# Developing Applications Using Java Web Frameworks

# Developing Applications Using Java Web Frameworks Trainer's Guide



**APTECH LIMITED**

Contact E-mail: ov-support@onlinevarsity.com

First Edition - 2015

**Dear Learner,**

We congratulate you on your decision to pursue an Aptech course.

Aptech Ltd. designs its courses using a sound instructional design model – from conceptualization to execution, incorporating the following key aspects:

- Scanning the user system and needs assessment

Needs assessment is carried out to find the educational and training needs of the learner.

Technology trends are regularly scanned and tracked by core teams at Aptech Ltd. TAG* analyzes these on a monthly basis to understand the emerging technology training needs for the Industry.

An annual Industry Recruitment Profile Survey# is conducted during August - October to understand the technologies that Industries would be adapting in the next 2 to 3 years. An analysis of these trends & recruitment needs is then carried out to understand the skill requirements for different roles & career opportunities.

The skill requirements are then mapped with the learner profile (user system) to derive the Learning objectives for the different roles.

- Needs analysis and design of curriculum

The Learning objectives are then analyzed and translated into learning tasks. Each learning task or activity is analyzed in terms of knowledge, skills and attitudes that are required to perform that task. Teachers and domain experts do this jointly. These are then grouped in clusters to form the subjects to be covered by the curriculum.

In addition, the society, the teachers, and the industry expect certain knowledge and skills that are related to abilities such as learning-to-learn, thinking, adaptability, problem solving, positive attitude etc. These competencies would cover both cognitive and affective domains.

A precedence diagram for the subjects is drawn where the prerequisites for each subject are graphically illustrated. The number of levels in this diagram is determined by the duration of the course in terms of number of semesters etc. Using the precedence diagram and the time duration for each subject, the curriculum is organized.

- Design & development of instructional materials

The content outlines are developed by including additional topics that are required for the completion of the domain and for the logical development of the competencies identified. Evaluation strategy and scheme is developed for the subject. The topics are arranged/organized in a meaningful sequence.

The detailed instructional material – Training aids, Learner material, reference material, project guidelines, etc.- are then developed. Rigorous quality checks are conducted at every stage.

- Strategies for delivery of instruction

Careful consideration is given for the integral development of abilities like thinking, problem solving, learning-to-learn etc. by selecting appropriate instructional strategies (training methodology), instructional activities and instructional materials.

The area of IT is fast changing and nebulous. Hence, considerable flexibility is provided in the instructional process by specially including creative activities with group interaction between the students and the trainer. The positive aspects of Web based learning –acquiring information, organizing information and acting on the basis of insufficient information are some of the aspects, which are incorporated, in the instructional process.

- Assessment of learning

The learning is assessed through different modes – tests, assignments & projects. The assessment system is designed to evaluate the level of knowledge & skills as defined by the learning objectives.

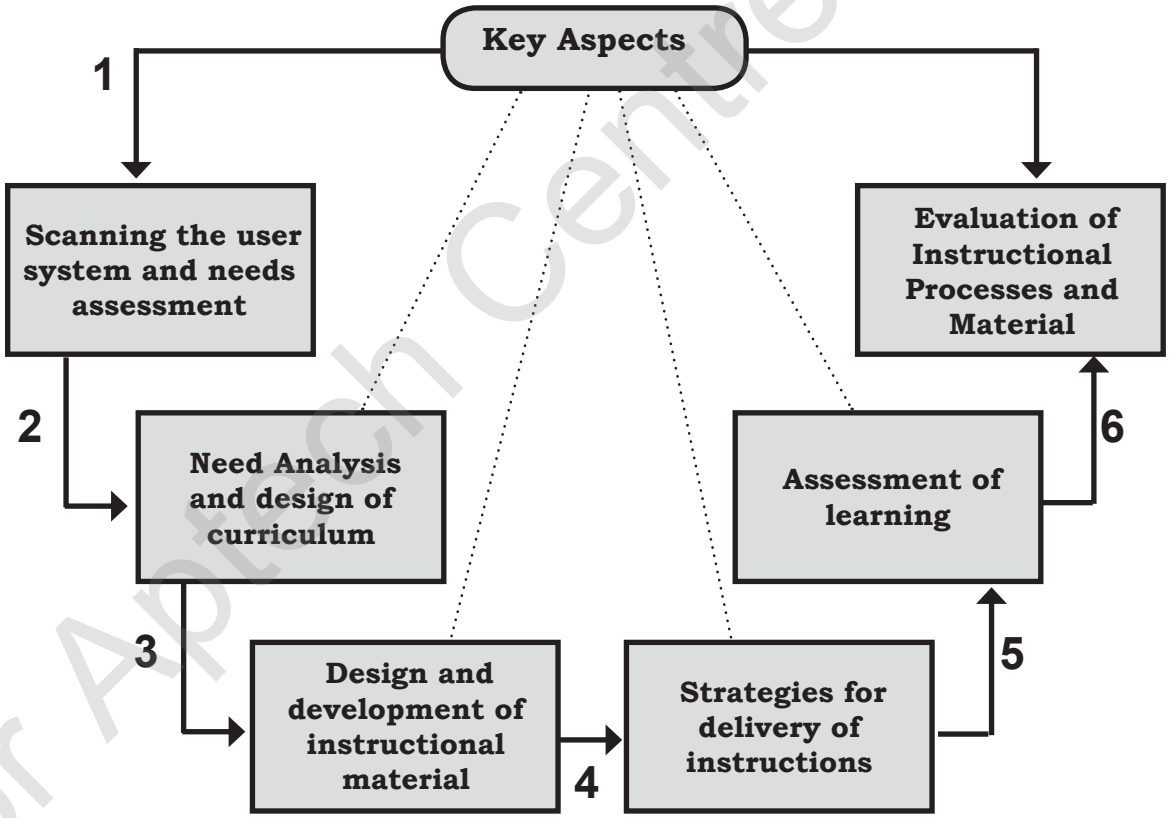
- Evaluation of instructional process and instructional materials

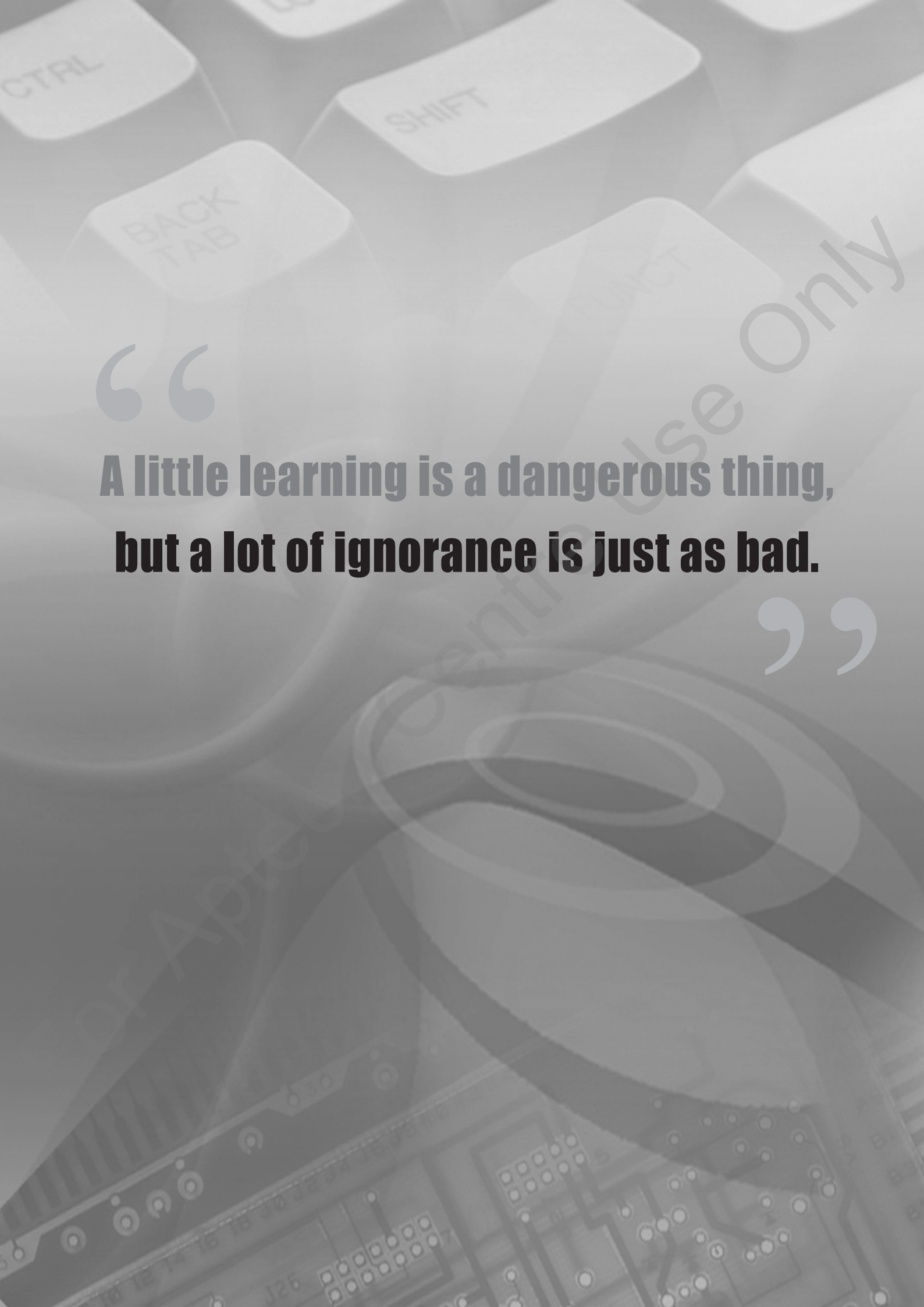
The instructional process is backed by an elaborate monitoring system to evaluate - on-time delivery, understanding of a subject module, ability of the instructor to impart learning. As an integral part of this process, we request you to kindly send us your feedback in the reply pre-paid form appended at the end of each module.

*TAG – Technology & Academics Group comprises of members from Aptech Ltd., professors from reputed Academic Institutions, Senior Managers from Industry, Technical gurus from Software Majors & representatives from regulatory organizations/forums.

Technology heads of Aptech Ltd. meet on a monthly basis to share and evaluate the technology trends. The group interfaces with the representatives of the TAG thrice a year to review and validate the technology and academic directions and endeavors of Aptech Ltd.

**Aptech New Products Design Model**






“A little learning is a dangerous thing,
but a lot of ignorance is just as bad.”

# Preface

The book 'Developing Applications Using Java Web Frameworks' Trainer's Guide aims to teach the students how to develop rich and interactive Web applications using JSF and Struts frameworks. The faculty/trainer should teach the concepts in the theory class using the slides. This Trainer's Guide will provide guidance on the flow of the module and also provide tips and additional examples wherever necessary. The trainer can ask questions to make the session interactive and also to test the understanding of the students.

This book is the result of a concentrated effort of the Design Team, which is continuously striving to bring you the best and the latest in Information Technology. The process of design has been a part of the ISO 9001 Certification for Aptech-IT Division, Education Support Services. As part of Aptech's quality drive, this team does intensive research and curriculum enrichment to keep it in line with industry trends.

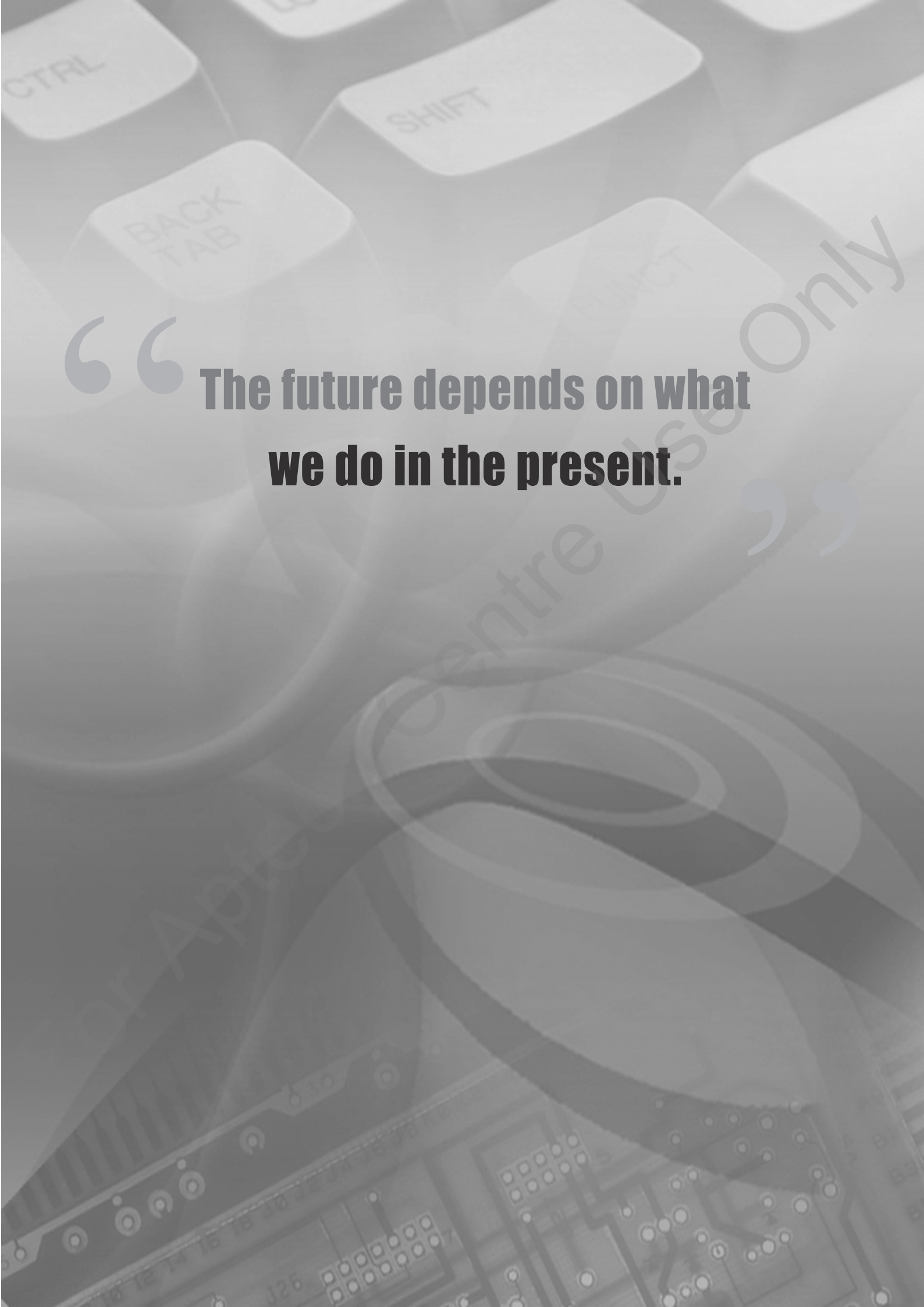
We will be glad to receive your suggestions.

Design Team


“Practice is the best of all instructors.

# Table of Contents

## Sessions




“The future depends on what we do in the present.”

# Session 1 – Introduction to Struts Framework

## 1.1 Pre-Class Activities

Familiarize yourself with the topics of this session in-depth. Prepare a question or two which will be a key point to relate the current session objectives.

### 1.1.1 Objectives

By the end of this session, the learners will be able to:

- Explain Model-View-Controller (MVC) architecture
- Explain the role of components involved in MVC architecture
- Explain JSP Model 1 and JSP Model 2 approaches
- Describe Struts framework
- List the features and components of Struts 1 framework
- Explain the architecture of Struts 1 framework
- Explain the process of developing Web applications using Struts 1 components

### 1.1.2 Teaching Skills

To teach this session successfully, you should be aware of Model-View-Controller (MVC) architecture and the role of components involved in MVC architecture. Familiarize yourself with Struts framework, features and components of Struts 1 framework, and the architecture of Struts 1 framework.

To teach this session, you should also aware yourself with the process of developing Web applications using Struts 1 components.

For teaching in the class, you are expected to use slides and LCD projectors.

**Tips:**

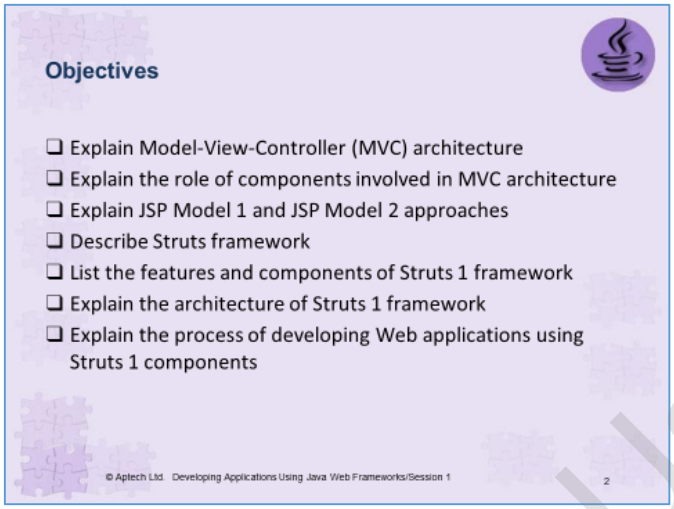
It is recommended that you test the understanding of the students by asking questions in between the class.

**In-Class Activities:**

Follow the order given here during In-Class activities.

**Overview of the Session:**

Give the students a brief overview of the current session in the form of session objectives. Show the students slide 2 of the presentation.



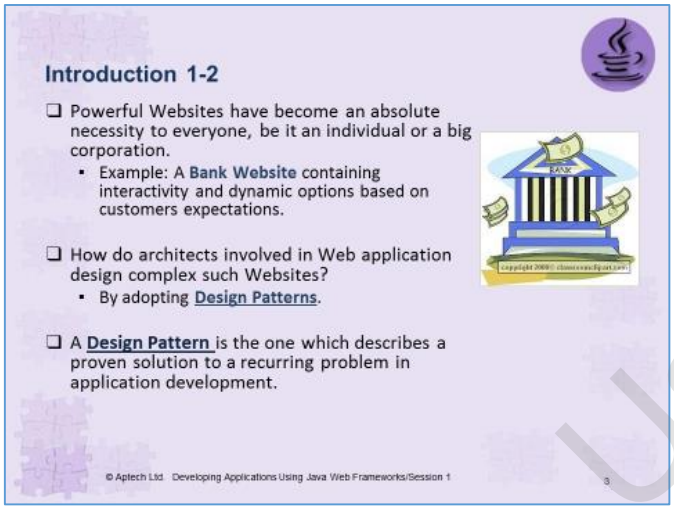

Tell the students that they will be introduced to Model-View-Controller (MVC) architecture and the role of components involved in MVC architecture. Explain JSP Model 1 and JSP Model 2 approaches.

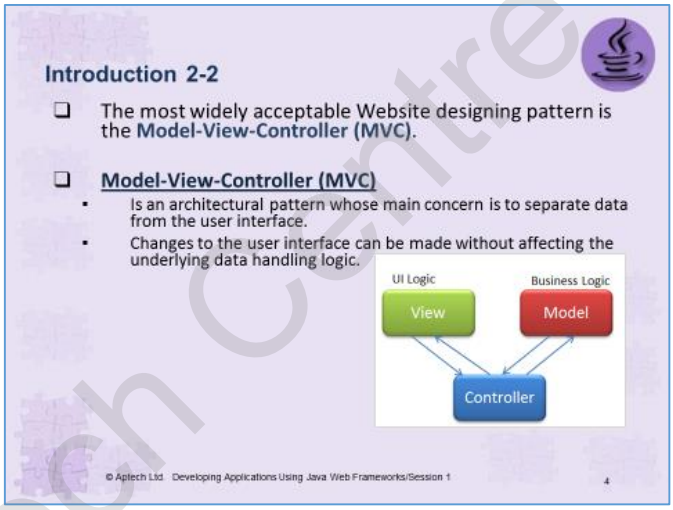
In this session, they will learn Struts framework, features and components of Struts 1 framework, and the architecture of Struts 1 framework. Finally, the session discusses about the process of developing Web applications using Struts 1 framework components.

## 1.2 In-Class Explanations

### Slides 3 and 4

Let us understand about design patterns.




Use slides 3 and 4 to introduce the design patterns.

Tell them that the use of Internet has brought a major shift in the ways of performing trade and communication over the large networks. Today, every business feels the need to have a presence in the virtual world through the Internet. Thus, building of powerful Websites have become an absolute necessity to everyone.

