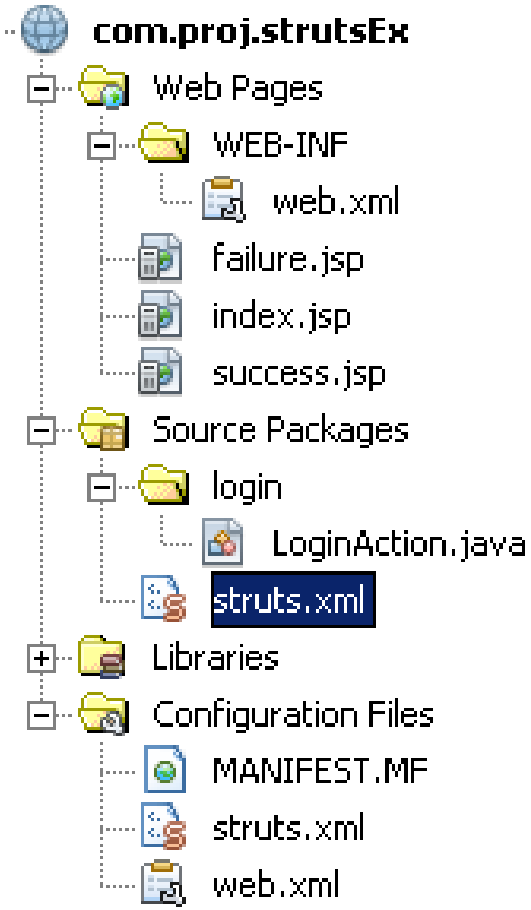
Basic directory structure of web applications using Struts2 Framework





- **Source Packages** : The packages are of two types,
  - **Default packages** – single level package. All the required action (.java) classes are placed here.
  - **Other (user-defined packages)** – we can create multilevel packages here. Action classes can be placed here just as default package.
  - The action classes contain the getter and setter methods of all the variables declared in the class. Plus, it also contains one method called excecute().
- **Libraries**: All the necessary libraries such as “struts-tags”, JSTL’s EL, jdk, jre are placed here. The path of these libraries is also mentioned in context.xml. In the IDE’s like NetBeans or Eclipse, they get automatically created.
- **Configuration Files**: All the configuration files like : context.xml, web.xml, manifest.mf are to be placed under this folder. Here, context.xml and web.xml are same as defined above. The manifest files (manifest.mf) can have extensions of “.mf” or “manifest”. This file contains code that provides versatility to the code so that it can be executed on any platform. Any web application after deployment is transferred as either “war”, “rar” or “ear” extension. The manifest.mf file contains a “meta” tag which specifies the main file to execute when your web application is deployed.
- **Web.xml**: The core of any web application (using any framework) is web.xml. It is also known as Deployment descriptor. The file is to be placed under the Web - INF folder. It contains the actual running description of the application. Which file to run first, what should be the time-out of the sessions, which filters are to be implemented, what is the path of filters and other similar information is placed in web.xml
- **Context.xml**: The configuration file, context.xml is to be placed under META-INF folder. Context.xml contains the information about the path of the Apache Tomcat server, path of application resources such as images, audio-video files etc.

Struts Example



Web.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.1" 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_3_1.xsd">
    <filter>
        <filter-name>struts2</filter-name>
        <filter-class>
            org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter
        </filter-class>
    </filter>
    <filter-mapping>
        <filter-name>struts2</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
    <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>
    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>
</web-app>
```

index.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>JSP Page</title>
    </head>
    <body>
        <form action=login" method=post">
        Enter user Name: <input type="text" id="uname" name="uname"/><br/>
        Enter password:<input type="password" id="pass" name="pass"/><br/>
        <input type="submit" value="Login">
        </form>
    </body>
</html>
```

struts.xml

```
<!DOCTYPE struts PUBLIC
"-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
"http://struts.apache.org/dtds/struts-2.0.dtd">

<struts>
    <!-- Configuration for the default package. -->
    <package name="default" extends="struts-default">
        <action name="login" class="login.LoginAction">
            <result name="SUCCESS">/success.jsp</result>
            <result name="FAILURE">/failure.jsp</result>
        </action>
    </package>
</struts>
```