Then, to prove the use of Internet and influenced businesses, discuss the example of Bank Website. Almost all popular banks have hosted their Websites. These Websites provide various interactive and dynamic options based on the customized needs and requirements of the customers. One of the key areas of designing a bank Website is its offerings provided on the various business needs. Thus, designing a Website with reusable software components is a difficult task.

Normally to handle complex needs of the business, applications are designed based on design patterns. A design pattern is a well-proved solution for solving the repetitive problems occurring in the application development.

Design Patterns is used for two main purposes in software development. They are as follows:

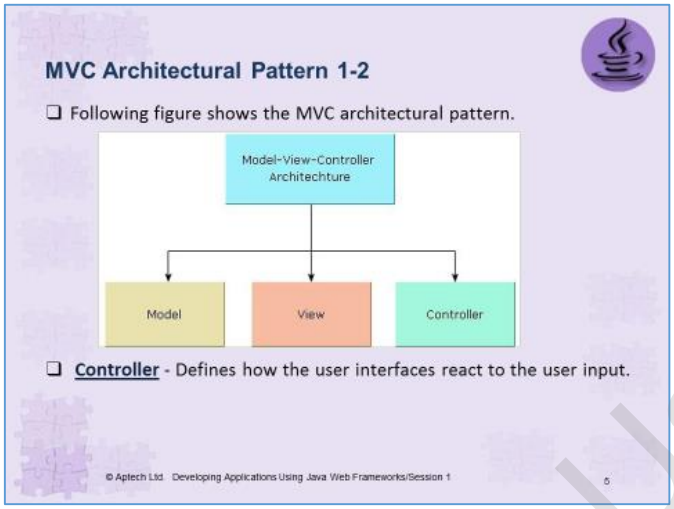
1. Design patterns provide a standard terminology and are specific to particular scenario.
2. Design patterns provide best solutions to certain problems faced during software development.

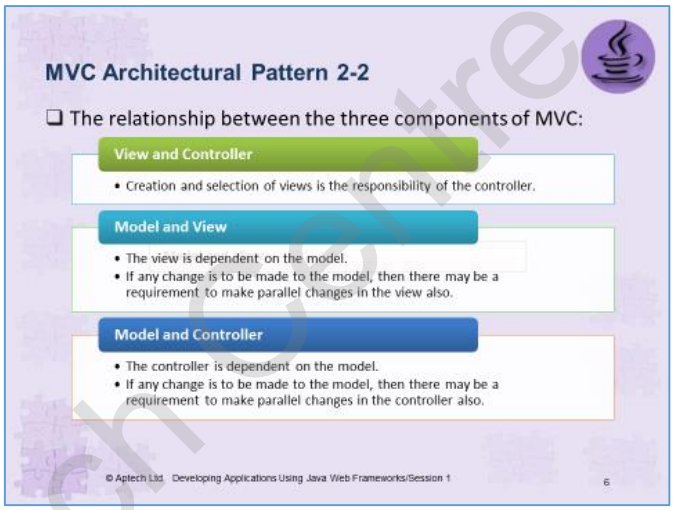
In software designing, design patterns are strategies which are language-independent and used for solving commonly encountered object-oriented design problems. Design patterns are proven techniques for implementing robust, reusable, and extensible object-oriented software. In more technical terms, a pattern is a proven solution to a problem which has been documented so that it can be used to easily handle similar problems in the future.

One such design pattern, is Model–view–controller (MVC) is a software architectural pattern for implementing user interfaces. MVC is adopted as a popular architecture for World Wide Web (WWW) applications. It divides a given software application into three interconnected parts, so as to separate internal representations of information from the ways that information is presented to or accepted from the user.

## Slides 5 and 6

Let us understand MVC architectural pattern.

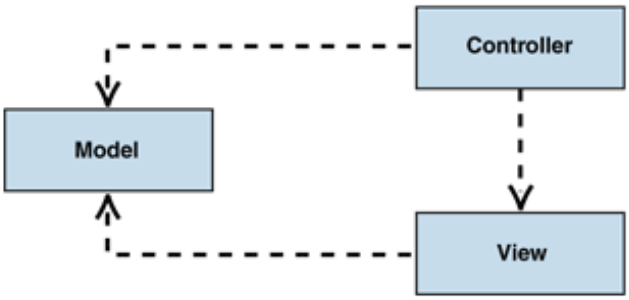



Use slides 5 and 6 to explain MVC architectural pattern.

Tell them that MVC architectural pattern enables the separation of data (Model) from the user interface (View). This is made possible by introducing an intermediate component called the Controller. The Controller defines how the user interfaces react to the user input.

The following figure depicts the structural relationship between the three objects:



Then, describe how these components are related to each other.

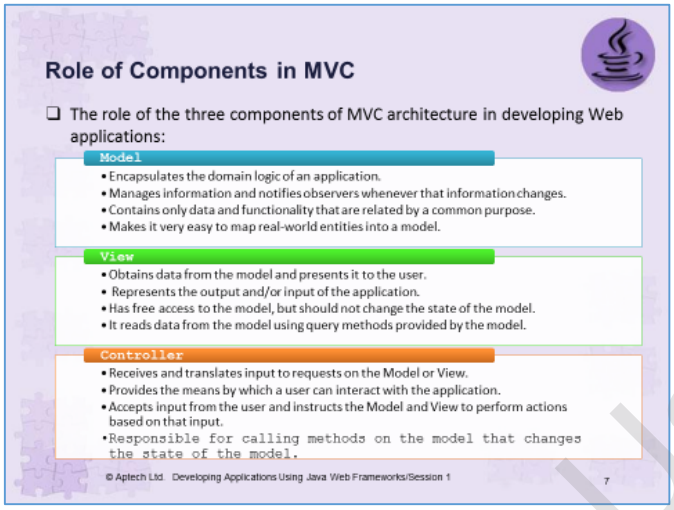
- **Model** - The model manages the behavior and data of the application domain. The View is dependent on the Model. If any change is to be made to the Model, then there may be a requirement to make parallel changes in the View also.
  The Controller is dependent on the Model. If any change is to be made to the Model, then there may be a requirement to make parallel changes in the Controller also.

- **View** - The View manages the display of information to the client.

- **Controller** - The Controller interprets the inputs from the user, informing the Model and/or the View to change as appropriate. Creation and selection of View is the responsibility of the Controller.

## Slide 7

Let us understand the role of Components in MVC.




Use slide 7 to explain the role of the components in MVC.

**Model**

- It encapsulates the domain logic of an application.
- It manages information and notifies observers whenever that information changes.
- It contains only data and functionality that are related by a common purpose. If you need to model two groups of unrelated data and functionality, you create two separate models.
- It makes it very easy to map real-world entities into a model.
  - View renders contents of model for user.
  - When model changes, view must be updated.

**Controller**

- It receives and translates input to requests on the Model or View.
- It provides the means by which a user can interact with the application.
- It accepts input from the user and instructs the Model and View to perform actions based on that input.
- It maps the end-user action to the application response. For example, if the user chooses a menu item, the Controller is responsible for determining how the application should respond.

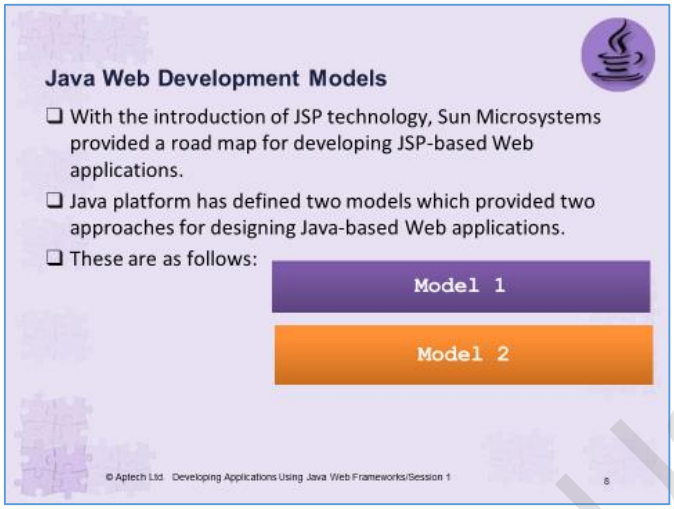
- It is responsible for calling methods on the model that changes the state of the model.

**View**

- It obtains data from the model and presents it to the user. The View represents the output and/or input of the application.
- It generally has free access to the model, but should not change the state of the model.
- It reads data from the model using query methods provided by the model.

## Slide 8

Let us understand Java Web development models.




Use slide 8 to explain the Java Web development models.

Tell them that Sun Microsystems defined two models which provided approaches for designing Java-based Web applications.

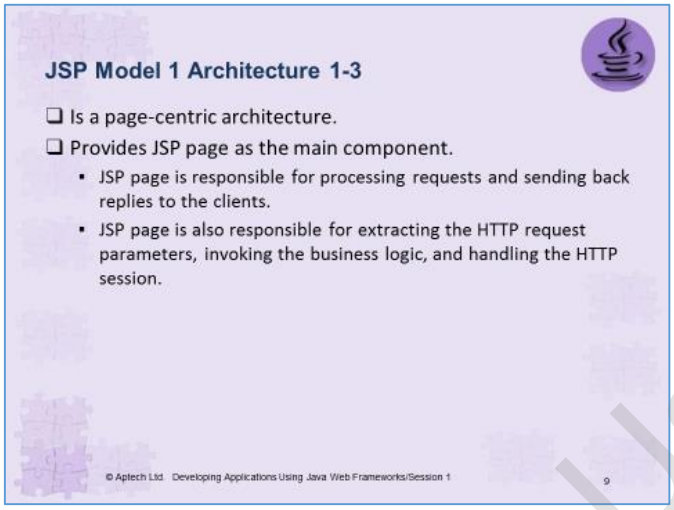
These are as follows:

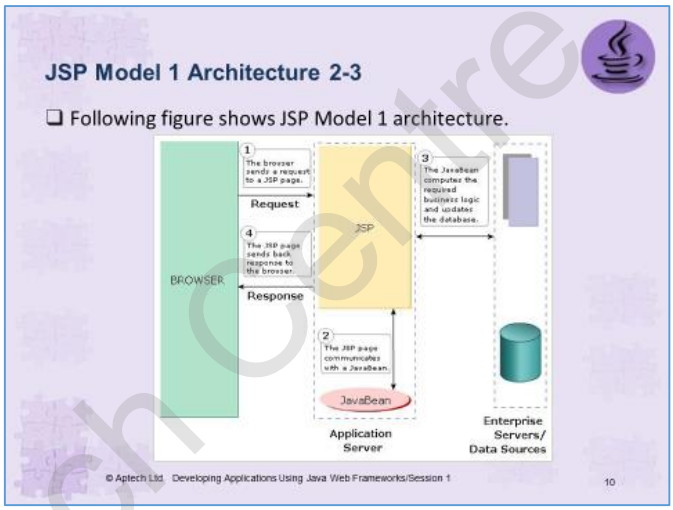
- **Model 1** – This was provided as a primary solution which was given, when JSP technology was introduced.

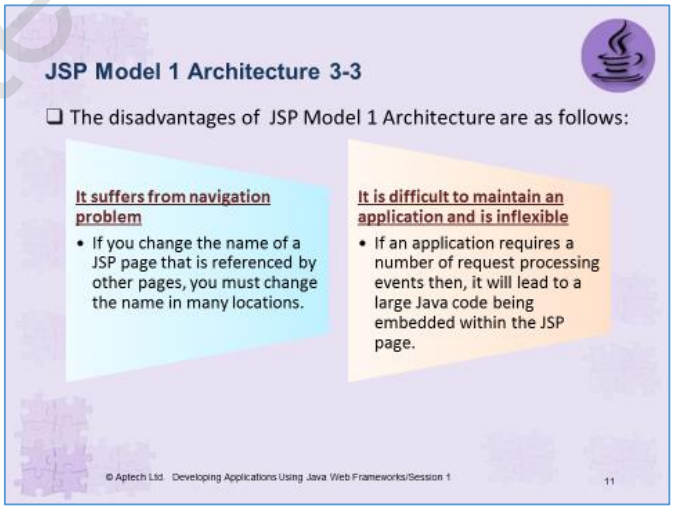
- **Model 2** – This model was introduced and presented as a correct way for designing the Web applications based on Servlet and JSP technology. Further, Model 2 was adopted as base model by many MVC-based Web frameworks such as Struts.

## Slides 9 to 11

Let us understand JSP Model 1 architecture.




Use slides 9 to 11 to explain JSP Model 1 architecture.

Model 1 architecture is one of the approaches for developing a Web application. In Model 1 architecture, the JSP page is not only responsible for processing requests and sending back replies to the clients but also for extracting the HTTP request parameters, invoking the business logic and handling the HTTP session. Figure 1.2 shows Model-1 architecture.

**Purpose of Model 1**

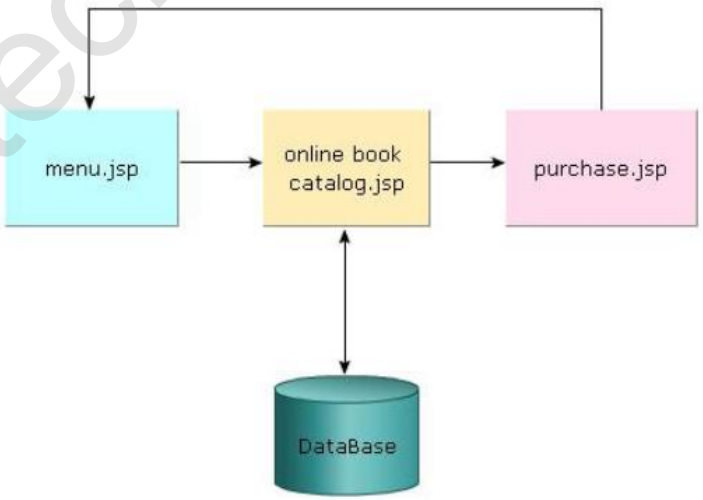
The purpose of Model 1 architecture is to enable Web designers to develop Web applications such that it separates 'business logic' from 'presentation logic'.
Business logic deals with the method of modeling real world business objects such as accounts, loans, travel, and such others in the application. It also deals with the storage mechanism for these objects, object interactions, access, and update rights for them.
Presentation logic deals with methods of displaying these objects. For example, decisions related to displaying user accounts in a list form.

**Model 1 - Architecture**

Using slide 10, explain the model 1 architecture. Tell them that in this model, there is no extra servlet involved in the process. The client request is sent directly to a JSP page, which may communicate with JavaBeans or other services, but ultimately the JSP page selects the next page for the client to view.

JSP Model 1 architecture has a page centric architecture. In this architecture, the application is composed of a series of interrelated JSP pages and these JSP pages handle all aspects of application including presentation, control, and the business logic.

In page-centric architecture, the business process logic and control decisions are hard-coded inside JSP pages in the form of JavaBeans, scriptlets, and expressions. The following figure shows an example of Model 1 architecture:




The main problem in Servlet technology is that Servlet needs to recompile when any modification is done in the Servlet code. It does not provide separation of concern. The presentation and business logic are mixed up in the Servlet code.

JSP removes all the problems of Servlet. It provides better separation of concern, so that presentation and business logic is separated. User does not need to redeploy the application if JSP page is modified. JSP provides support to develop Web application using JavaBean, custom tags, and JSTL so, that user can put the business logic separate from the JSP and it is also easier to test and debug.

The Model 1 architecture is commonly used in smaller, simple task applications due to its ease of development.

Some of the advantages of using JSP Model 1 Architecture are as follows:

- It provides a more lightweight design for small and static applications.
- It is suitable for applications that have very simple page flow, have little need for centralized security control or logging, and changes little over time.
- It provides separation of presentation from content.

Model 1 applications can often be refactored to Model 2 when application requirements change.

Then using slide 11, explain the disadvantages of Model 1 architecture.

**In-Class Question:**

After you finish explaining Model 1 architecture, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.

Which of the following statements with respect to JSP Model 1 Architecture are true?

a. Model 1 architecture suffers from navigation problem.

b. Model 1 architecture business logic and control-decisions are hard-coded inside JSP pages.

c. In Model 1 architecture the presentation logic cannot be displayed.

d. In Model 1 architecture it is inefficient to model the business logic.

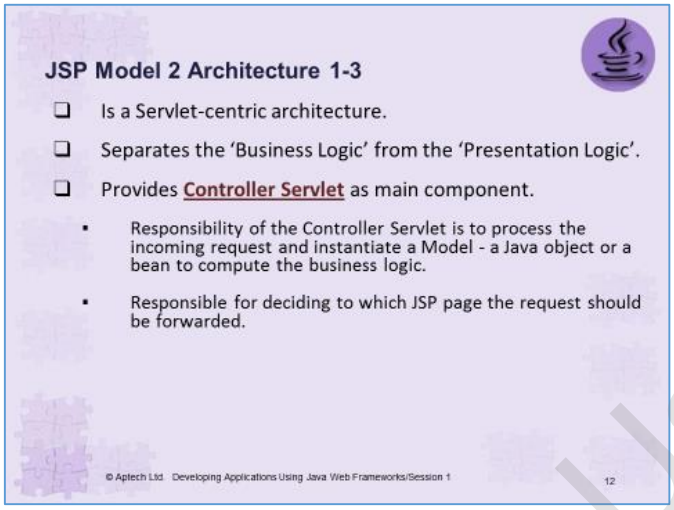
e. Model 1 architecture can process business logic.

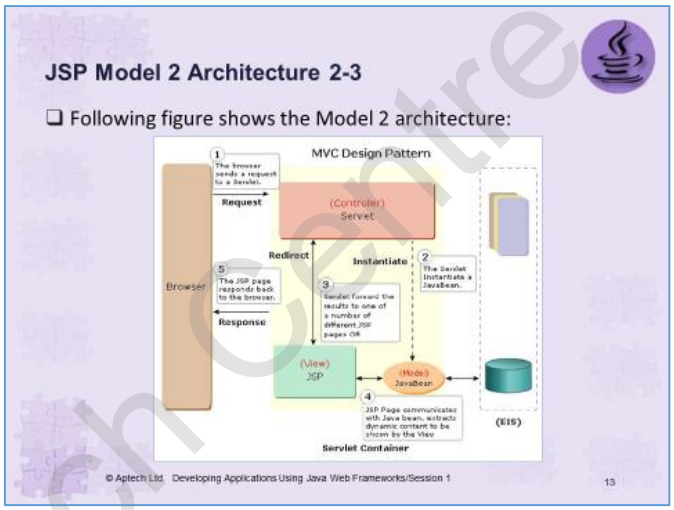
**Answer:**

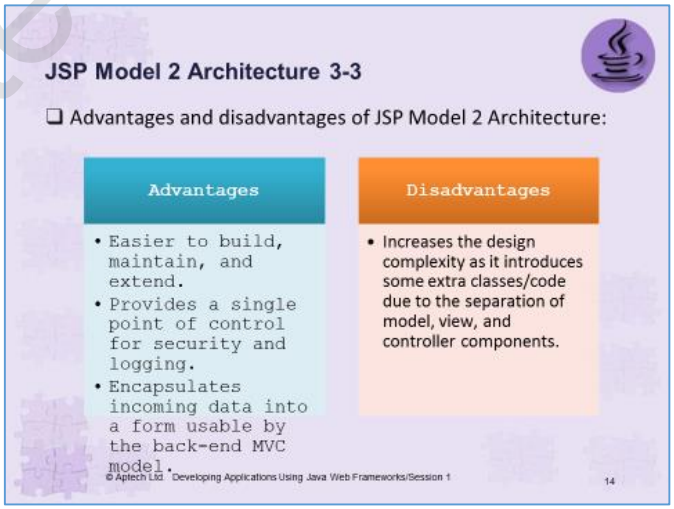
a-true, b-true, c-false, d-false, and e-true

## Slides 12 to 14

Let us understand JSP Model 2 Architecture.




Use slides 12 to 14 to explain JSP Model 2 Architecture.

Model 2 architecture is an approach used for developing a Web application. It separates the 'Business Logic' from the 'Presentation Logic'. Besides this, Model 2 has an additional component - a Controller.

Here, a Servlet acts as a Controller. The responsibility of the Controller Servlet is to process the incoming request and instantiate a Model - a Java object or a bean to compute the business logic. It is also responsible for deciding to which JSP page the request should be forwarded.

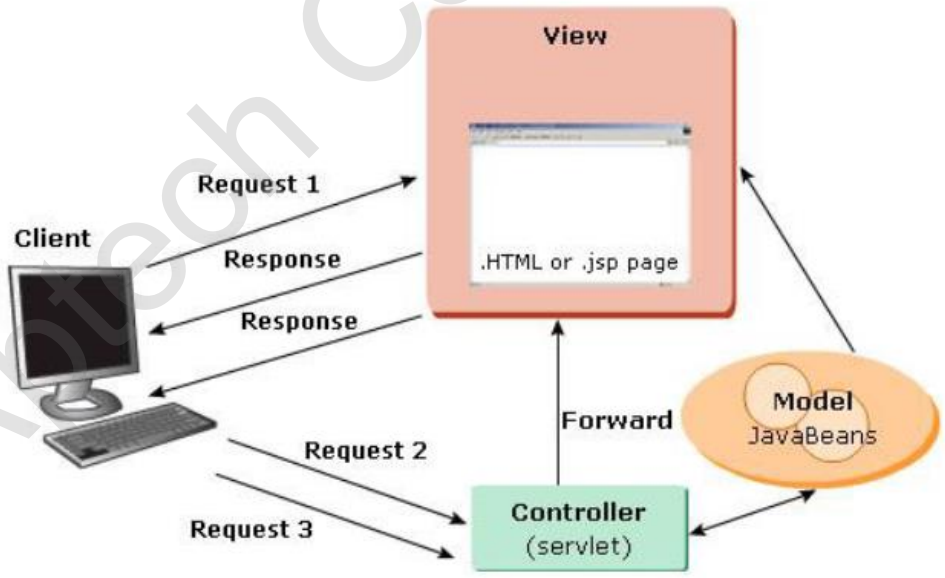
JSP page is responsible for the View component and retrieves the objects created by the Servlet and extracts dynamic content for insertion within a template for display.

Explain the figure provided on slide 13 to explain the components of the Model 2.

**Model 2 - Architecture**

Model 2 has Servlet-centric architecture. The Servlet acts as controller and selects suitable business logic to handle the request. It also redirects the results to appropriate view. JSP acts as View for response generation. Such a separation of business and presentation logic makes it possible to accommodate multiple interfaces to a Web application. These may include Web or wireless or GUI.

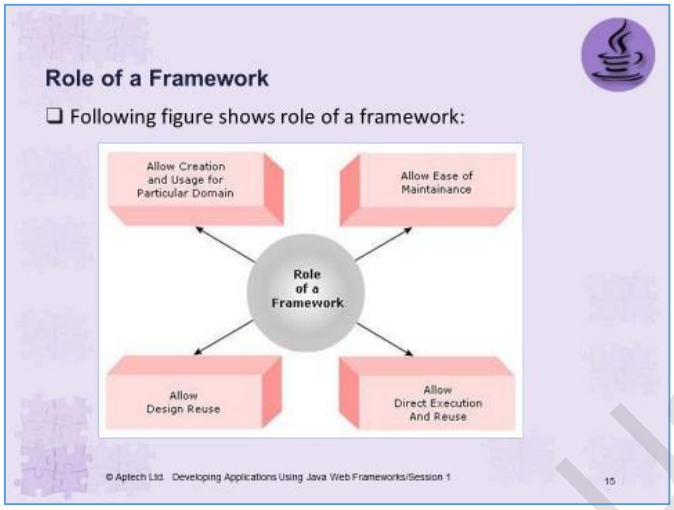
The following figure shows the components of MVC architecture:




Then, using slide 14, explain the advantages and disadvantages of JSP Model 2.

## Slide 15

Let us understand the role of a framework.



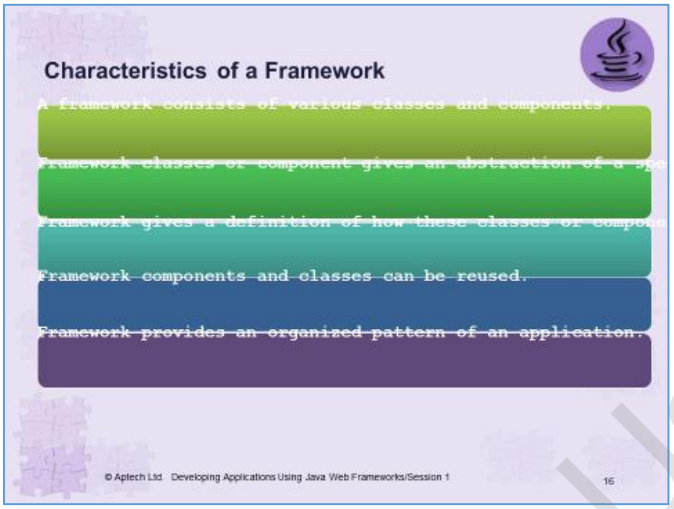

Use slide 15 to explain the role of a Framework. A software framework is a universal, reusable software environment. It provides particular functionality as part of a larger software platform to facilitate development of software applications, products, and solutions. Software frameworks includes support programs, compilers, code libraries, tool sets, and Application Programming Interfaces (APIs) that bring together all the different components to enable development of a project or solution.

Role of a framework are as follows:

- They propagate design reuse over code reuse.
- They can be written in any programming language such as Java and C++ and thus can be executed and reused directly.
- They are created and used for a particular application domain.
- They not only allow building application faster, but the applications have similar structure and are easier to maintain.

## Slide 16

Let us understand the characteristics of a framework.



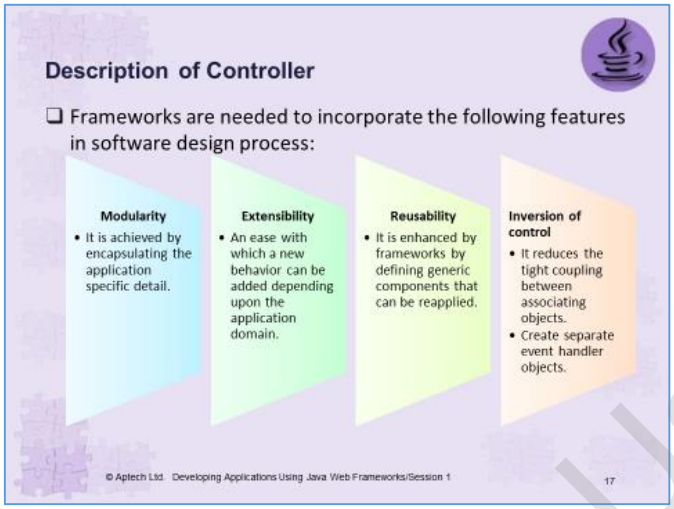

Use slide 16 to explain the characteristics of a framework.

The different characteristics of a framework are as follows:

- A framework consists of various classes and components. Each of these classes or component gives an abstraction of a specific concept.
- A framework also gives a definition of how these classes or components work with each other to provide a solution to a problem.
- Framework components and classes can be reused. This is because framework provides a general behavior that various applications can use it.
- Framework provides an organized pattern of an application.

## Slide 17

Let us understand Controller.



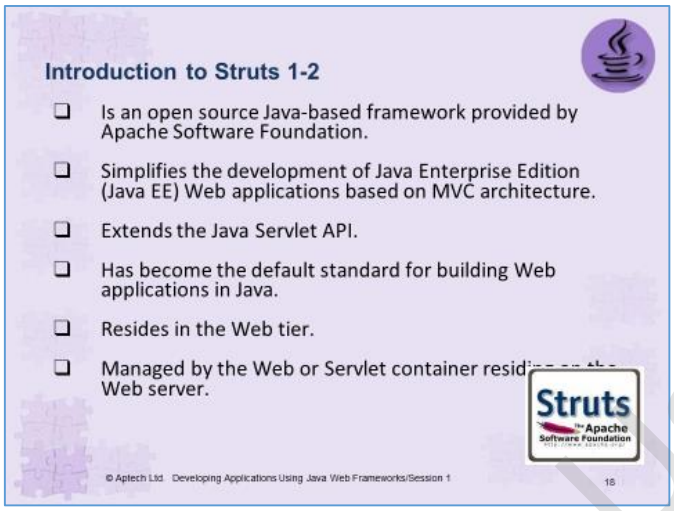

Use slide 17 to explain Controller.

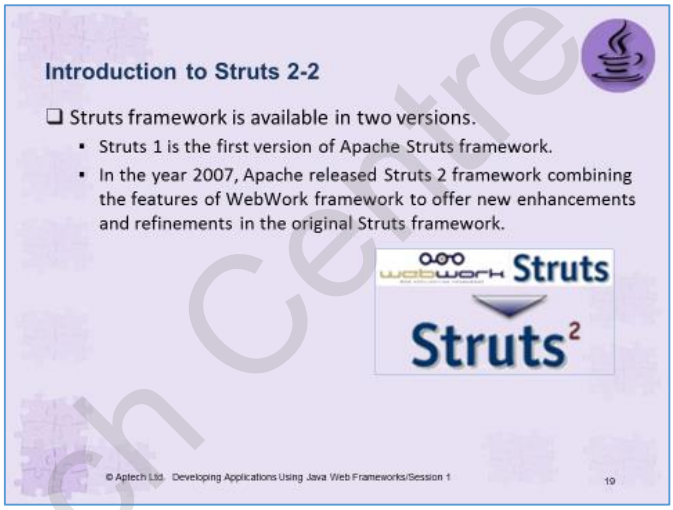
Following features are incorporated in software design process:

- **Modularity** - Modularity is achieved by encapsulating the application specific detail, which may change and separate it from a stable interface.
- **Reusability** - Design level reusability is enhanced by frameworks by defining generic components that can be reapplied to create new applications.
- **Inversion of control** - Inversion of Control is the design pattern, which reduces the tight coupling between associating objects by creating separate event handler objects.
- **Extensibility** - Extensibility in design means the ease with which a new behavior can be added depending upon the application domain.

## Slides 18 and 19

Let us understand Struts.



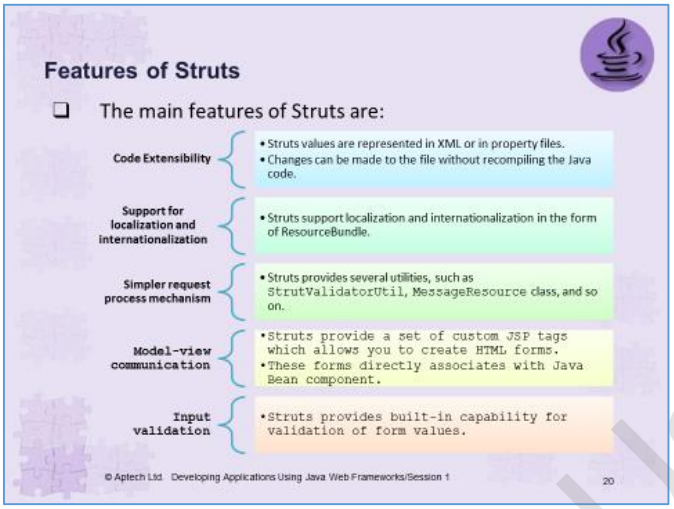

Use slides 18 and 19 to explain Struts. The purpose of this framework is to simplify the development of Java Enterprise Edition (Java EE) Web applications based on MVC architecture. Struts extends the Java Servlet API and has become the default standard for building Web applications in Java.

The Struts framework was developed by Craig McClanahan and supported by Apache Software Foundation's (ASF) Jakarta group. Struts 1 is the first version of Apache Struts framework. Later in the year 2007, Apache released Struts 2 framework combining the features of WebWork framework to offer new enhancements and refinements in the original Struts framework.

Struts framework basically consists of two entities - first, it is an MVC Web application that can be extended or added by the applications. Second, it provides a set of libraries, as tool set to build Web applications.

## Slide 20

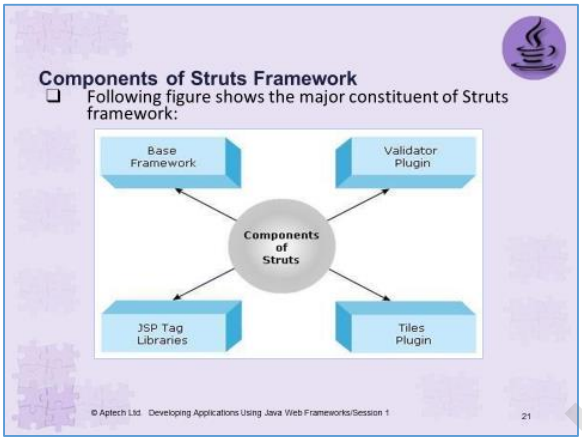
Let us understand the features of Struts.




Use slide 20 to explain the features of struts.

## Slide 21

Let us understand the components of Struts Framework.




Use slide 21 to explain the components of Struts Framework.

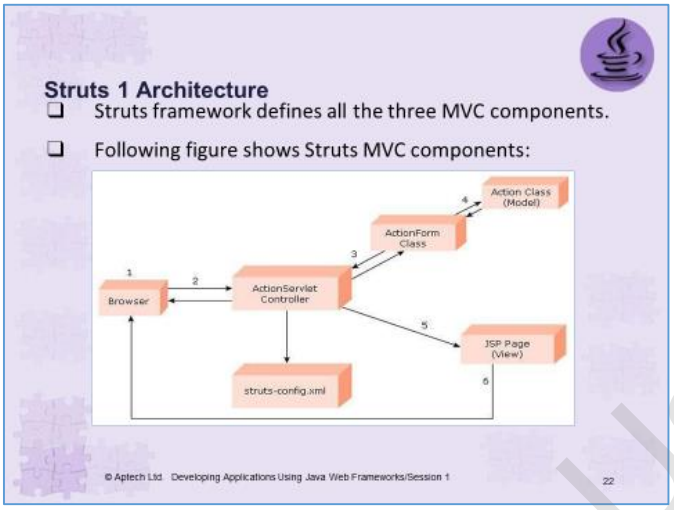
The Struts framework is a rich collection of Java libraries and can be broken down into the following major pieces:

- Base framework: The base framework provides the core MVC functionality and is comprised of the building blocks for a user's application. At the foundation of the base framework is the Controller servlet: `ActionServlet`. The rest of the base framework is comprised of base classes that a user's application will extend and several utility classes.

  Most important base classes are the `Action` and `ActionForm` classes. These two classes are used widely in all Struts applications. `Action` classes are used by `ActionServlet` to process specific requests. `ActionForm` classes are used to capture data from HTML forms and to be a conduit of data back to the View layer for page generation.
- JSP tag libraries:
  - **HTML**: Used to generate HTML forms that interact with the Struts APIs.
  - **Bean:** Used to work with Java bean objects in JSPs, such as accessing bean values.
  - **Logic:** Used to cleanly implement simple conditional logic in JSPs.
  - **Nested:** Used to allow arbitrary levels of nesting of the HTML, Bean, and Logic tags that otherwise do not work.
- Tiles plugin: Tiles is a rich JSP templating framework that facilitates the reuse of presentation (HTML) code.
- Validator plugin: Validator provides a rich framework for performing data validation on both the server side and client side (browser).

## Slide 22

Let us understand Struts 1 architecture.




Use slide 22 to explain Struts 1 architecture.

- `ActionServlet:` `ActionServlet` resides in the package `org.apache.struts.action` and it is the only one controller servlet for the entire Web application. The struts framework provides a controller servlet (ActionServlet) which is defined in the Struts libraries that are included in the IDE and it is configured as Servlet in the web.xml file.

  The `ActionServlet` instantiates a Handler. The Handler class name is obtained from an XML file based on the URL path information. This XML file is referred to as Struts configuration file and by default named as `struts-config.xml`. The Handler is called Action in the Struts terminology. This class is created by extending the Action class in `org.apache.struts.action.Action package.`

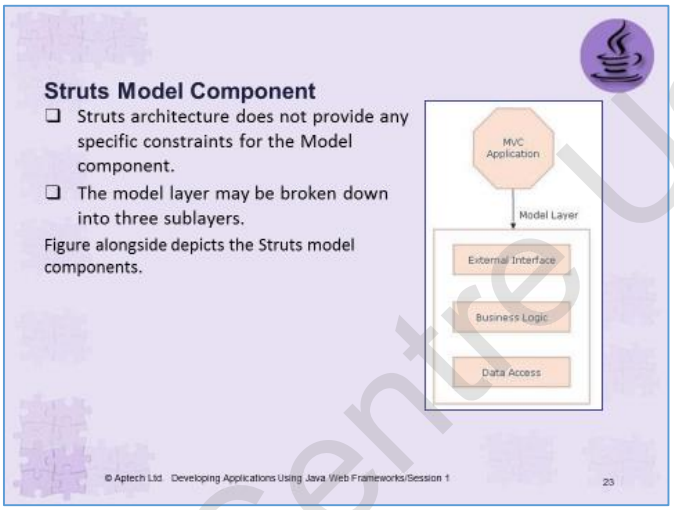
  The controller Servlet uses a `struts-config.xml` file to map incoming requests to Struts Action objects and instantiate any `ActionForm` objects.

- `ActionForm`: A Struts `ActionForm` is used to persist data between requests.

- `Action` class: This class is created by extending the Action class in `org.apache.struts.action` package. The `Action` object processes the data stored in the form bean using `execute()` method. Once the `Action` object processes a request, it forwards the results to the appropriate view.

## Slide 23

Let us understand Struts Model Component.




Use slide 23 to explain Struts Model Component.

Tell the students that in an MVC application, application developer may use technology such as Enterprise Java Bean (EJB), Java Data Object (JDO), or the Data Access Objects (DAO) pattern to implement the Model component.

The Model layer is divided into three sub layers:

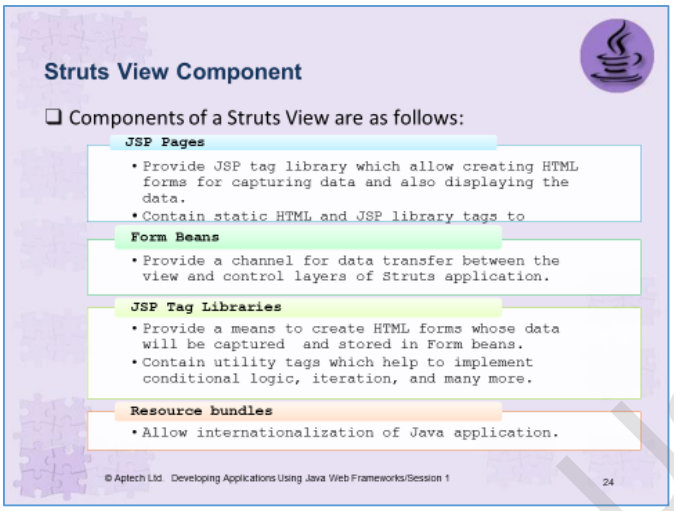
- **External interface**: Composed of code that provides an interface that external code uses to interact with the Model.

- **Business logic**: Encompasses the bulk of the Model code and provides the business functionality for an application.

- **Data access**: Composed of code for communicating with an application's data sources such as databases.

## Slide 24

Let us understand Struts View Component.




Use slide 24 to explain Struts View Component. Tell the students in an MVC application, the View component provides an interface to the user's application. It does not contain business logic, such as calculating interest for a financial application.

The View layer also does not contain any code for persisting data to or retrieving data from a data source. There are many forms that the View component of a Struts application can take. It can be HTML/JSP or it can be XML/XSLT, Velocity, Swing, or whatever the application requires.

Struts' HTML/JSP View layer support is divided into the following major components:

- JSP pages
- Form Beans
- JSP tag libraries
- Resource bundles

**JSP Pages**

JSP are used mainly for displaying data and capturing data. Struts provide a set of tag libraries that supports displaying data and creating HTML forms that capture data. Additionally, the tags support displaying content stored in resource bundles. Therefore, JSP with Form Beans provide the bulk of the Struts View layer. The JSP tag libraries connects those two together and the resource bundles provide a means of content management.