LoginAction.java

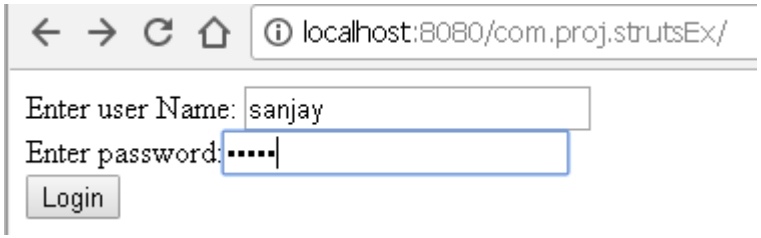
```
package login;
import com.opensymphony.xwork2.ActionSupport;
public class LoginAction extends ActionSupport
{
    String uname,pass;
    public String getUname() {
        return uname;
    }
    public void setUname(String uname) {
        this.uname = uname;
    }

    public String getPass() {
        return pass;
    }
    public void setPass(String pass) {
        this.pass = pass;
    }
    @Override
    public String  execute()
    {
        System.out.println("HELLO");
        System.out.println("UNAME"+uname);
        System.out.println("PASS"+pass);
        if(uname.equals(pass))
        {
            System.out.println("SUCCESS");
            return "SUCCESS";
        }
        else
        {
             System.out.println("FAILURE");
            return "FAILURE";
        }
    }
}
```

success.java

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@taglib uri="/struts-tags" prefix="s" %>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>JSP Page</title>
    </head>
    <body>
        <h1>Hello <s:property value="uname"/>!!!</h1>
    </body>
</html>
```

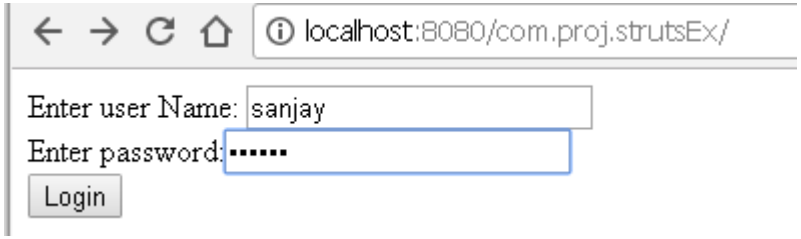
failure.java

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>JSP Page</title>
    </head>
    <body>
        <h1>Error!</h1>
    </body>
</html>
```




```
Warning:   Action [login"] does not match allowed action names pattern [[a-zA-Z0-9._!/\-]*], cleaning it up!
Info:   HELLO
Info:   UNAMEsanjay
Info:   PASS12345
Info:   FAILURE
```

Error!




```
Warning:   Action [login"] does not match allowed action names pattern [[a-zA-Z0-9._!/\-]*], cleaning it up!
Info:   HELLO
Info:   UNAMEsanjay
Info:   PASSsanjay
Info:   SUCCESS
```

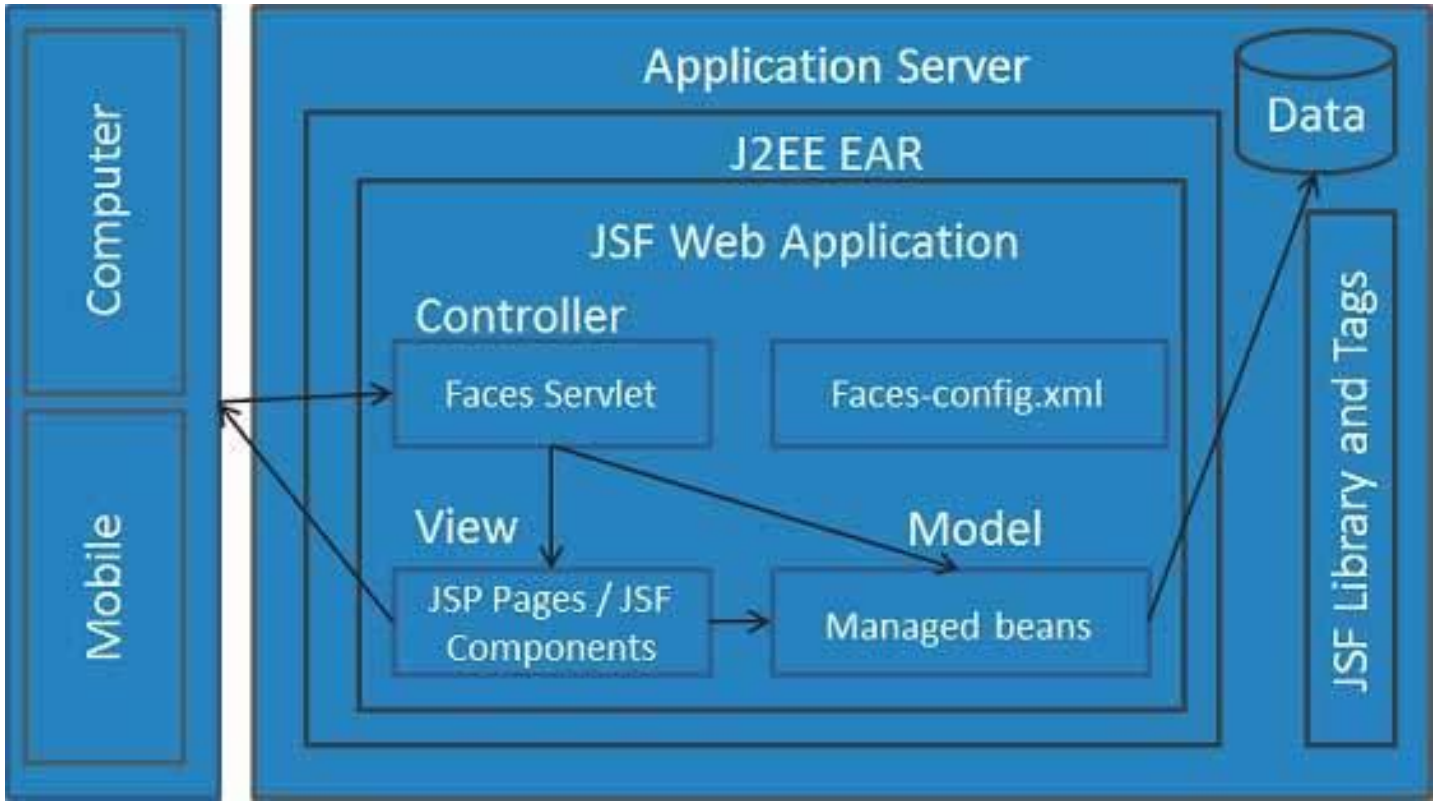
Hello sanjay!!!