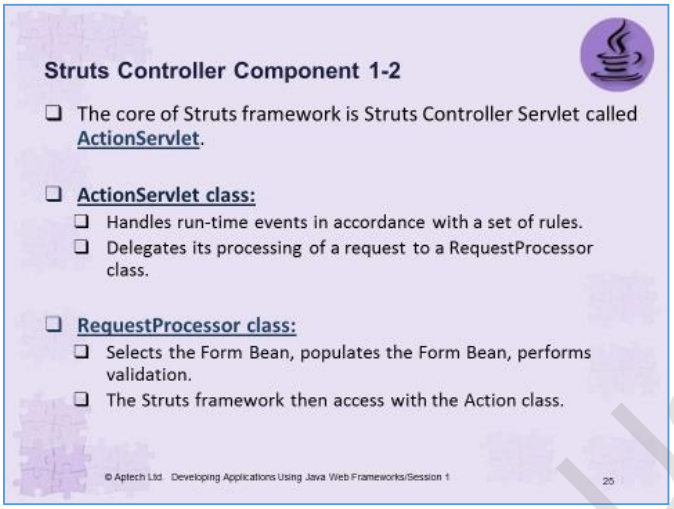
**Form Beans**

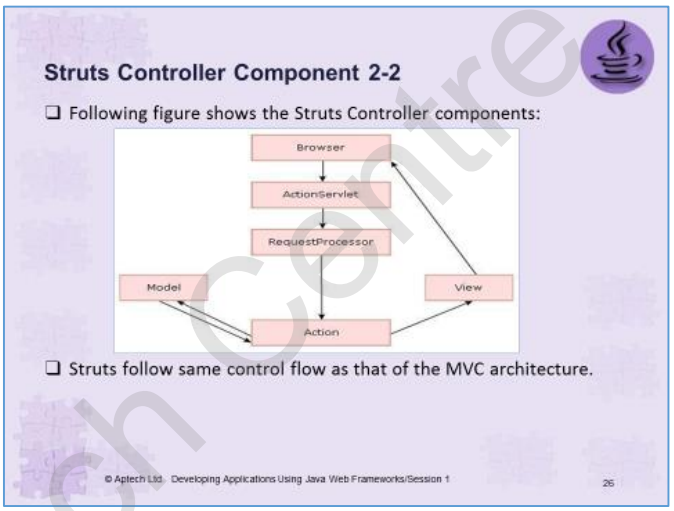
Form Beans are basic Java beans with getter and setter methods for each of their properties, allowing their data to be set and retrieved easily. The `org.apache.struts.action.ActionForm` class is the base abstract class that all Form Beans must subclass.

Form Beans are principally comprised of fields, and getter and setter methods for those fields. Business logic and data access code should not be placed in these classes. That code goes in Model layer classes. The only other methods that should be in these classes are helper methods or methods that override `ActionForm`'s base `reset( )` and `validate( )` methods.

## Slides 25 and 26

Let us understand Struts Controller Component.




Use slides 25 and 26 to explain Struts Controller Component. Tell the students the Controller component of an MVC application is responsible for creating the abstraction between the Model and View layers. This abstraction is the foundation of the MVC design pattern. The Controller is the central point of access to the application. All requests to an MVC Web application flow through the controller. Controller layer provides processing common to all requests, such as security, caching, logging, and so on.

Controller servlet, `ActionServlet`, is responsible for initializing a Struts application's configuration from the Struts configuration file and for receiving all incoming requests to the application. Upon receiving a request, `ActionServlet` transfers its processing to the RequestProcessor class. The `RequestProcessor` class processes all aspects of the request, including selecting the Form Bean associated with the request, populating the Form Bean with data, validating the Form Bean, and then selecting the correct Action class to execute for the request.

**In-Class Question:**

After you finish explaining Struts Controller Component, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.

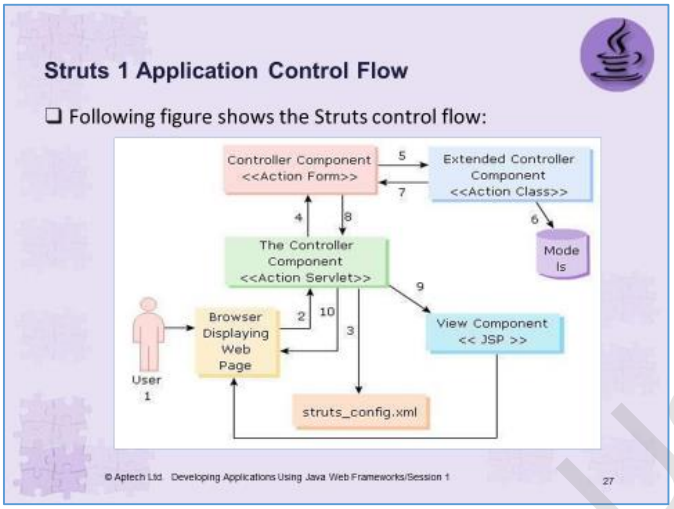
Which Controller Servlet is responsible for initializing a Struts application's configuration from the Struts configuration file and for receiving all incoming requests to the application?

**Answer:**

`ActionServlet`

## Slide 27

Let us understand Struts 1 application control flow.



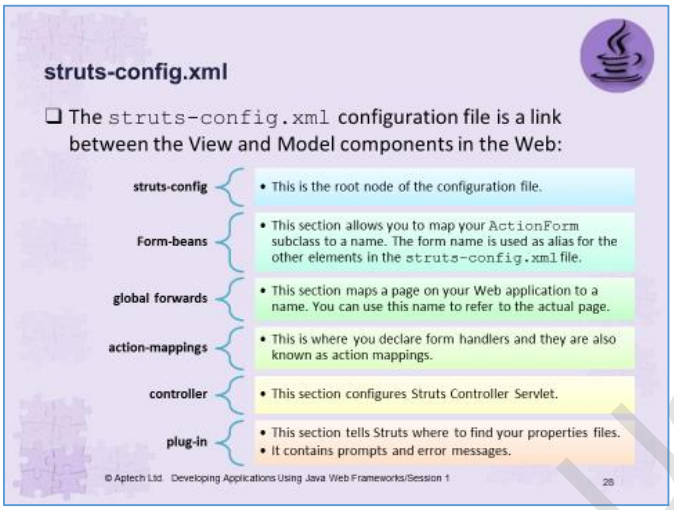

Use slide 27 to explain Struts 1 application control flow.

The control flow `ActionServlet` is as follows:

- Initially, the user sends a request to the Controller Servlet through a View.
- The Controller Servlet looks up the requested URI in the XML configuration file and determines the name of the `Action` class that will process the business logic.
- The `Action` class acts on the Model component as per the application's logic.
- On completion of processing the request, the Action class returns the results to the `ActionServlet` class.
- Based on the results provided, class decides which View to be forwarded with these results.
- The selected View displays the result thus, completing the request–response cycle.

## Slide 28

Let us understand `struts-config.xml`.




Use slide 28 to explain `struts-config.xml`.

The `struts-config.xml` configuration file is a link between the View and Model components in the Web application. The Struts configuration file is an XML-based file. The format of Struts configuration file is based on a Document Type Definition (DTD) file that specifies how the configuration tags must be ordered in the file, what settings are required, and so on.

The configuration file basically contains three main elements:

- `<form-beans>`
- `<global-forwards>`
- `<action-mappings>`

**Form-beans**: Contains form bean definitions. The Form beans create `ActionForm` instances at runtime. The details of each form bean are provided in the `<form-bean>` element. The `<form-property>` elements within the form bean contain the property names and the property types declared in the form bean.

Attributes of `<form-bean>` are as follows:

`name`: Must have a unique form bean name. The name specified is used as a reference in the action configuration.

`type`: The value is a fully-qualified classname of the `ActionForm` class used with the form bean.

**Global forwards**: Contains the global forward definitions. The forward name is the logical name used to map to a specific JSP. The `<forward>` element contains the logical name and the name of the corresponding resource which it maps to.

Attributes of `<forward>` are as follows:

- `name`: Contains the logical name of the forward. This name is used in the `ActionForm` class's `execute()` method to forward it to the next resource.
- `path`: Contains the context relative path of the resource to which the forward name maps.
- `contextRelative`: This takes the value true or false implying, whether the resource specified in the path attribute is relative to the context.

**Action-mappings**: Contains the action definitions. Each action mapping is defined in an `<action>` element. The `<forward>` definition within `<action>`, maps the result of the action to the JSP page invoked. If there is no forward defined in the global forwards, it is specified in the `<action-mapping>` element.

Attributes of `<action-mappings>` are as follows:

- `path`: Contains the name of the request received.
- `type`: Contains the name of the action class invoked.
- `scope`: Specifies the scope in which the Controller (`ActionServlet`) must look for the bean.

Attributes of `<forward>` within `<action-mappings>` are the same as that of the global `<forward>`.

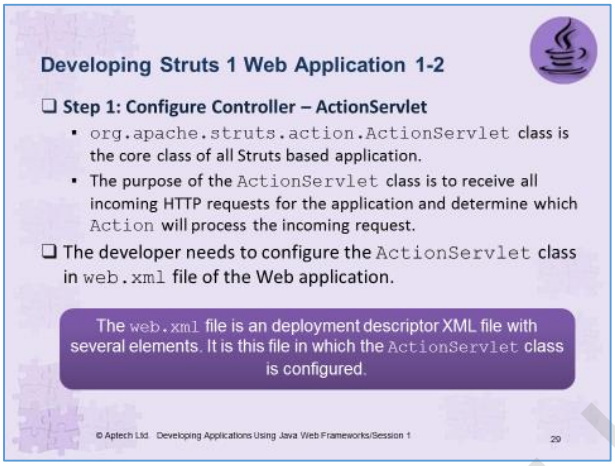
**Controller**: This section configures Struts controller servlet.

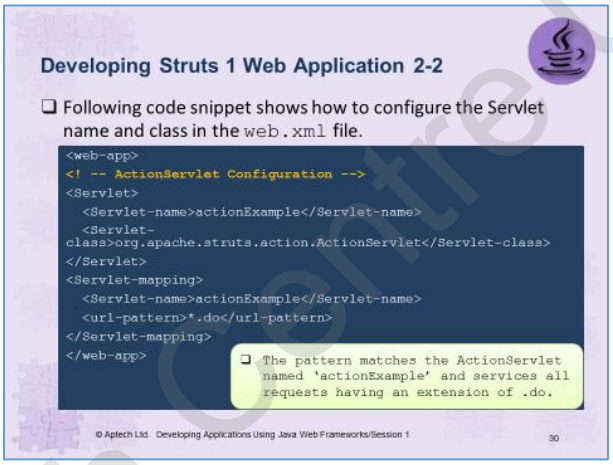
The following figure shows the `struts-config.xml` file:

```
<struts-config>

   <!-- ========== Form Bean Definitions ============ -->
   <form-beans>
      <form-bean name="login" type="test.struts.LoginForm" />
   </form-beans>

   <!-- ========== Global Forward Definitions ========= -->
   <global-forwards>
   </global-forwards>

   <!-- ========== Action Mapping Definitions ======== -->
   <action-mappings>
      <action
         path="/login"
         type="test.struts.LoginAction" >

         <forward name="valid" path="/jsp/MainMenu.jsp" />
         <forward name="invalid" path="/jsp/LoginView.jsp" />
      </action>
   </action-mappings>

   <!-- ========== Controller Definitions ======== -->
   <controller
      contentType="text/html;charset=UTF-8"
      debug="3"
      maxFileSize="1.618M"
      locale="true"
      nocache="true"/>

</struts-config>
```

## Slides 29 and 30

Let us understand developing Struts 1 Web application.




Use slides 29 and 30 to explain developing Struts 1 Web application. The `org.apache.struts.action.ActionServlet` class is the core class of all Struts based application. In Struts based Web applications, the purpose of the `ActionServlet` class is to receive all incoming HTTP requests for the application and determine which Action class will process the incoming request.

The `ActionServlet` class is not an abstract class and can be used for assembling simple application. However, a specialized `ActionServlet` class may be created for specific application. `ActionServlet` class is derived from `javax.Servlet.http.HttpServlet` class and implements the life cycle methods of the Servlet class such as `doGet()`, `doPost()`, and so on.

`ActionServlet` class is responsible for initializing the Struts framework for a Web application and receiving all the requests. It is a concrete class and may be used as it is in an application or can be extended. Every Struts based Web application will have a deployment descriptor named web.xml. A `web.xml` file is an XML file with several elements. It is this file in which the `ActionServlet` class is configured.

Slide 30 shows how to configure servlet name and class in the `web.xml` file.

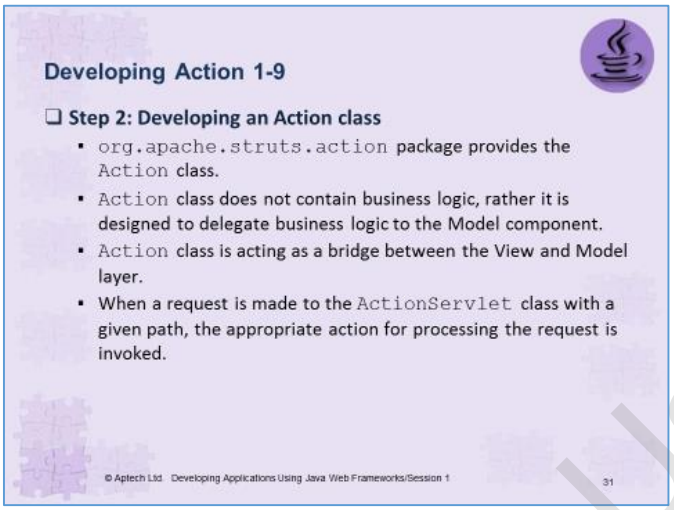
```
<web-app>
< ! -- ActionServlet Configuration -- >
<Servlet>
<Servlet-name>actionExample</Servlet-name>
<Servlet-
class>org.apache.struts.action.ActionServlet</Servlet-class>
</Servlet>
<Servlet-mapping>
<Servlet-name> actionExample</Servlet-name>
<url-pattern>*.do</url-pattern>
</Servlet-mapping>
</web-app>
```

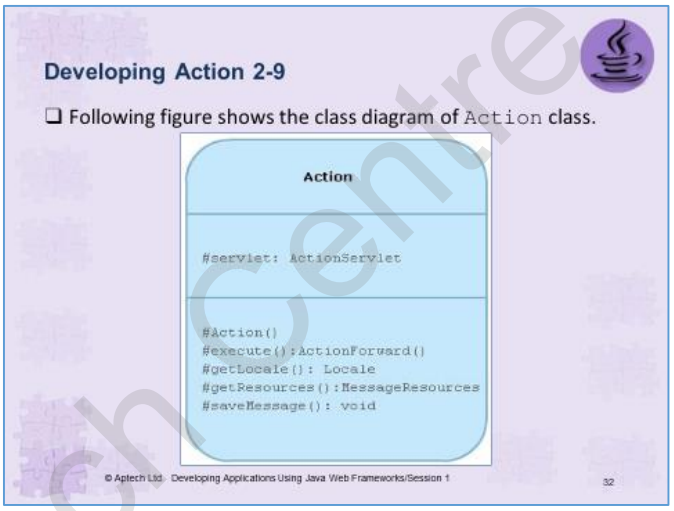
Tell them that the given code shows the configuration of a servlet class with the URL pattern. It shows that the Servlet name is configured as '**actionExample**' and Servlet class is the default ActionServlet class which is the controller class of the Struts framework.

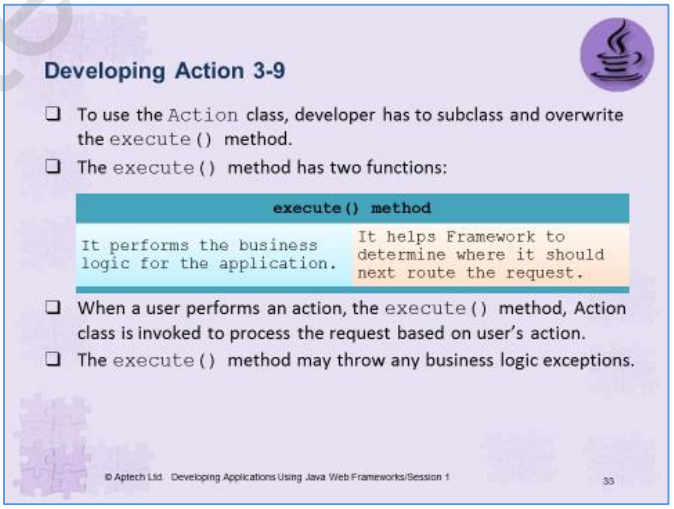
Further, in the `<Servlet-mapping>` description, the tag `<url-pattern>` describes a pattern to resolve URLs. A portion of HTTP request is compared to the `<url-pattern>`. If the pattern matches, then this means that the **ActionServlet** named '**actionExample**' should service all the requests having an extension of `.do`.

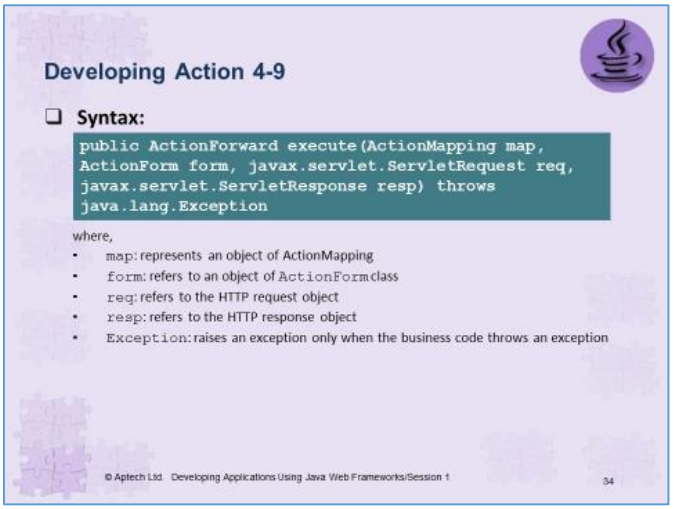
## Slides 31 to 39

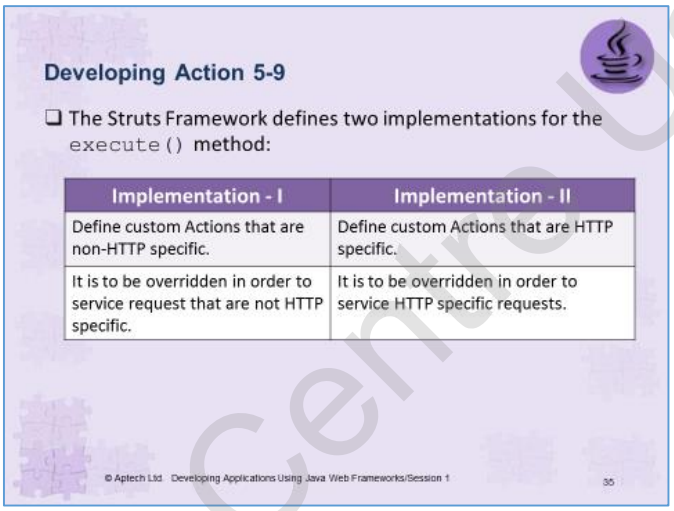
Let us understand developing `Action` class.

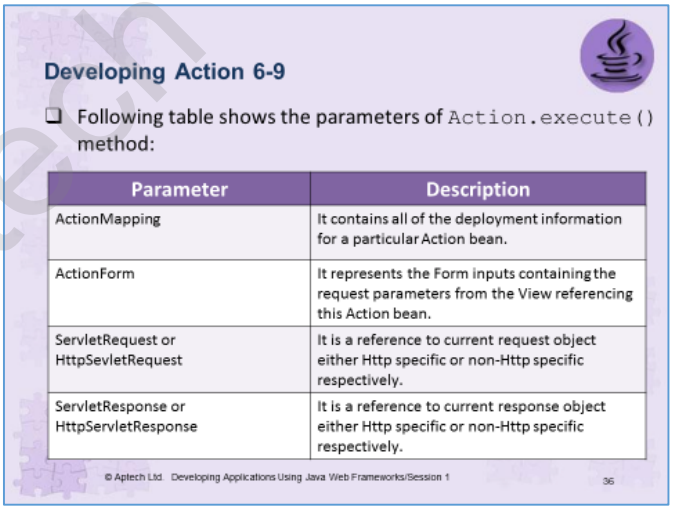


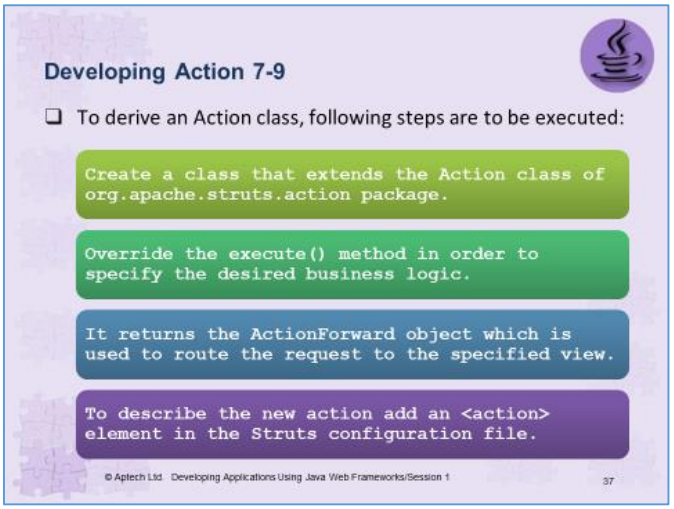



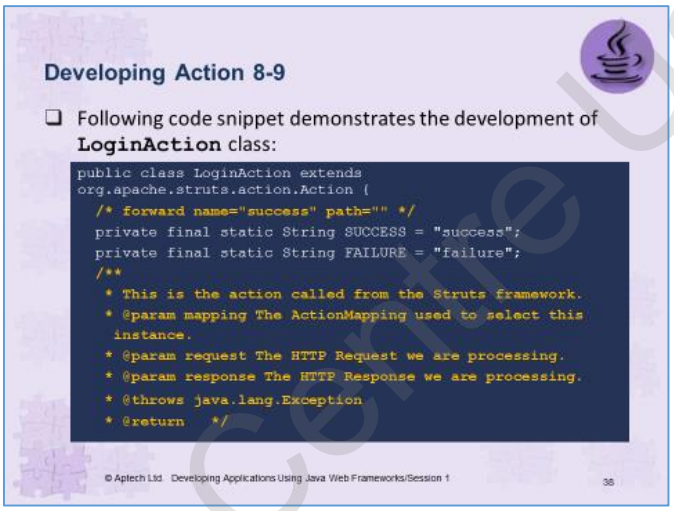



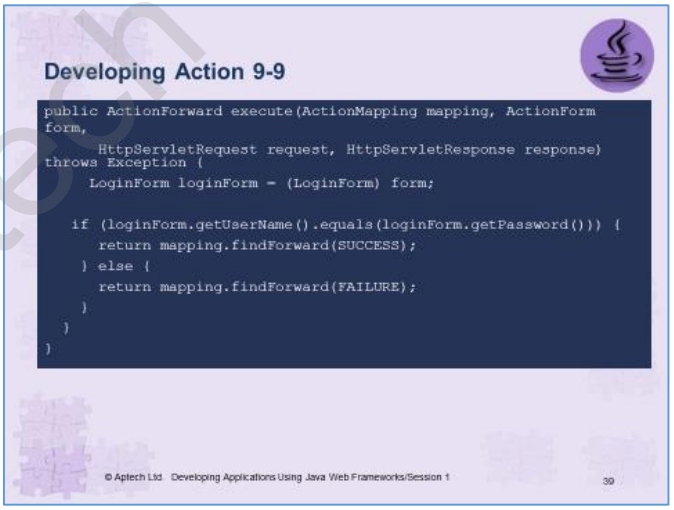





Use slides 31 to 39 to explain developing Action class. Tell the students that Struts framework ends and application code begins in the Action class. The `Action` class does not contain business logic, rather it is designed to delegate business logic to the Model component. It transfers the data from the view layer to the specific business process layer and then, returns the processed data from business layer to the view layer.

Every action is mapped to a path in the Struts configuration file. When a request is made to the `ActionServlet` class with a given path, the action for processing the request is invoked.

Then, explain the figure given on slide 32, defining the methods of Action class. Tell them that the execution of the business logic begins with the `execute()` method. It is similar to the `service()` method provided in the life cycle methods of the servlet.

The `execute()` method has two functions:

1. It performs the business logic for the application.
2. It helps framework to determine where it should next route the request.

The `ActionServlet` class uses the `execute()` method to pass the parameterized class to the `ActionForm` class. The `execute()` method returns an object of type `ActionForward` class. Based on the value returned by `ActionForward` class, the RequestProcessor class determines where to forward the request such as a JSP or another `Action`. If the response has already been completed, then the method simply returns null.

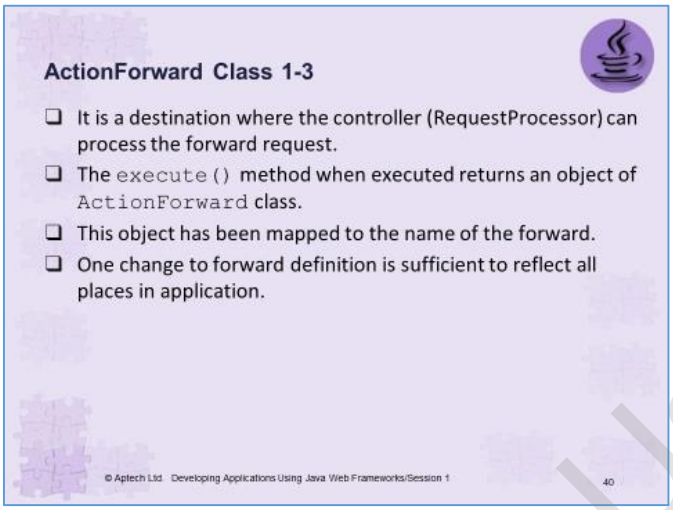
Then, explain the syntax of the `execute()` method given on slide 34. The following table explains the parameters of the `execute()` method:

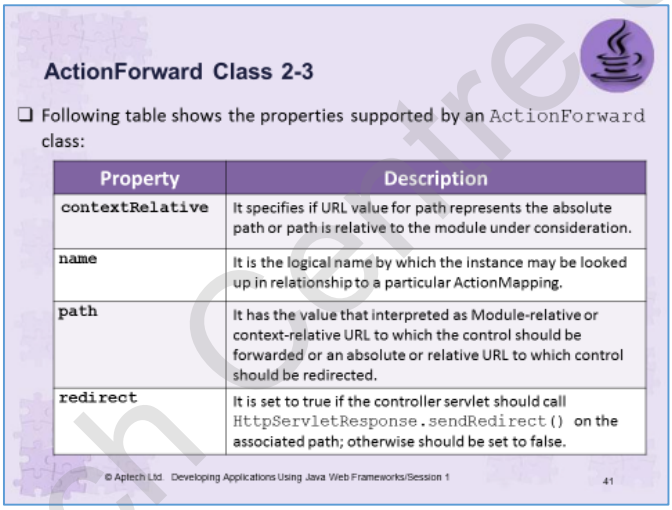
| Parameter | Description |
|---|---|
| `ActionMapping` | It contains all of the deployment information for a particular `Action` bean. |
| `ActionForm` | It represents the Form inputs containing the request parameters from the View referencing this `Action` bean. |
| `ServletRequest or HttpServ-letRequest` | It is a reference to current request object either Http specific or non-Http specific respectively. |
| `ServletResponse or HttpServletResponse` | It is a reference to current response object either Http specific or non-Http specific respectively. |

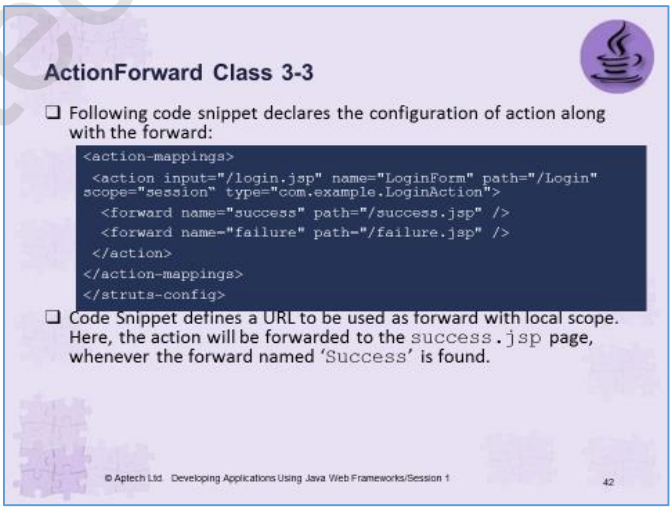
Then, tell them that Struts framework provides two implementations of the `execute()` method. The first `execute()` implementation is used to define custom Actions that are non-HTTP specific. This version is to be overridden in order to service request that are not HTTP specific. The second `execute()` implementation is used to define custom Actions that are HTTP specific. This version is to be overridden in order to service HTTP specific requests. Finally, explain the code snippet given on slides 38 and 39 that demonstrates the `LoginAction` class.

## Slides 40 to 42

Let us understand `ActionForward` Class.



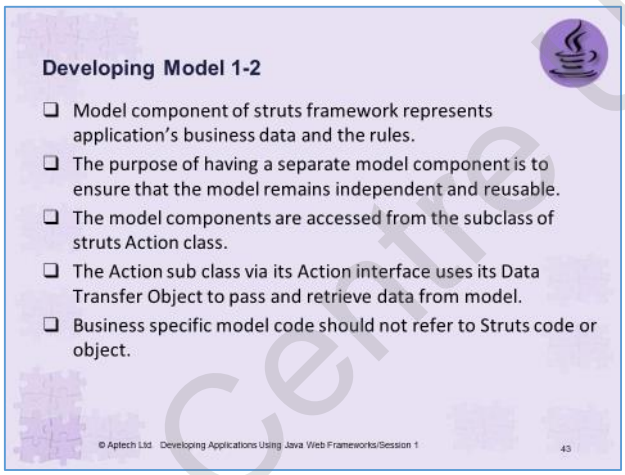

Use slides 40 to 42 to explain `ActionForward` class.

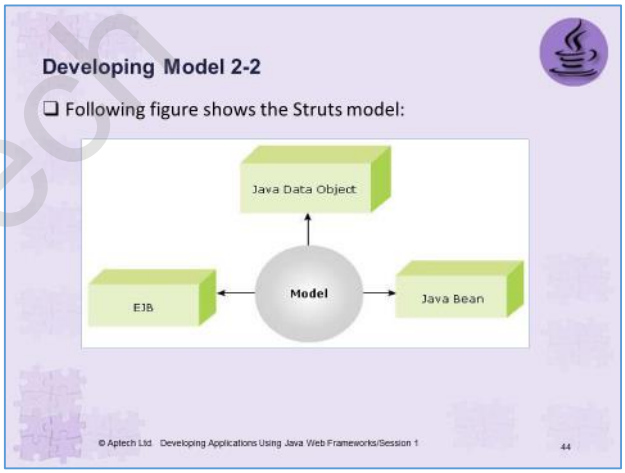
The `execute()` method of the Action class when executed, returns an object of `ActionForward` class. This object has been mapped to the name of the forward from Struts configuration file. Rather than hard-coding URLs inside the application, a developer can use forward to define logical names for URLs and then, use these logical names to reference the URLs. Referencing URLs by logical names gives the benefit of shielding.

Using slide 41, explain the properties of an `ActionForward` class. Then, using slide 42, explain the code snippet that shows configuration of action along with the forward to the result pages.

### Slides 43 and 44

Let us understand developing Model.

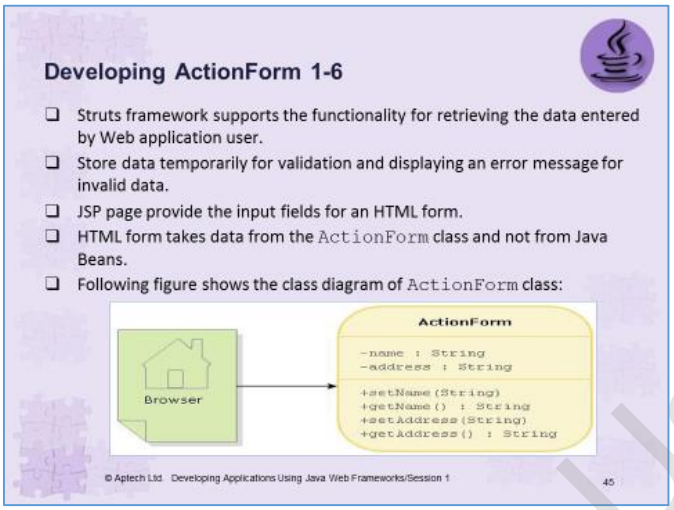


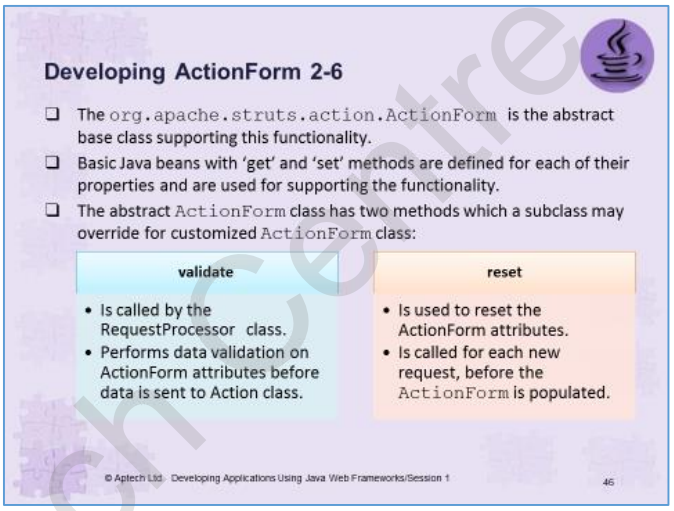



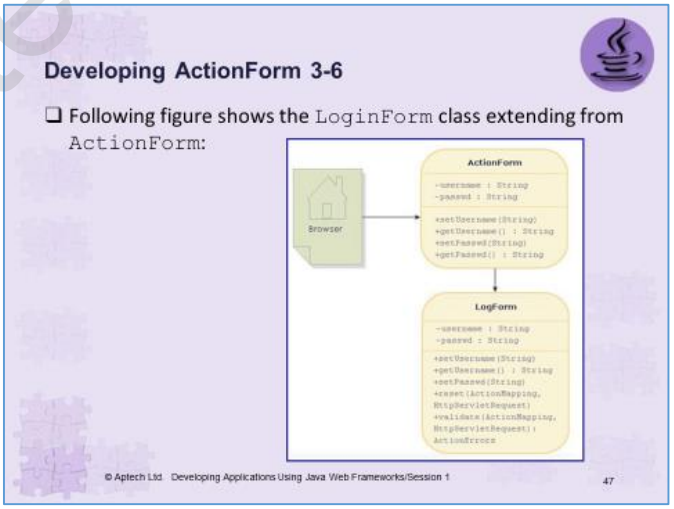

Use slides 43 and 44 to explain developing Model. The purpose of having a separate Model component is to ensure that the model remains independent of the client that is accessing it and thus is reusable. It is very essential for any application to ensure data integrity within the model. In a Struts-based application, the Model components are accessed from the subclass of Struts Action class that is part of the Struts Controller layer. Then, show the figure given on slide 44 that shows different Struts model.

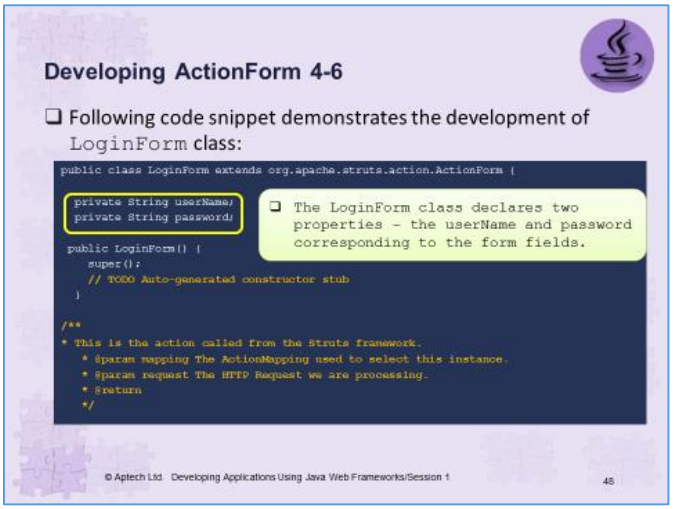
## Slides 45 to 50

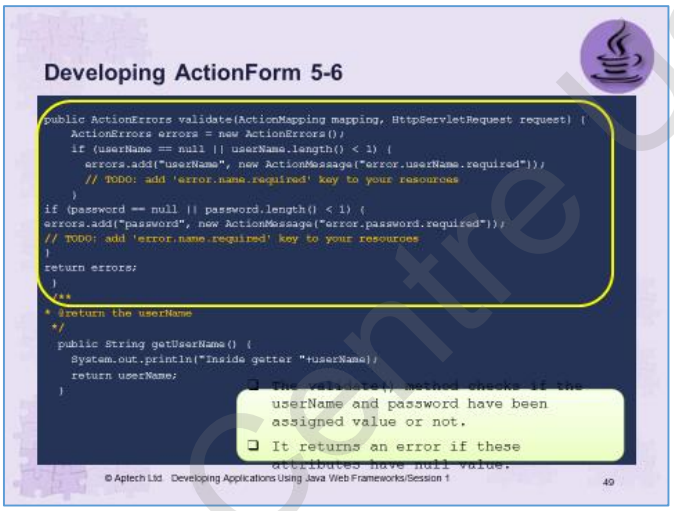
Let us understand developing `ActionForm`.

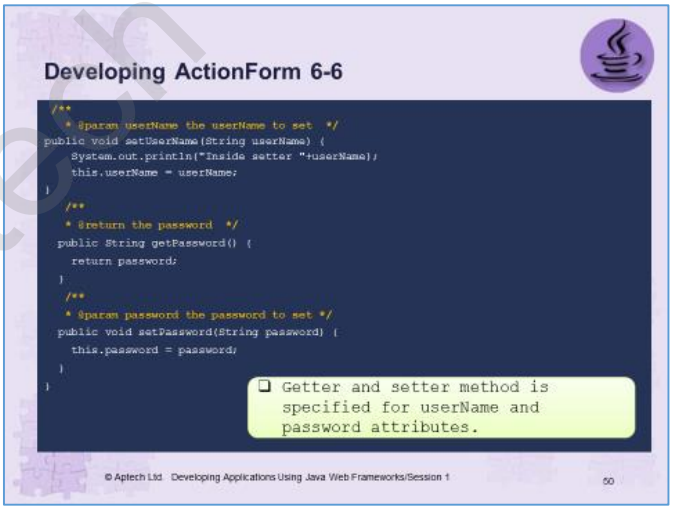





Use slides 45 to 50 to explain developing `ActionForm`.

Struts framework supports the functionality for retrieving the data entered by Web application user (that is from View layer), storing it temporarily for validation and displaying

an error messages for invalid data or sending valid data to Action class (that is to Controller layer).

Also, when the model layer returns data for display, the `ActionForm` class is populated. This is then used by JSP page to provide the input fields for an HTML form. Then, tell them that a subclass of `org.apache.struts.action.ActionForm` is created to capture the application specific form data as well as to display the required data on the presentation page.

To override the `ActionForm` class, you need to override two methods that are as follows:

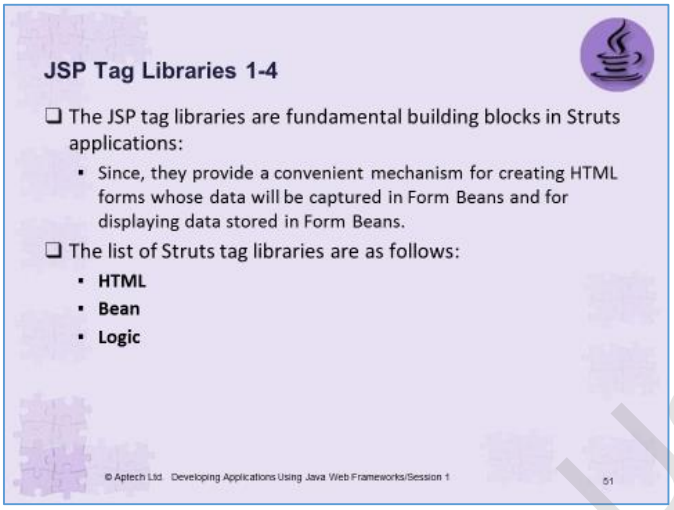
- The `reset()` method is used to reset the `ActionForm` attributes to whatever state the application wants.

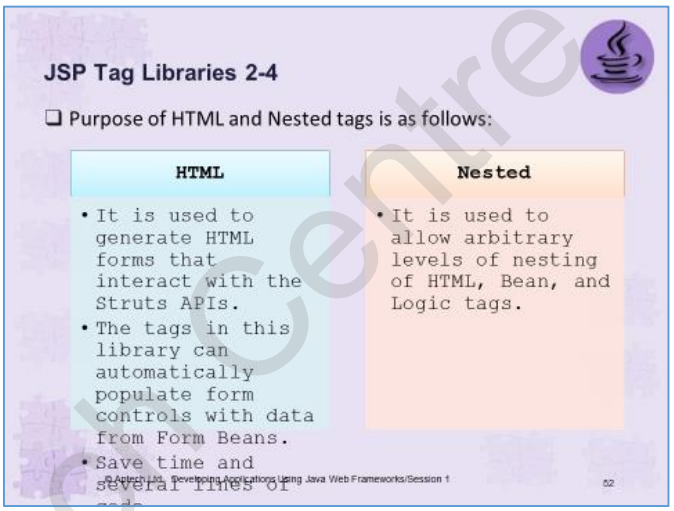
- The `validate()` method for performing data validation on `ActionForm` attributes before data is sent to `Action` class.