Java Server Faces (JSF)

Java Server Faces (JSF)

- It is a **server side component based user interface framework**
- It is **used to develop web applications**
- It **provides a well-defined programming model and consists of rich API and tag librarie**s.
- The JSF API provides components (inputText, commandButton etc) and helps to manage their states.
- It also provides server-side validation, data conversion, defining page navigation, provides extensibility, supports for internationalization, accessibility etc.
- The JSF Tag libraries are used to add components on the web pages and connect components with objects on the server. It also contains tag handlers that implements the component tag.
- With the help of these features and tools, you can easily and effortlessly create server-side user interface.

JSF Architecture



JSF Features

- Latest version of JSF 2.2 provides the following features.
  - Component Based Framework
  - Implements Facelets Technology
  - Integration with Expression Language
  - Support HTML5
  - Ease and Rapid web Development.
  - Support Internationalization
  - Bean Annotations
  - Default Exception Handling
  - Templating
  - Inbuilt AJAX Support
  - Security

JSF User Interface Component Model

- JSF provides rich set of components library to define the architecture of application.
- It includes the following:
  - **Rich set of classes** for specifying the state and behavior of user interface components.
  - A **rendering model** that defines how to render the components in various ways.
  - A **conversion model** that defines how to register data converters onto a component.
  - An **event and listener model** that defines how to handle component events.
  - A **validation model** that defines how to register validators onto a component.

JSF User Interface Components

- JavaServer Faces HTML tag library represents HTML form components and other basic HTML elements, which are used to display or accept data from the user.
- A JSF form send this data to the server after submitting the form.

Tag	Functions	Rendered As	Appearance
h:inputText	It allows a user to input a string.	An HTML <input type="text"> element	A field
h:outputText	It displays a line of text.	Plain text	Plain text
h:form	It represents an input form.	An HTML <form> element	No appearance
h:commandButton	It submits a form to the application.	An HTML <input type=value> element for which the type value can be "submit", "reset", or "image"	A button
h:inputSecret	It allows a user to input a string without the actual string appearing in the field.	An HTML <input type="password"> element	A field that displays a row of characters instead of the actual string entered.
h:inputTextarea	It allows a user to enter a multiline string.	An HTML <textarea> element	A multirow field

h:commandLink	It links to another page or location on a page.	An HTML <a href> element	A link
h:inputHidden	It allows a page author to include a hidden variable in a page.	An HTML <input type="hidden"> element	No appearance
h:inputFile	It allows a user to upload a file.	An HTML <input type="file"> element	A field with a Browse button
h:graphicImage	It displays an image.	An HTML <img> element	An image
h:dataTable	It represents a data wrapper.	An HTML <table> element	A table that can be updated dynamically.