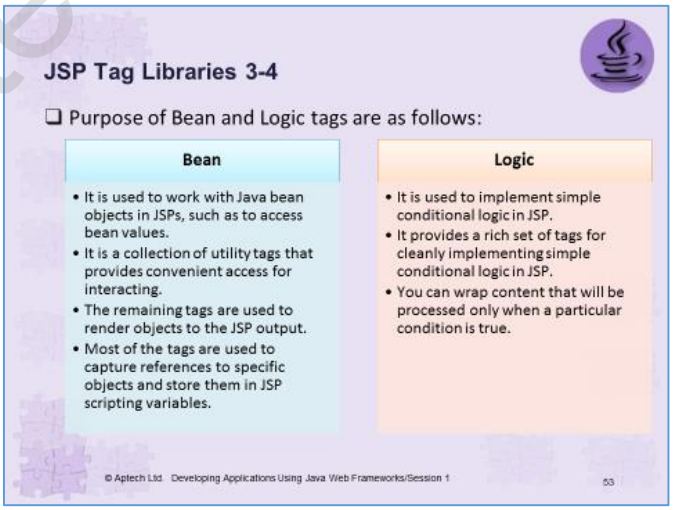
Using slides 48 to 50, explain the code snippet that demonstrates the `LoginForm` class. Tell them that the code creates a `LoginForm` class which is extended from `ActionForm` class. The `LoginForm` class declares two properties - the userName and password corresponding to the form fields. A getter and setter method is specified for both the attributes. Next, the `validate()` method checks if the userName and password have been assigned value or not. It returns an error if these attributes have null value.

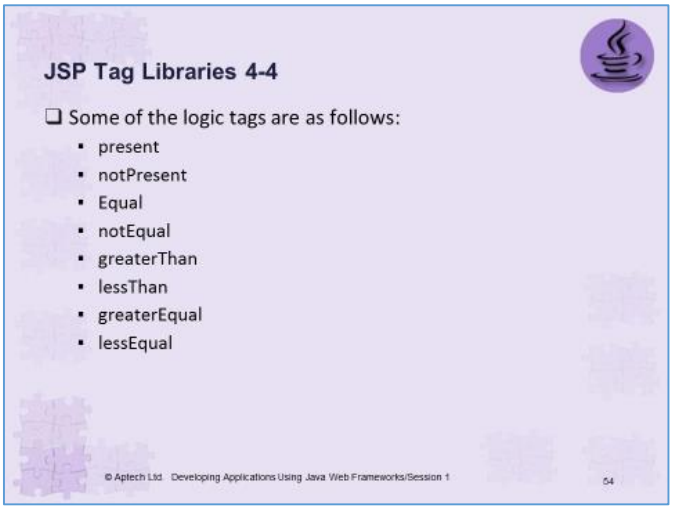
## Slides 51 to 54

Let us understand JSP tag libraries.






Use slides 51 to 54 to explain JSP tag libraries. The JavaServer Pages Standard Tag Library (JSTL) contains core functionality that provides common functionalities for many JSP applications. JSTL helps the user to employ a single, standard set of tags. This standardization helps to deploy the user's applications on any JSP container supporting JSTL.

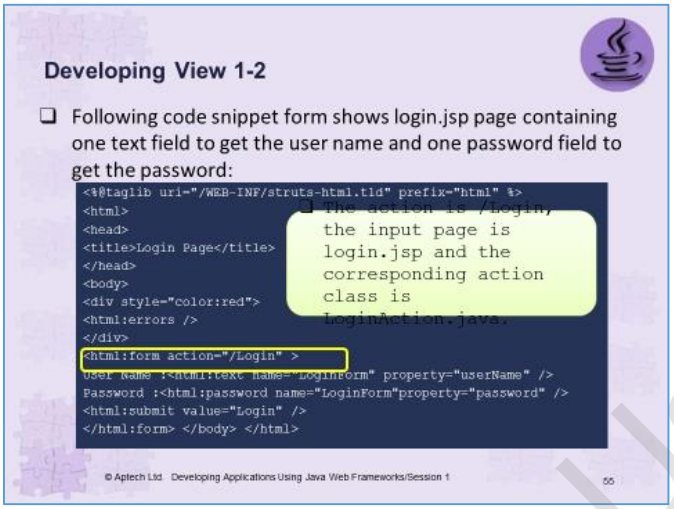
JSTL has tags such as iterators and conditionals for handling flow control, tags for manipulating XML documents, internationalization tags, tags for accessing databases using SQL, and commonly used functions.

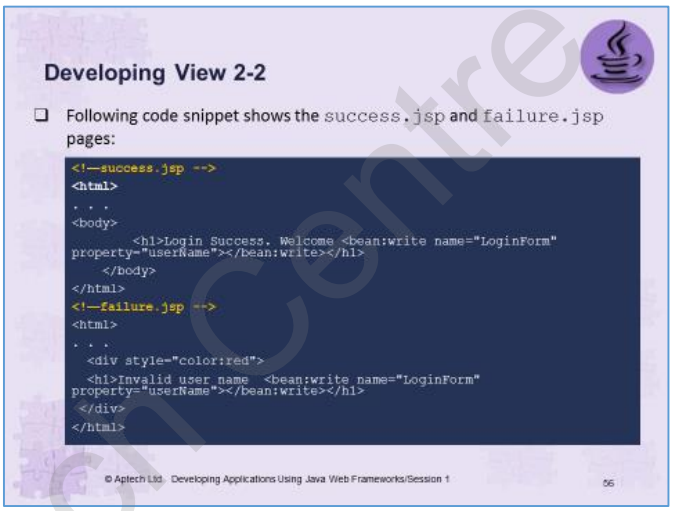
Using slide 52, explain the purpose of HTML and nested tags.

Using slide 53, explain the purpose of Bean and Logic tags. Also, mention the list of the various Logic tags.

## Slides 55 and 56

Let us understand developing View.



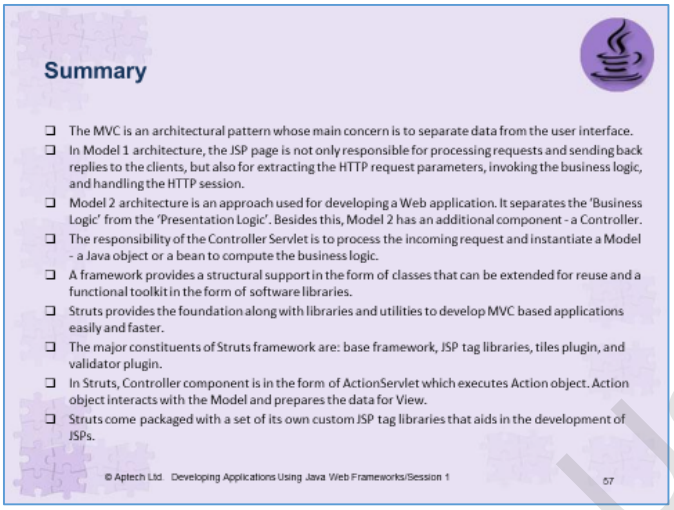




Use slides 55 and 56 to explain how to develop View. The `login.jsp` page has one text field to get the user name and one password field to get the password. The form also has one `Submit` button, which when clicked calls the login action. `<html:errors/>` tag is used to display the error messages to the user.

Using slide 56, show the creation of `success.jsp` and `failure.jsp` pages.

## Slide 57

Let us summarize the session.




In slide 57, you will summarize the session. You will end the session, with a brief summary of what has been taught in the session.

### 1.3 Post Class Activities for Faculty

You should familiarize yourself with the topics of the next session. You should also explore with Struts 2 framework that is the next version of Struts 1 framework.

**Tips:**

You can also check the Articles/Blogs/Expert Videos uploaded on the OnlineVarsity site to gain additional information related to the topics covered in the next session. You can also connect to online tutors on the OnlineVarsity site to ask queries related to the sessions.

# Session 2 – Working with Struts 2 Framework

## 2.1 Pre-Class Activities

Before you commence the session, you should familiarize yourself with the topics of this session in-depth. You should revisit topics of the previous session for a brief review.

Here, you can ask students the key topics they can recall from previous session. Prepare a question or two which will be a key point to relate the current session objectives.

### 2.1.1 Objectives

By the end of this session, the learners will be able to:

- Explain the features of Struts 2 framework
- Explain the architecture of Struts 2 framework
- Explain the components and request life cycle of Struts 2 framework
- Explain the difference between Struts 1 and Struts 2 framework
- Explain how to implement Struts 2 actions using Action interface and ActionSupport class
- Explain the process of managing data from forms to JavaBean properties
- Explain how to configure Struts 2 components in the configuration file
- Explain Result and Result Types
- Explain the annotations provided by Struts 2 framework
- Explain how to develop a Java Web application using Struts 2 framework

### 2.1.2 Teaching Skills

To teach this session successfully, you should be aware of the features of Struts 2 framework and its architecture. Familiarize yourself with the components and request life cycle of Struts 2 framework and the implementation of Struts 2 actions using `Action` interface and `ActionSupport` class.

Aware yourself with process of managing data from forms to JavaBean properties and how to configure Struts 2 components in the configuration file. You should know how to configure Result and Result Types for the actions. Finally, aware yourself with annotations for configuring actions and develop a Java Web application using Struts 2 framework.

For teaching in the class, you are expected to use slides and LCD projectors.

**Tips:**

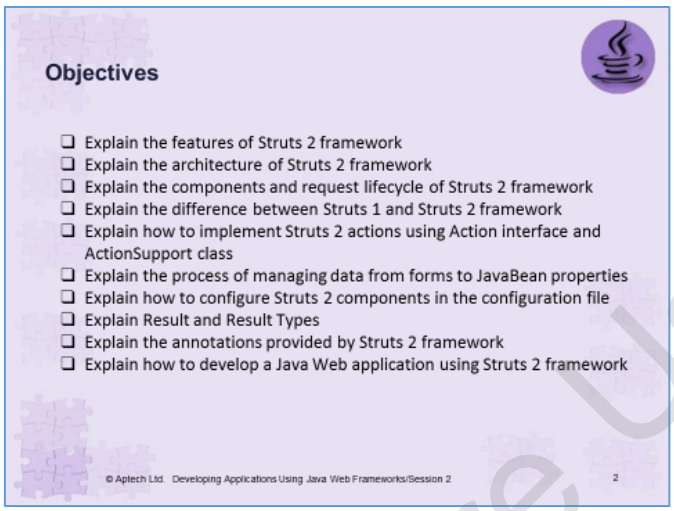
It is recommended that you test the understanding of the students by asking questions in between the class.

**In-Class Activities:**

Follow the order given here during In-Class activities.

**Overview of the Session:**

Give the students a brief overview of the current session in the form of session objectives. Show the students slide 2 of the presentation.



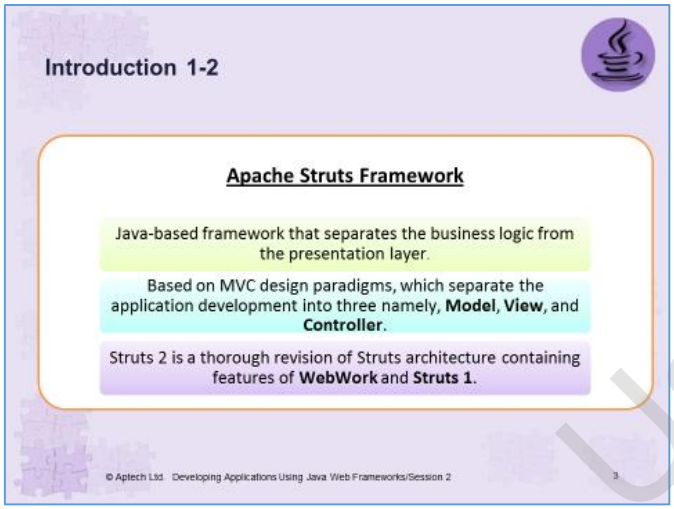

Tell the students that they will be introduced to the features of Struts 2 framework and its architecture. Explain the components and request lifecycle of Struts 2 framework and the implementation of Struts 2 actions using `Action` interface and `ActionSupport` class.

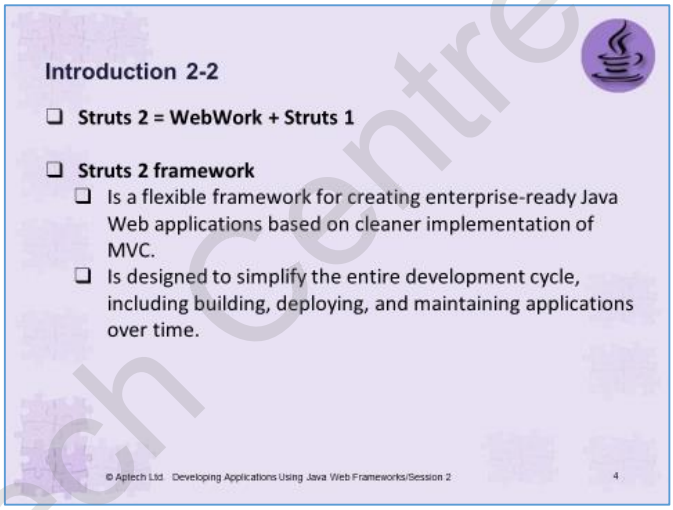
In this session, they will learn the process of managing data from forms to JavaBean properties and how to configure Struts 2 components in the configuration file. Discuss about Result and Result Types supported for actions. Finally, the session discusses about the action annotations and develops a Java Web application using Struts 2 framework.

## 2.2 In-Class Explanations

### Slides 3 and 4

Let us understand Struts 2 framework.




Use slides 3 and 4 to explain Struts 2 framework.

The main limitation of Struts 1 is that it encourages a fix approach to MVC approach when designing Web applications. It provides a standard component template defined for the applications. Thus, resulting in a rigid approach.

The next generation of Apache Struts 2 came with the launching of Struts 2 which made the Web application development easier. It is the second generation of Web application framework based on WebWork framework created by OpenSymphony. After working independently for several years, OpenSymphony joined with Apache to create Struts 2 framework. Thus, Struts 2 is not just the next version of Struts 1, but it is a thorough revision of the Struts architecture containing features of WebWork and Struts 1.

Struts 2 is a flexible framework for creating enterprise-ready Java Web applications based on cleaner implementation of MVC. The framework is designed to simplify the entire development cycle, including building, deploying, and maintaining applications over time.

Tell the students that Struts 2 framework renders supports to POJO-based actions, Validation Support, AJAX Support, Integration support to various frameworks such as Hibernate, Spring, Tiles, and so on. It provides support for various Views such as Velocity, Freemarker, JSP, and so on.

### In-Class Question:

After you finish explaining Struts 2 framework, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.

What are the different technologies used as Views in Struts 2.0?

### Answer:

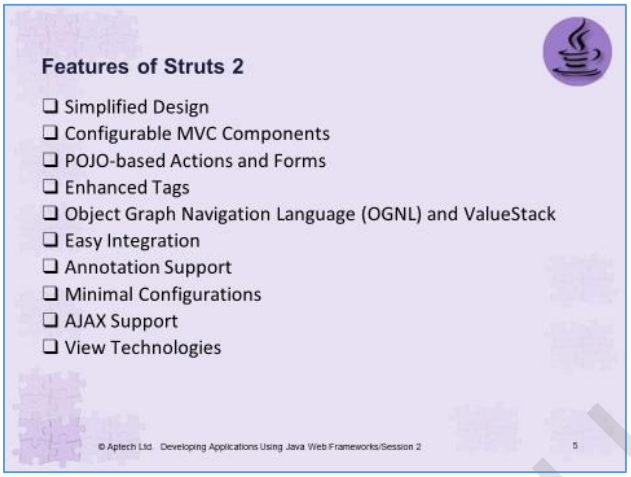
Velocity, Freemarker, and JSP.

### Tips:

**WebWork** was a Java-based Web application framework developed by OpenSymphony that was later on merged into the current Struts 2 framework. It was designed to improve developer productivity and simplify code. WebWork was built on top of XWork, which provided a generic command pattern framework as well as an Inversion of Control container.

## Slide 5

Let us understand the features of Struts 2.




Use slide 5 to explain the features of Struts 2.

Struts 2 is an advanced version of Struts framework. It provides many new features that were not in Struts 1.

Some of the important features of Struts 2 are as follows:

**Simplified Design**

Struts 2 framework classes are based on interfaces and most of its core interfaces are HTTP independent and framework neutral. This enables the users to test Struts application very easily without referring the low-level HTTP Request and HTTP Response objects in the application.

**Configurable MVC Components**

In Struts 2 framework, the component information is configured in struts.xml file. Therefore, any change in the information can be made by simply changing it in the xml file.

**POJO-based Actions and Forms**

Struts 2 Action classes are POJOs which are simple java class. The action class acts as a model in the Web application. They are not required to implement any interface or inherit any class. Therefore, testing of the code is highly simplified. Even, to capture user input from the form, developers can use JavaBeans instead of ActionForms provided in Struts 1.

**Enhanced Tags**

Struts 2 tags enable to add stylesheet-driven markup capabilities. Here, the user can create consistent pages with fewer codes. Markup tags can be changed by changing an underlying

stylesheet in Struts 2. In addition, it is very easy for the developer to extend the tags and write one's own functionality. Struts 2 provide various types of tags such as Data tags, UI tags, Control tags, and so on which simplify the application development.

**Object Graph Navigation Language (OGNL) and ValueStack**

Struts 2 uses an elegant expression language called Object Graph Navigation Language (OGNL) which is more powerful and flexible than Java Standard Tag Library (JSTL) expression language used by Struts 1 framework. OGNL supports Asynchronous JavaScript and XML (AJAX) implementation in User Interface (UI). OGNL allows the developer to refer and manipulate the data present on the ValueStack. ValueStack is simply a storage area that holds application data associated with the processing of a request.

**Easy Integration**

Struts 2 is an extensible framework can be easily integrated with any other frameworks. A framework must provide an abstraction layer for integrating with other frameworks. The frameworks that can be integrated with the Struts 2 applications include Hibernate, Spring, Tiles, and so on.

**Theme and Template Support**

Struts 2 provides three types of themes: xhtml, simple, and css_xhtml. The default theme of struts 2 is xhtml. These themes and templates can be used for common look and feel across the application. Using these themes and templates, a user can change the overall look of a Website, simply by changing the theme files.

**Annotation Support**

Struts 2 uses annotations. The advantage of using annotations is that it eliminates the manual task of configuring the components of the application in the XML configuration file. Annotations allow the developers to add meta-data information into the Java source code declaring Actions. This meta-data information is translated into components during runtime.

**Minimal Configurations**

For the Struts 2 application, minimal configuration is required as most of the settings will take the default values. Configuration is required only if there are any modifications in the default settings.

**AJAX Support**

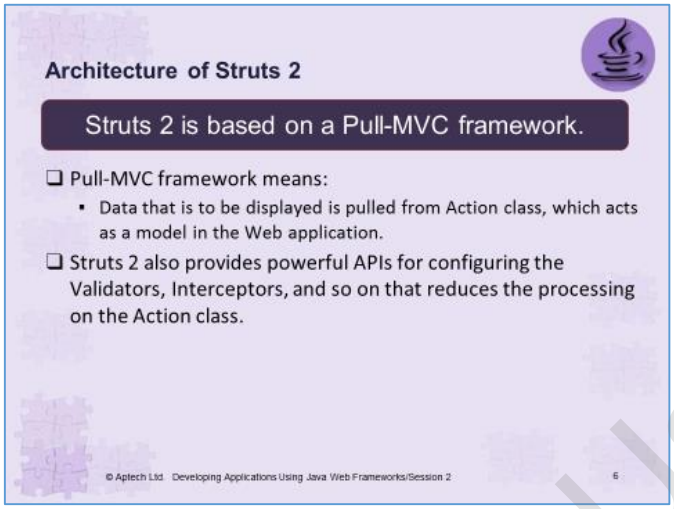
Struts 2 integrates AJAX support into the framework by creating AJAX tags.

**View Technologies**

Struts 2 provide a great support for presenting the results on the Web pages. The multiple view technologies include JSP, FreeMarker, Velocity, and XSLT.

## Slide 6

Let us understand the architecture of Struts 2.

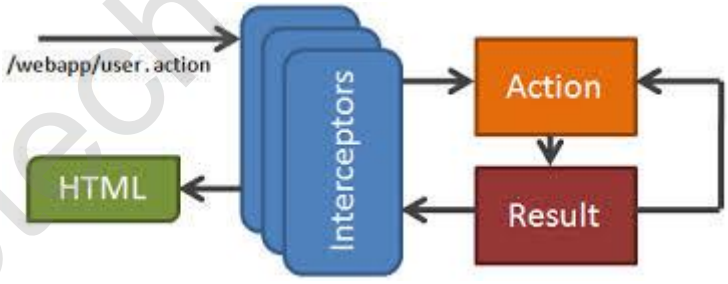



Use slide 6 to explain the architecture of Struts 2.