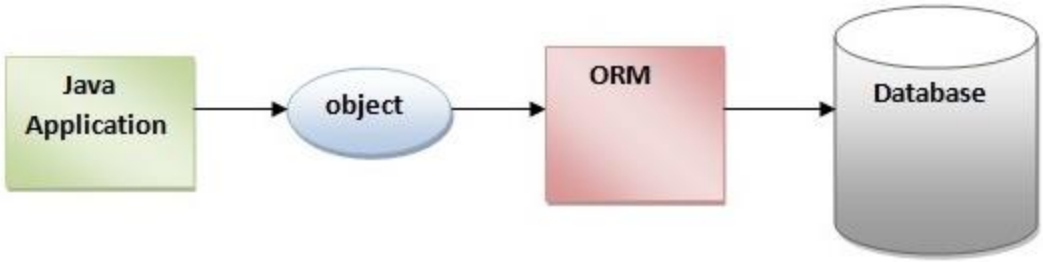
h:message	It displays a localized message.	An HTML tag if styles are used	A text string
h:messages	It displays localized messages.	A set of HTML tags if styles are used	A text string
h:outputFormat	It displays a formatted message.	Plain text	Plain text
h:outputLabel	It displays a nested component as a label for a specified input field.	An HTML <label> element	Plain text
h:outputLink	It links to another page or location on a page without generating an action event.	An HTML <a> element	A link

h:panelGrid	It displays a table.	An HTML <table> element with <tr> and <td> elements	A table
h:panelGroup	It groups a set of components under one parent.	A HTML or element	A row in a table
h:selectBooleanCheckbox	It allows a user to change the value of a Boolean choice.	An HTML <input type="checkbox" > element	A check box
h:selectManyCheckbox	It displays a set of check boxes from which the user can select multiple values.	A set of HTML <input> elements of type checkbox	A group of check boxes
h:selectManyListbox	It allows a user to select multiple items from a set of items all displayed at once.	An HTML <select> element	A box

h:selectManyMenu	It allows a user to select multiple items from a set of items.	An HTML <select> element	A menu
h:selectOneListbox	It allows a user to select one item from a set of items all displayed at once.	An HTML <select> element	A box
h:selectOneMenu	It allows a user to select one item from a set of items.	An HTML <select> element	A menu
h:selectOneRadio	It allows a user to select one item from a set of items.	An HTML <input type="radio"> element	A group of options
h:column	It represents a column of data in a data component.	A column of data in an HTML table	A column in a table

Hibernate

Hibernate Framework



- It was started in 2001 by Gavin King as an alternative to EJB2 style entity bean
- It simplifies the development of java application to interact with the database.
- It is an open source, lightweight, **ORM (Object Relational Mapping)** tool.
- An ORM tool simplifies the data creation, data manipulation and data access.
- It is a programming technique that maps the object to the data stored in the database.
- The ORM tool internally uses the JDBC API to interact with the database.

What is ORM?

- ORM stands for **O**bject-**R**elational **M**apping (ORM) is a programming technique for converting data between relational databases and object oriented programming languages such as Java, C# etc.
- Advantages
  - Lets business code access objects rather than DB tables
  - Hides details of SQL queries from OO logic
  - Based on JDBC 'under the hood’
  - No need to deal with the database implementation
  - Entities based on business concepts rather than database structure
  - Transaction management and automatic key generation
  - Fast development of application.

Hibernate

- It is an Object-Relational Mapping(ORM) solution for JAVA
- It is a powerful, high performance Object-Relational Persistence and Query service for any Java Application.
- Hibernate maps Java classes to database tables and from Java data types to SQL data types and relieve the developer from 95% of common data persistence related programming tasks
- Hibernate sits between traditional Java objects and database server to handle all the work in persisting those objects based on the appropriate O/R mechanisms and patterns.

Advantages of Hibernate Framework

- **Open source and Lightweight:** Hibernate framework is opensource under the LGPL license and lightweight.
- **Fast performance:** The performance of hibernate framework is fast because cache is internally used in hibernate framework. There are two types of cache in hibernate framework first level cache and second level cache. First level cache is enabled by default.
- **Database Independent query:** HQL (Hibernate Query Language) is the object-oriented version of SQL. It generates the database independent queries. So you don't need to write database specific queries. Before Hibernate, If database is changed for the project, we need to change the SQL query as well that leads to the maintenance problem.
- **Automatic table creation:** Hibernate framework provides the facility to create the tables of the database automatically. So there is no need to create tables in the database manually.
- **Simplifies complex join:** To fetch data form multiple tables is easy in hibernate framework.
- **Provides query statistics and database status:** Hibernate supports Query cache and provide statistics about query and database status.

Advantages of Hibernate Framework

- Hibernate takes care of mapping Java classes to database tables using XML files and without writing any line of code.
- Provides simple APIs for storing and retrieving Java objects directly to and from the database.
- If there is change in Database or in any table then the only need to change XML file properties.
- Abstract away the unfamiliar SQL types and provide us to work around familiar Java Objects.
- Hibernate does not require an application server to operate.
- Manipulates Complex associations of objects of your database.
- Minimize database access with smart fetching strategies.
- Provides Simple querying of data.

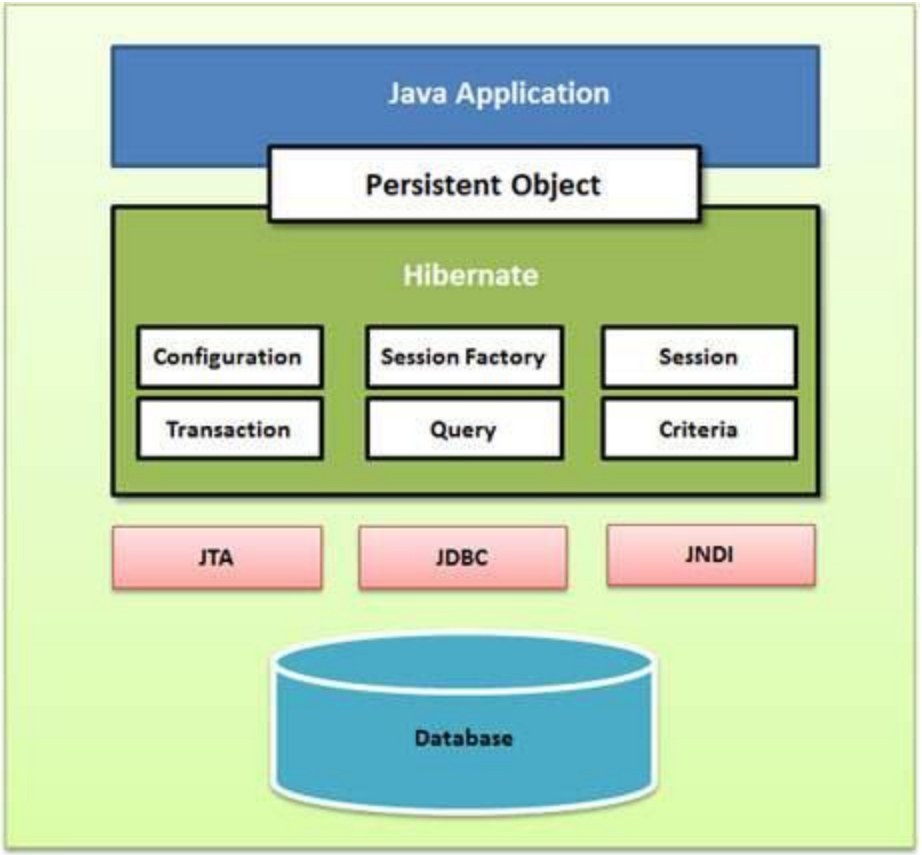
Supported Databases

- Hibernate supports almost all the major RDBMS.
- Following is list of few of the database engines supported by Hibernate,
  - HSQL Database Engine
  - DB2/NT
  - MySQL
  - PostgreSQL
  - FrontBase
  - Oracle
  - Microsoft SQL Server Database
  - Sybase SQL Server
  - Informix Dynamic Server

Supported Technologies

- Hibernate supports a variety of other technologies, including the following:
  - XDoclet Spring
  - J2EE
  - Eclipse plug-ins
  - Maven

Hibernate Architecture



There are 4 layers in hibernate architecture java application layer, hibernate framework layer, Internal API layer and database layer and it uses many objects session factory, session, transaction etc.. Along with existing Java API such as JDBC (Java Database Connectivity), JTA (Java Transaction API) and JNDI (Java Naming Directory Interface).

Elements of Hibernate Architecture

- Configuration Object
- SessionFactory Object
- Session Object
- Transaction Object
- Query Object
- Criteria Object

Configuration Object

- It is the first Hibernate object you create in any Hibernate application and usually created only once during application initialization
- It represents a configuration or properties file required by the Hibernate.
- The Configuration object provides two keys components:
  - **Database Connection:** This is handled through one or more configuration files supported by Hibernate. These files are **hibernate.properties** and **hibernate.cfg.xml**.
  - **Class Mapping Setup**

    This component creates the connection between the Java classes and database tables..

SessionFactory Object

- Configuration object is used to create a SessionFactory object which in turn configures Hibernate for the application using the supplied configuration file and allows for a Session object to be instantiated.
- The SessionFactory is a thread safe object and used by all the threads of an application.
- The SessionFactory is heavyweight object so usually it is created during application start up and kept for later use.
- You would need one SessionFactory object per database using a separate configuration file.
- So if you are using multiple databases then you would have to create multiple SessionFactory objects.

Session Object

- A Session is used to get a physical connection with a database.
- The Session object is lightweight and designed to be instantiated each time an interaction is needed with the database.
- Persistent objects are saved and retrieved through a Session object.
- The session objects should not be kept open for a long time because they are not usually thread safe and they should be created and destroyed them as needed.