Struts 2 is based on a pull-MVC framework. It means that the data that is to be displayed is pulled from Action class, which acts as a model in the Web application.

However, Struts 1 was based on action-based MVC framework. In action-based framework, a Servlet act as a controller and provides a centralized point of control for client page requests.

The following figure shows the high-level architecture of Struts 2.0 framework:




The steps for user's request life cycle in Struts 2 is as follows:

1. User sends a request to the server, requesting for some resource. The request is send to the servlet container on the Struts configured Web server.

2. The servlet filter looks at the request and then determines the appropriate `Action`.

3. Configured interceptors functionalities are applied before and after the execution of the action class.

4. Then, the selected action is executed to perform the requested operation.

5. The result is prepared by the view and returns the result to the user.

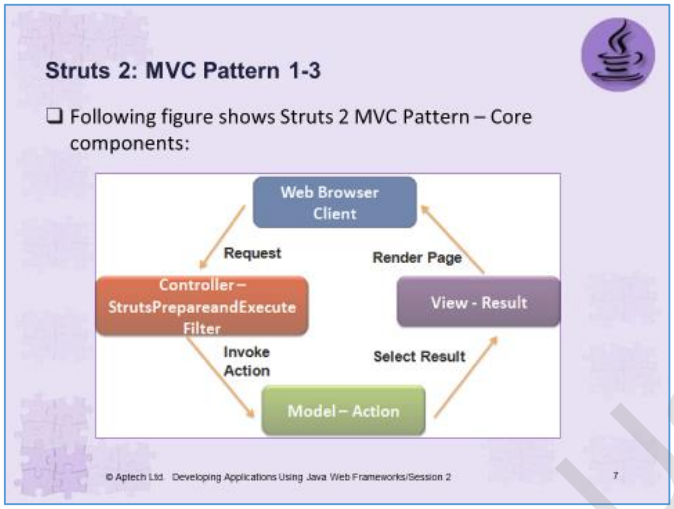
**Tips:**

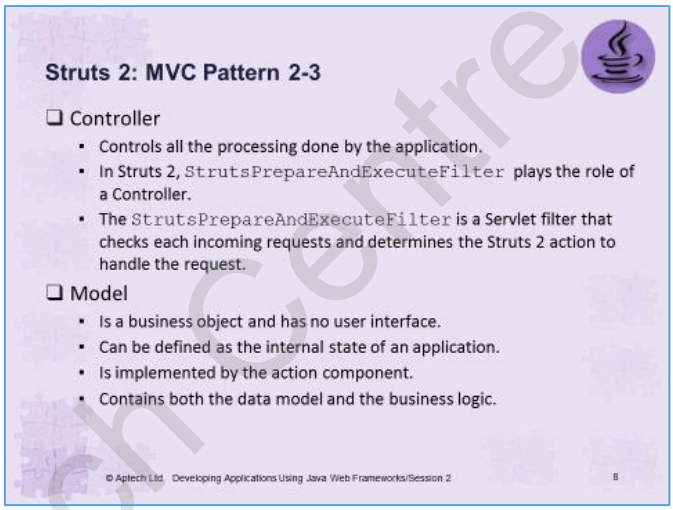
Normally, frameworks are divided into two types that are as follows:

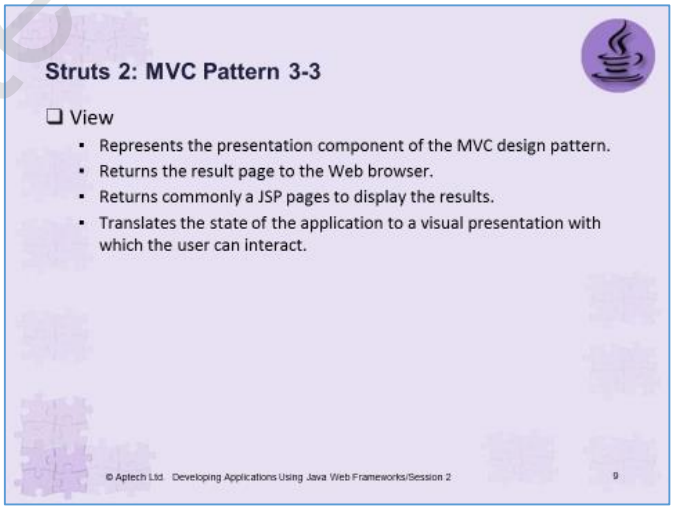
- Invasive - It will force the programmers to create their classes by extending or implementing from per-defined classes or interfaces provided by that framework. Struts 1.0 is the type of invasive framework.
- Non-Invasive - It does not forces the programmer to extend or implement its own classes or interfaces. Struts 2.0 is the type of non-invasive framework.

## Slides 7 to 9

Let us understand Struts 2: MVC Pattern.



Use slides 7 to 9 to explain Struts 2: MVC Pattern.

Struts 2 follows the MVC design pattern which helps to define the coding pattern or style. The components defined for MVC architecture of Struts 2.0 framework are as follows:

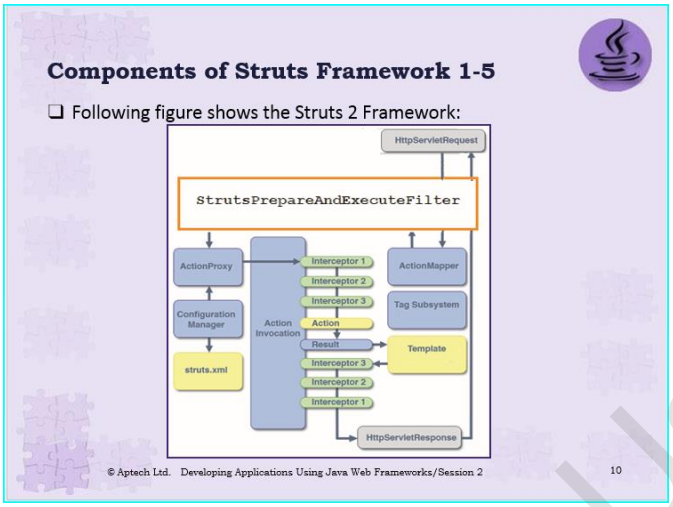
- **Controller**: The main task of the Controller is to map the incoming request to appropriate set of actions. Controller is the first component that will process the HTTP request coming from the client. In Struts 2, `StrutsPrepareAndExecuteFilter` plays the role of a Controller. It is a Servlet filter that checks each incoming requests and determines the Struts 2 action to handle the request. It is important to note that earlier version of Struts framework that is Struts 1.0 provided `FilterDispatcher` as a Controller.

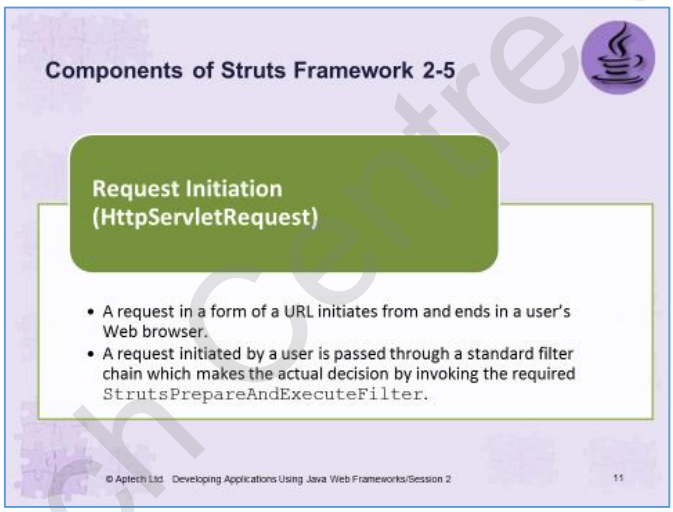
- **Model**: The model in MVC design pattern in Struts 2 framework is implemented by the action component. The model is the core of an application, as it contains both the data model and the business logic. A Web application can have a number of actions mapping to various requests. Thus, after receiving a request the Controller checks the mapping to determine the correct action that will handle the request. Once the correct action is found, the Controller invokes the action and hands over control of the request processing to the action. Once the action class completes the request processing, it forwards the result to the Struts 2 View component.

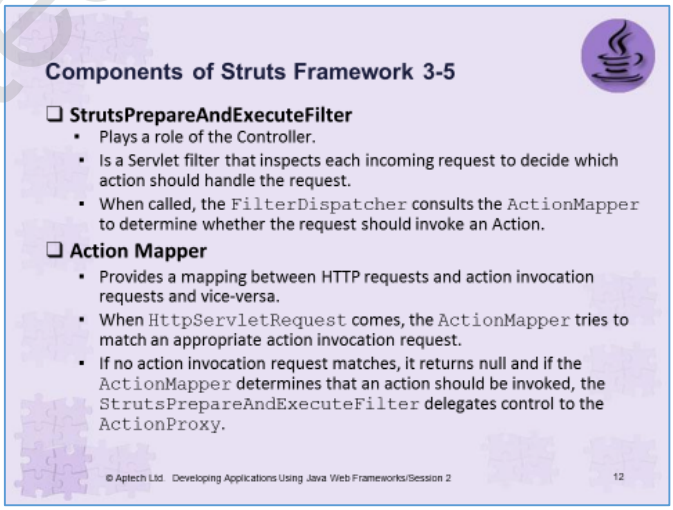
- **View**: The JSP page is used to commonly display the results. Usually, it is the outcome of the action processing such as success or error.

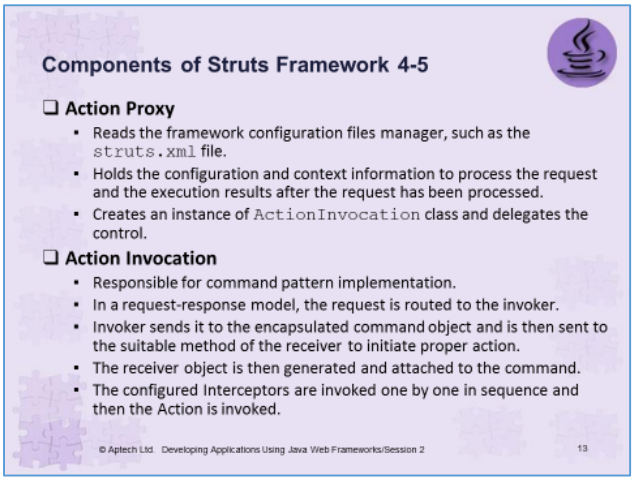
## Slides 10 to 14

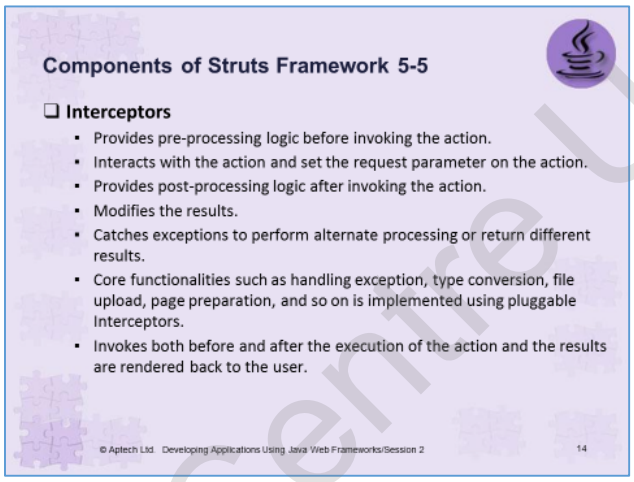
Let us understand the components of Struts framework.






Use slides 10 to 14 to explain the components of Struts framework.

Using the figure as shown on slide 10, explain the steps that will come under the life cycle of a Struts 2 based application.

1. Client sends the request to the Web or servlet container available in the Web server such as Tomcat. The Web container hands over the request to the application requested by the client.
2. The application passes the request to the front controller (`FilterDispatcher` or `StrutsPrepareAndExecuteFilter`) based on configured `web.xml` file. Tell them, from Struts 2.1 onwards, `org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter` plays a role of the Controller. This important object is a Servlet filter that inspects each incoming request to decide which action should handle the request.

3. The `StrutsPrepareAndExecuteFilter` gets the Action information to be executed from `ActionMapper` class. It is an interface which provides a mapping between HTTP requests and action invocation requests and vice-versa.
4. The Controller calls the `ActionProxy`. The `ActionProxy` obtains the information of action and interceptor stack based on `struts.xml` and `struts-default.xml` file. The `struts-default.xml` is part of the struts 2 core jar file.
5. The `ActionProxy` forwards the request to the `ActionInvocation`.
6. The `ActionInvocation` is responsible to execute all the interceptors and respective action and result is generated and sent back to the `ActionInvocation`.
7. The `ActionInvocation` hands over the response to `HttpServletResponse` object with the help of controller and result is sent to the client.
8. The client reads the values from `ValueStack` using expression language and display output on the screen.

Then, discuss on the use of Interceptors that allow the developers to develop the code specification that can be executed before or after the execution of an action.
Using interceptors, the developer can perform the following:

- Provide pre-processing logic before invoking the action.
- Interact with the action and set the request parameter on the action.
- Provide post-processing logic after invoking the action.
- Modify the results.
- Catch exceptions to perform alternate processing or return different results.

The configured Interceptors are invoked one by one in sequence and then the Action is invoked. Once the Action returns, it looks for a proper result associated with the Action result code mapped in the `struts.xml` file.

**In-Class Question:**

After you finish explaining components of Struts framework, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.

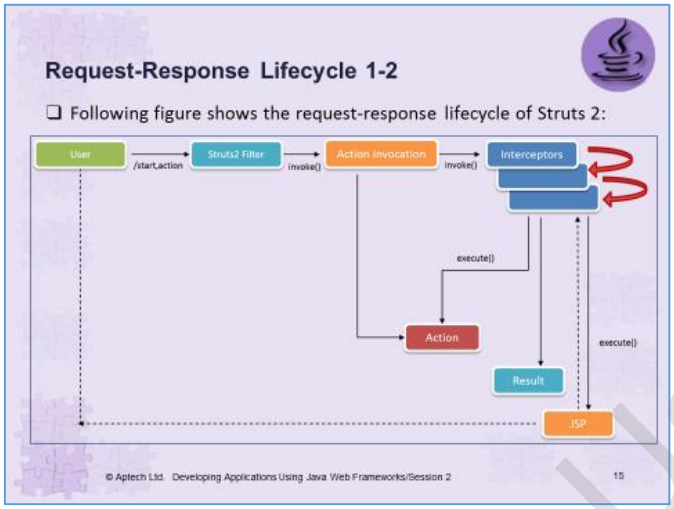
Which component maps HTTP request to the action invocation request?

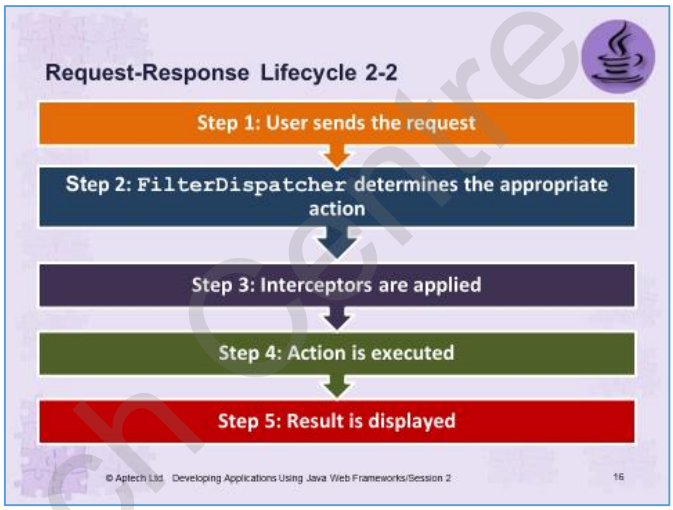
**Answer:**

Action Mapper

## Slides 15 and 16

Let us understand the Request-Response life cycle.




Use slides 15 and 16 to explain the Request-Response life cycle.

The following step shows Request-Response life cycle in Struts 2:

**Step 1: User sends the request**

The request-response cycle starts and ends from the user's Web browser. A request is in the form of an URL. This request URL represents an Action.

The request processing life cycle of Struts 2 follows the given order:

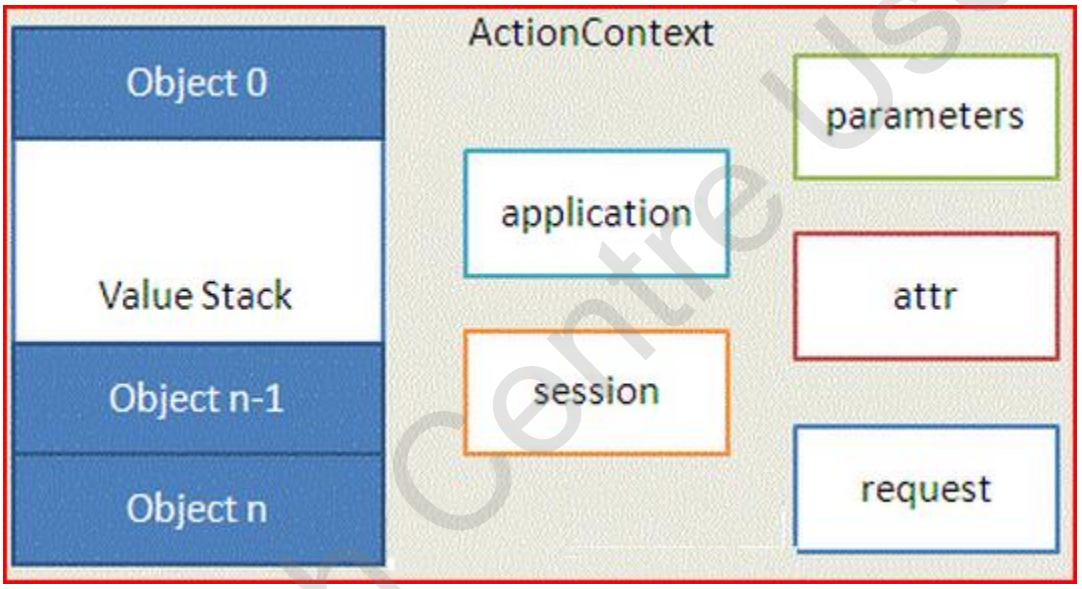
- Displays static content through user interface provided by the View.
- Determines the action configuration, that is, determines which action maps to the URL from the incoming request.
- Creates the action context as it converts the request to a protocol independent format for the actions to use.

- Creates the action proxy class containing the configuration and context information for processing of the request. It also contains the execution results after the requests have been processed.
- Performs cleanup of the `ActionContext` object and ensures no memory leak.

**ActionContext**

ActionContext is a global storage area that holds all the data associated with the processing of a request. In Struts 2, the action resides on the ValueStack which is a part of the ActionContext. ActionContext is thread local which means that the values stored in the ActionContext are unique per thread, this makes the Struts 2 actions as thread safe.

The following figure depicts the ActionContext:




**Step 2: StrutsPrepareAndExecuteFilter determines the appropriate action**

The `StrutsPrepareAndExecuteFilter` accepts the request and then consult `ActionMapper` to determine the suitable Action. If `ActionMapper` discovers an action to be invoked, then `StrutsPrepareAndExecuteFilter` delegates the control to `ActionProxy`.

**ActionProxy**

The `ActionProxy` class stores the configuration and context information for request processing and execution results after the request has been processed. `ActionProxy` class then creates an instance of the `ActionInvocation` class and delegates the control. The `ActionInvocation` object manages the execution environment and contains the conversational state of the action being processed. This class is the core of the `ActionProxy` class. Actions, interceptors, and results are the three environmental components.

**Step 3: Interceptors are applied**

The developer is required to write code around each and every action. The `ActionInvocation` class creates an instance of the action class. Struts 2 create a new instance for each and every request that is received. This instance behaves as POJO. The interceptors help to specify the 'request processing life cycle' for an action.