Transaction Object

- A Transaction represents a unit of work with the database and most of the RDBMS supports transaction functionality.
- Transactions in Hibernate are handled by an underlying transaction manager and transaction (from JDBC or JTA).
- This is an optional object and Hibernate applications may choose not to use this interface, instead managing transactions in their own application code.

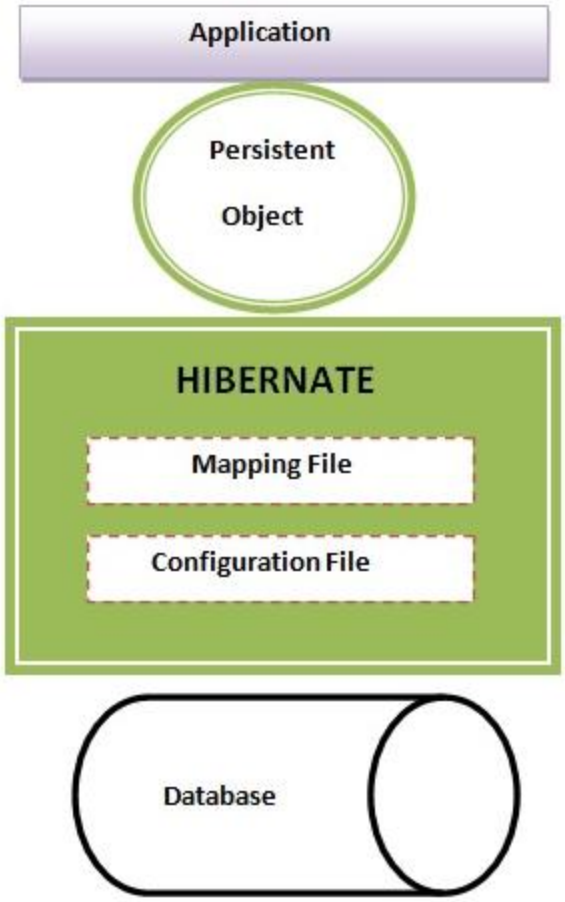
Query Object

- Query objects use SQL or Hibernate Query Language (HQL) string to retrieve data from the database and create objects.
- A Query instance is used to bind query parameters, limit the number of results returned by the query, and finally to execute the query.

Criteria Object

- Criteria object are used to create and execute object oriented criteria queries to retrieve objects.

Steps to create Hibernate Application



- Create the Persistent class
- Create the mapping file for Persistent class
- Create the Configuration file
- Create the class that retrieves or stores the persistent object
- Load the jar file
- Run the hibernate application

Create the Persistent class

- A simple Persistent class should follow some rules:
  - **A no-arg constructor:** It is recommended that you have a default constructor at least package visibility so that hibernate can create the instance of the Persistent class by newInstance() method.
  - **Provide an identifier property (optional):** It is mapped to the primary key column of the database.
  - **Declare getter and setter methods (optional):** The Hibernate recognizes the method by getter and setter method names by default.
  - **Prefer non-final class:** Hibernate uses the concept of proxies, that depends on the persistent class. The application programmer will not be able to use proxies for lazy association fetching.

```
public class Employee
{
private int id;
private String firstName,lastName;

public int getId()
{
   return id;
}
public void setId(int id)
 {
   this.id = id;
}
public String getFirstName()
 {
   return firstName;
}
public void setFirstName(String firstName)
 {
   this.firstName = firstName;
}
public String getLastName()
 {
   return lastName;
}
public void setLastName(String lastName)
 {
   this.lastName = lastName;
}

}
```

Create the mapping file for Persistent class

- The mapping file name conventionally, should be **class_name.hbm.xml**.
- There are many elements of the mapping file.
  - **hibernate-mapping** is the root element in the mapping file.
  - **class** It is the sub-element of the hibernate-mapping element. It specifies the Persistent class.
  - **id** It is the subelement of class. It specifies the primary key attribute in the class.
  - **generator** It is the subelement of id. It is used to generate the primary key. There are many generator classes such as assigned (It is used if id is specified by the user), increment, hilo, sequence, native etc. We will learn all the generator classes later.
  - **property** It is the subelement of class that specifies the property name of the Persistent class.

employee.hbm.xml

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-mapping PUBLIC
 "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
 "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

 <hibernate-mapping>
  <class name="com.javatpoint.mypackage.Employee" table="emp1000">
    <id name="id">
     <generator class="assigned"></generator>
    </id>

    <property name="firstName"></property>
    <property name="lastName"></property>

  </class>

 </hibernate-mapping>
```

Create the Configuration file

- Following is the list of important properties you would require to configure for a databases in a standalone situation:

S.N	Properties and Description
1	**hibernate.dialect** This property makes Hibernate generate the appropriate SQL for the chosen database.
2	**hibernate.connection.driver_class**The JDBC driver class.
3	**hibernate.connection.url**The JDBC URL to the database instance.
4	**hibernate.connection.username**The database username.
5	**hibernate.connection.password**The database password.
6	**hibernate.connection.pool_size**Limits the number of connections waiting in the Hibernate database connection pool.
7	**hibernate.connection.autocommit**Allows autocommit mode to be used for the JDBC connection.

hibernate.cfg.xml

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
     <session-factory>
       <property name="hbm2ddl.auto">update</property>
       <property name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
       <property name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>

       <property name="connection.username">system</property>
       <property name="connection.password">oracle</property>
       <property name="connection.driver_class">oracle.jdbc.driver.OracleDriver</proper
ty>
    <mapping resource="employee.hbm.xml"/>
    </session-factory>

</hibernate-configuration>
```

Create the class that retrieves or stores the object

In this class, we are simply storing the employee object to the database.

```
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;

public class StoreData {
public static void main(String[] args) {

    //creating configuration object
    Configuration cfg=new Configuration();
    cfg.configure("hibernate.cfg.xml");//populates the data of the configuration file

    //creating seession factory object
    SessionFactory factory=cfg.buildSessionFactory();

    //creating session object
    Session session=factory.openSession();
```

```
//creating transaction object
   Transaction t=session.beginTransaction();

   Employee e1=new Employee();
   e1.setId(115);
   e1.setFirstName("sonoo");
   e1.setLastName("jaiswal");

   session.persist(e1);//persisting the object

   t.commit();//transaction is commited
   session.close();

   System.out.println("successfully saved");

}
}
```

Load the jar file

- For successfully running the hibernate application, you should have the hibernate4.jar file.
- Some other jar files or packages are required such as
  - cglib
  - log4j
  - commons
  - SLF4J
  - dom4j
  - xalan
  - xerces

Spring Framework

Spring Framework

- It was **developed by Rod Johnson in 2003.**
- Spring is a *lightweight* framework.
- It can be thought of as a *framework of frameworks* because it provides support to various frameworks such as Struts, Hibernate, Tapestry, EJB, JSF etc.
- The framework, in broader sense, can be defined as a structure where we find solution of the various technical problems.
- The Spring framework comprises several modules such as IOC, AOP, DAO, Context, ORM, WEB MVC etc..

Inversion Of Control (IOC) and Dependency Injection

- The technology that Spring is most identified with is the **Dependency Injection (DI)** flavour of Inversion of Control.
- When writing a complex Java application, application classes should be as independent as possible of other Java classes to increase the possibility to reuse these classes and to test them independently of other classes while unit testing. Dependency Injection helps in gluing these classes together and at the same time keeping them independent.
- Dependency injection can happen in the way of passing parameters to the constructor or by post-construction using setter methods.

Inversion Of Control (IOC) and Dependency Injection

tight coupling

```
class Employee{
Address address;
Employee(){
address=new Address();
}
}
```

loose coupling /IOC

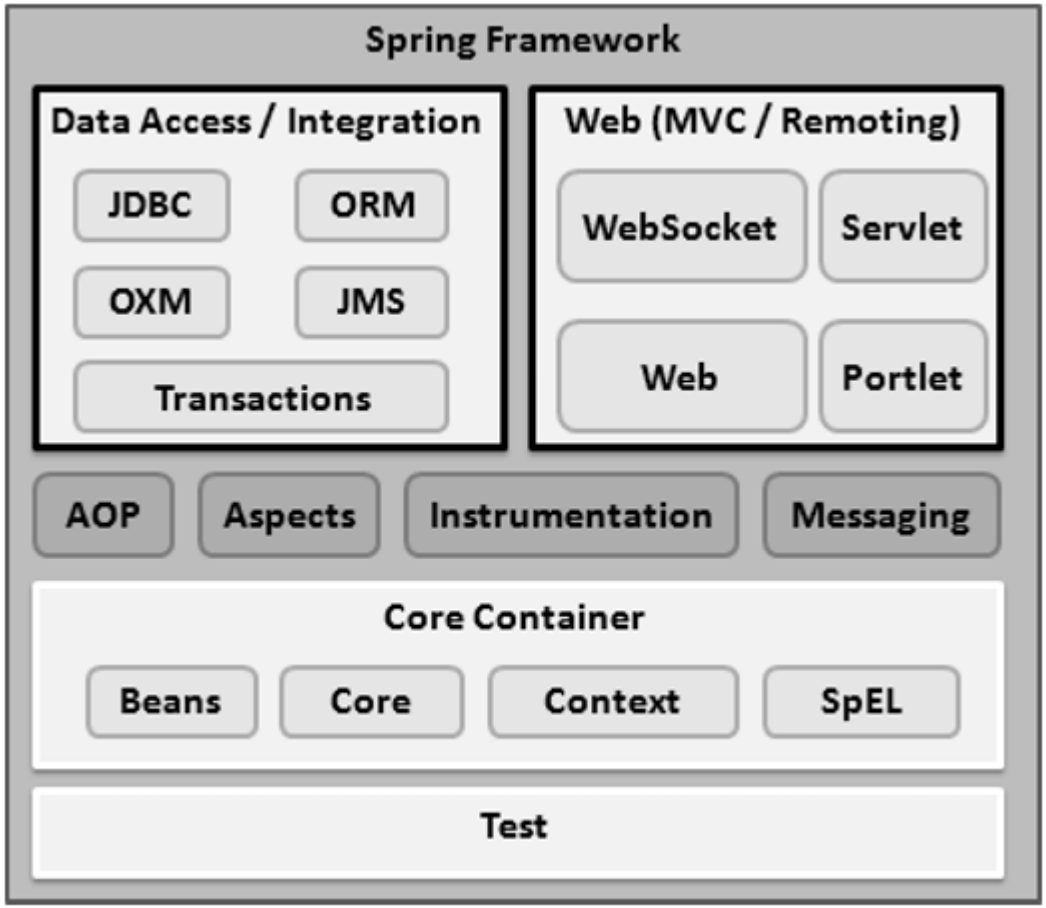
```
class Employee{
Address address;
Employee(Address address){
this.address=address;
}
}
```

- Thus, IOC makes the code loosely coupled. In such case, there is no need to modify the code if our logic is moved to new environment.
- In Spring framework, IOC container is responsible to inject the dependency. We provide metadata to the IOC container either by XML file or annotation.

Aspect Oriented Programming (AOP)

- The AOP module of Spring Framework provides an aspect-oriented programming implementation allowing you to define method-interceptors and point cuts to cleanly decouple code that implements functionality that should be separated.

Spring Framework - Architecture



Spring could potentially be a **one-stop shop for all your enterprise applications.** However, Spring is modular, allowing you to pick and choose which modules are applicable to you,

Core Container

- The Core Container consists of the Core, Beans, Context, and Expression Language modules the details of which are as follows −
  - The **Core** module provides the fundamental parts of the framework, including the IoC and Dependency Injection features.
  - The **Bean** module provides BeanFactory, which is a sophisticated implementation of the factory pattern.
  - The **Context** module builds on the solid base provided by the Core and Beans modules and it is a medium to access any objects defined and configured. The ApplicationContext interface is the focal point of the Context module.
  - The **SpEL** module provides a powerful expression language for querying and manipulating an object graph at runtime.

Data Access/Integration

- The Data Access/Integration layer consists of the JDBC, ORM, OXM, JMS and Transaction modules whose detail is as follows –
  - The **JDBC** module provides a JDBC-abstraction layer that removes the need for tedious JDBC related coding.
  - The **ORM** module provides integration layers for popular object-relational mapping APIs, including JPA, JDO, Hibernate, and iBatis.
  - The **OXM** module provides an abstraction layer that supports Object/XML mapping implementations for JAXB, Castor, XMLBeans, JiBX and XStream.
  - The Java Messaging Service **JMS** module contains features for producing and consuming messages.
  - The **Transaction** module supports programmatic and declarative transaction management for classes that implement special interfaces and for all your POJOs.

Web

- The Web layer consists of the Web, Web-MVC, Web-Socket, and Web-Portlet modules the details of which are as follows −
  - The **Web** module provides basic web-oriented integration features such as multipart file-upload functionality and the initialization of the IoC container using servlet listeners and a web-oriented application context.
  - The **Web-MVC** module contains Spring's Model-View-Controller (MVC) implementation for web applications.
  - The **Web-Socket** module provides support for WebSocket-based, two-way communication between the client and the server in web applications.
  - The **Web-Portlet** module provides the MVC implementation to be used in a portlet environment and mirrors the functionality of Web-Servlet module.

Miscellaneous

- There are few other important modules like AOP, Aspects, Instrumentation, Web and Test modules the details of which are as follows −
  - The **AOP** module provides an aspect-oriented programming implementation allowing you to define method-interceptors and pointcuts to cleanly decouple code that implements functionality that should be separated.
  - The **Aspects** module provides integration with AspectJ, which is again a powerful and mature AOP framework.
  - The **Instrumentation** module provides class instrumentation support and class loader implementations to be used in certain application servers.
  - The **Messaging** module provides support for STOMP as the WebSocket sub-protocol to use in applications. It also supports an annotation programming model for routing and processing STOMP messages from WebSocket clients.
  - The **Test** module supports the testing of Spring components with JUnit or TestNG frameworks.