**Step 4: Action is executed**

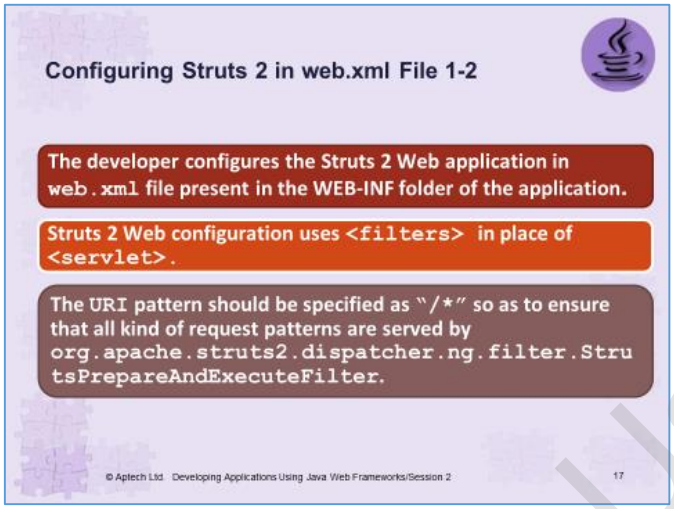
The action method is executed to carry out the database related operations like storing or retrieving data from the database. After processing of the request by the `execute()` method, a String result is returned which is mapped to an implementation of the `Result` interface.

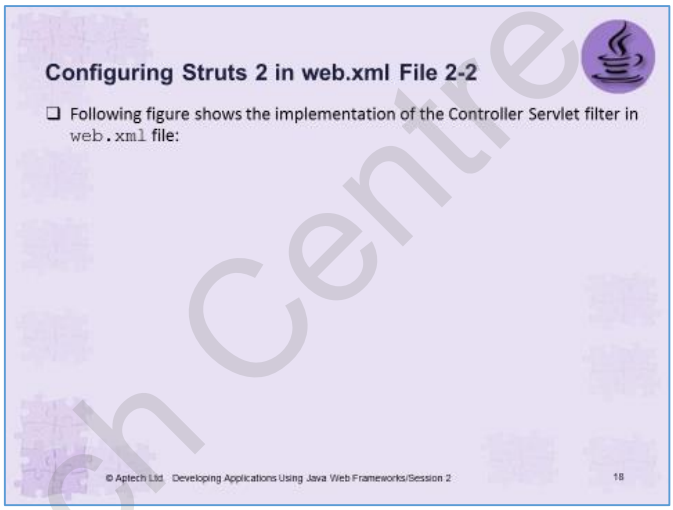
**Step 5: Result is displayed**

The result is sent to the Servlet container which sends the output to the user browser.

## Slides 17 and 18

Let us understand configuring Struts 2 in web.xml File.




Use slides 17 and 18 to explain configuring Struts 2 in `web.xml` file.

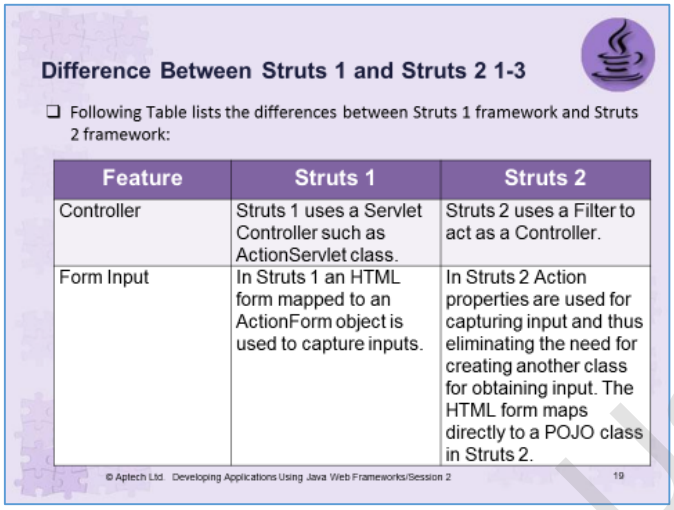
Tell the students that the `web.xml` configuration file is an XML configuration file that determines how elements of the HTTP request are processed by the servlet container. The `web.xml` file renders an entry point for any Web application. The entry point of Struts 2 application will be a filter defined in deployment descriptor (`web.xml`). Struts 2 Web configuration uses <`filters`> in place of <`Servlet`>. The URI pattern should be specified as '`/*`' so as to ensure that all kind of request patterns are served by `org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter`.

The following figure shows the implementation of the filter Controller in `web.xml`.

```
<?xml version="1.0" encoding="UTF-8"?>
. . .
<filter>
    <filter-name>struts2</filter-name>
    <filter-class>
org.apache.struts2.dispatcher.ng.filter.StrutsPrepare
                                    AndExecuteFilter
</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>example/index.jsp</welcome-file>
</welcome-file-list>
</web-app>
```

## Slides 19 to 21

Let us understand the difference between Struts 1 and Struts 2.

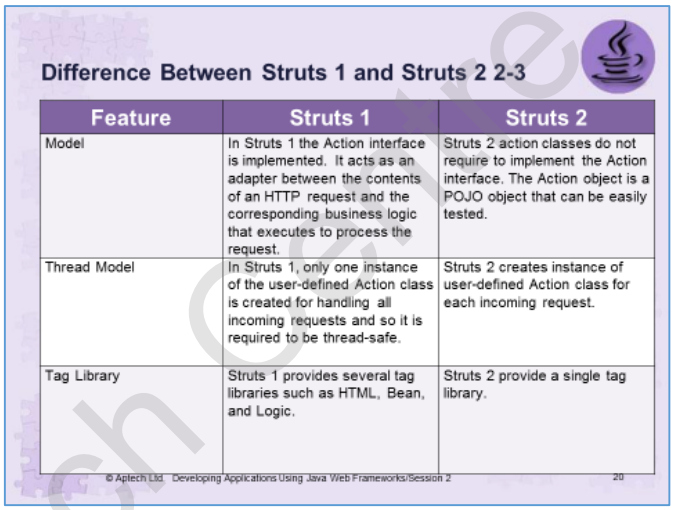


Difference Between Struts 1 and Struts 2 1-3

- Following Table lists the differences between Struts 1 framework and Struts 2 framework:

| Feature | Struts 1 | Struts 2 |
|---|---|---|
| Controller | Struts 1 uses a Servlet Controller such as ActionServlet class. | Struts 2 uses a Filter to act as a Controller. |
| Form Input | In Struts 1 an HTML form mapped to an ActionForm object is used to capture inputs. | In Struts 2 Action properties are used for capturing input and thus eliminating the need for creating another class for obtaining input. The HTML form maps directly to a POJO class in Struts 2. |

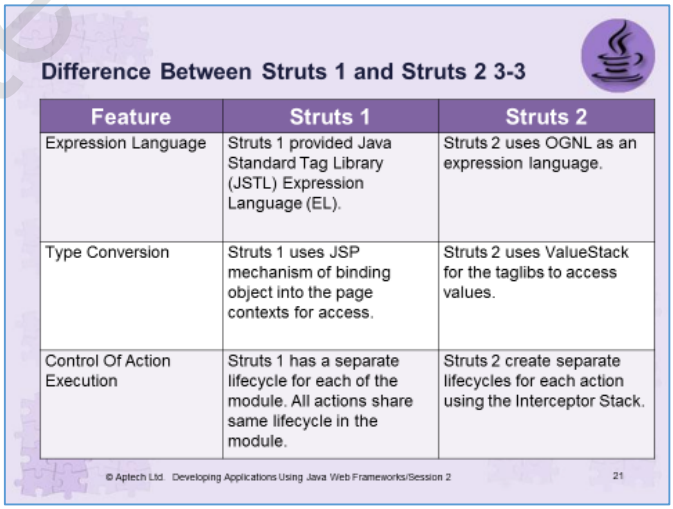




Difference Between Struts 1 and Struts 2 2-3

| Feature | Struts 1 | Struts 2 |
|---|---|---|
| Model | In Struts 1 the Action interface is implemented. It acts as an adapter between the contents of an HTTP request and the corresponding business logic that executes to process the request. | Struts 2 action classes do not require to implement the Action interface. The Action object is a POJO object that can be easily tested. |
| Thread Model | In Struts 1, only one instance of the user-defined Action class is created for handling all incoming requests and so it is required to be thread-safe. | Struts 2 creates instance of user-defined Action class for each incoming request. |
| Tag Library | Struts 1 provides several tag libraries such as HTML, Bean, and Logic. | Struts 2 provide a single tag library. |





Difference Between Struts 1 and Struts 2 3-3

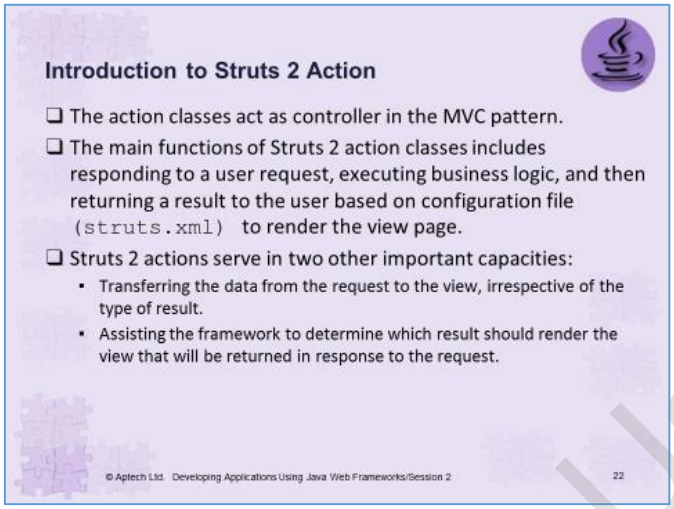
| Feature | Struts 1 | Struts 2 |
|---|---|---|
| Expression Language | Struts 1 provided Java Standard Tag Library (JSTL) Expression Language (EL). | Struts 2 uses OGNL as an expression language. |
| Type Conversion | Struts 1 uses JSP mechanism of binding object into the page contexts for access. | Struts 2 uses ValueStack for the taglibs to access values. |
| Control Of Action Execution | Struts 1 has a separate lifecycle for each of the module. All actions share same lifecycle in the module. | Struts 2 create separate lifecycles for each action using the Interceptor Stack. |



Use slides 19 to 21 to explain the difference between Struts 1 and Struts 2.

## Slide 22

Let us understand Struts 2 Action.




Use slide 22 to explain Struts 2 Action. Tell the students that in struts 2, action class is Plain Old Java Object (POJO). Each URL is mapped to a specific action, which provides the processing logic necessary to service the request from the user.

Action helps to the transfer of data from the request to the view, whether it is a JSP or other type of result. The action helps the framework in determining which result will provide the view that will be returned in the response to the request.

In Struts 2 action class, `execute()` method should be specified that represents the business logic.

**Requirements of Action class**

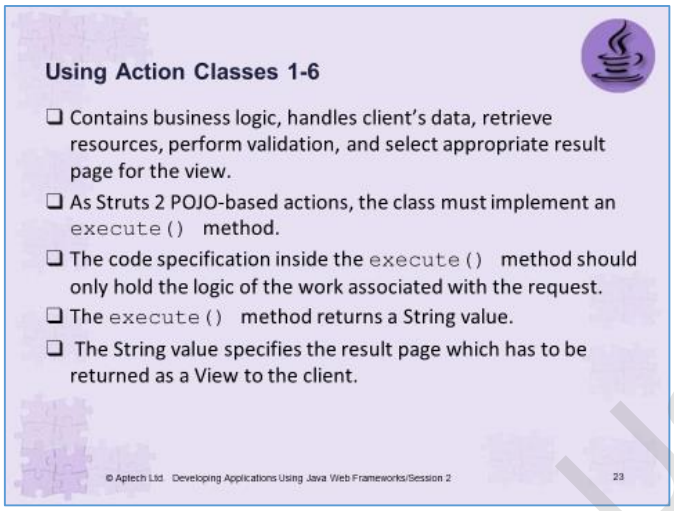
The only requirement for actions in Struts 2 is that there must be one no-argument method that returns either a String or Result object and must be a POJO. If the no-argument method is not specified, the default behavior is to use the `execute()` method.

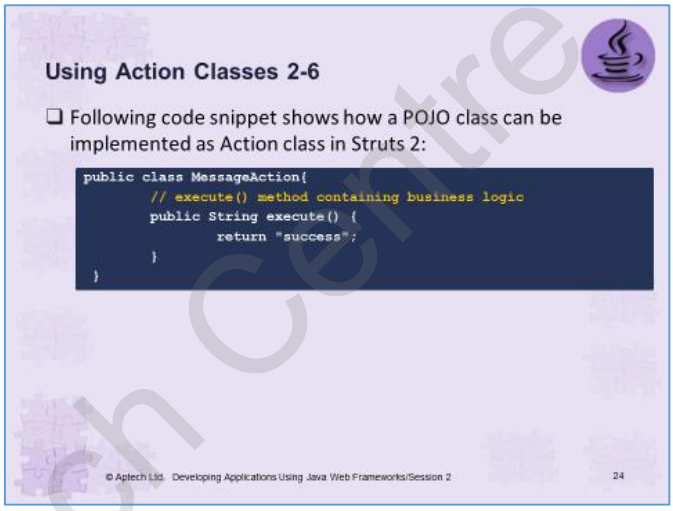
**Difference between Struts 1 and Struts 2 Action class**

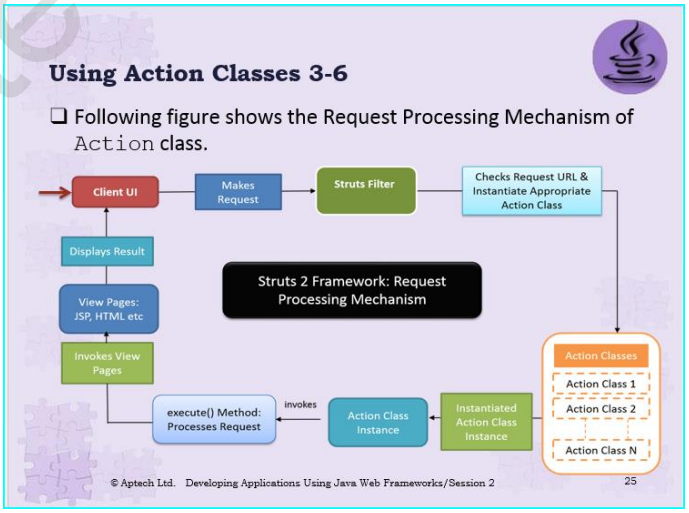
The Action class is a singleton class in struts 1.0 framework. Thus, it is not a thread safe and to make it thread safe, the developer needs to apply synchronization. However, in Struts 2.0, an Action class object will be created for each request, so it is by default thread safe.

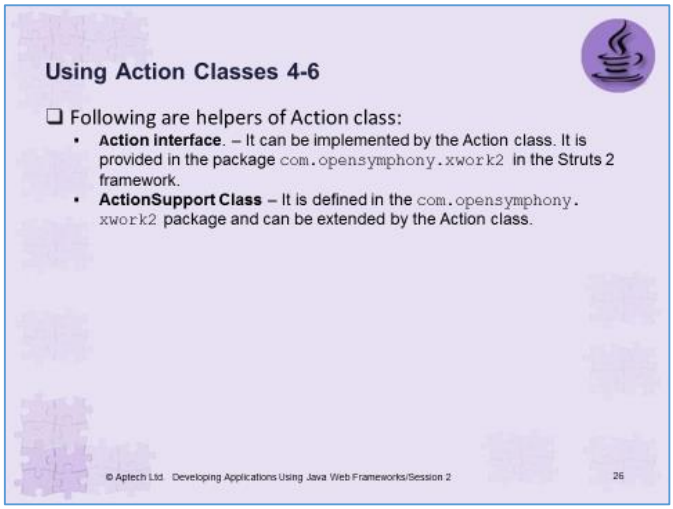
## Slides 23 to 28

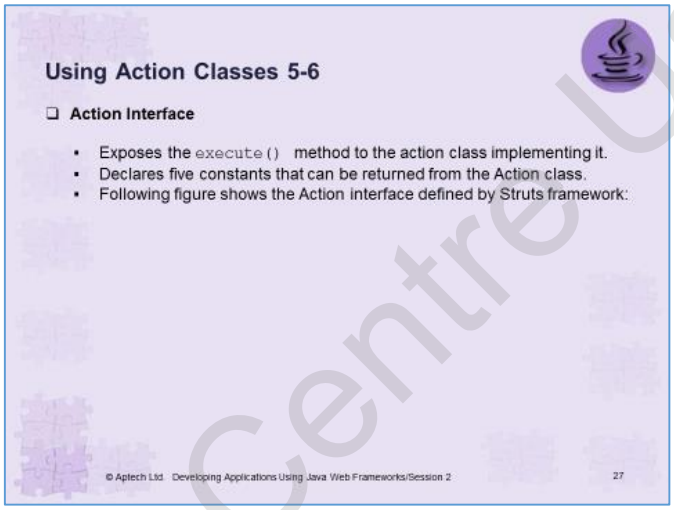
Let us understand using Action classes.

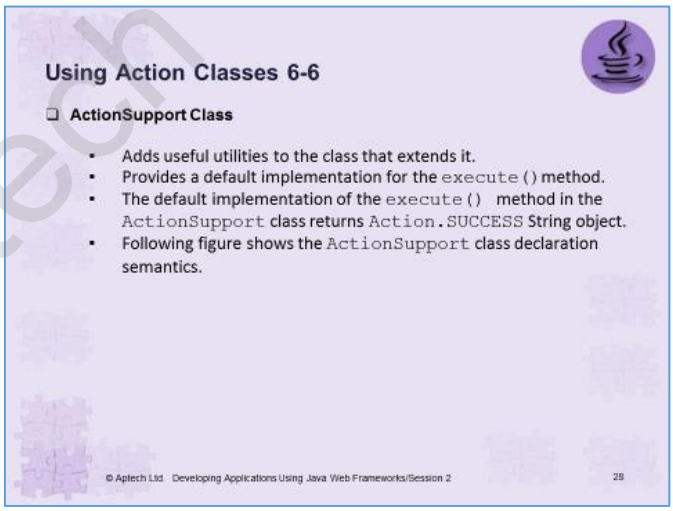





Use slides 23 to 28 to explain how to use action classes.

Tell them that there are two helpers for actions. They can be used:

- By implementing Action interface. The `Action` interface is provided in the package `com.opensymphony.xwork2` in the Struts 2 framework.

- By extending `ActionSupport` class defined in the `com.opensymphony.xwork2` package.

Explain that the `Action` interface has five constants that can be returned from the action class. They are as follows:

1. `SUCCESS` - indicates that action execution is successful and a success result should be shown to the user.
2. `ERROR` - indicates that action execution is failed and an error result should be shown to the user.
3. `LOGIN` - indicates that user is not logged-in and a login result should be shown to the user.
4. `INPUT` - indicates that validation is failed and an input result should be shown to the user again.
5. `NONE` - indicates that action execution is successful but no result should be shown to the user.

Similarly, `Actionsupport` class can be extended by the user-defined action class. It provides default implementation of the `execute()` method. In addition, it implements many interfaces such as `Action`, `Validateable`, `ValidationAware`, `TextProvider`, `LocaleProvider`, and so on. It is mostly used instead of `Action` interface.

The description is as follows:

- `Validateable` and `ValidationAware` interfaces that provide programmatic, annotation-based, and declarative XML-based validation.
- `TextProvider` and `LocaleProvider` interfaces that provide support for localization and internationalization.
- `Serializable` interface that helps to create classes for easy transfer of binary data over a communication channel.

The following code snippet shows the use of `ActionSupport` class:

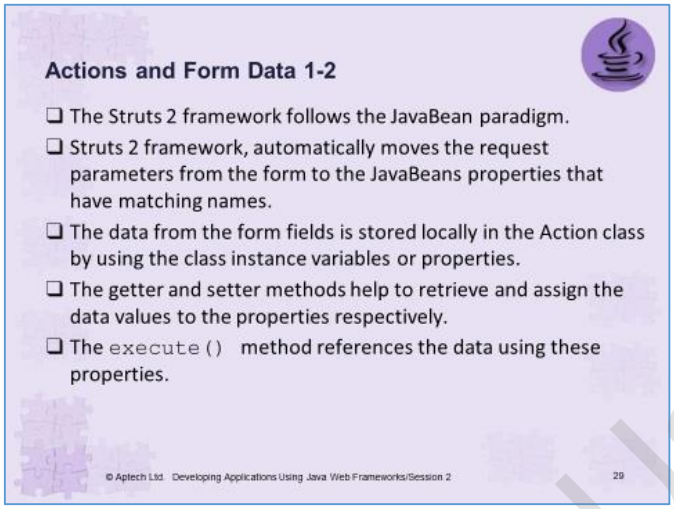
```
public class HelloWorldAction extends ActionSupport {
private String name;


   public String execute() throws Exception {
            if ("john".equals(name))
            {
                return SUCCESS;
```

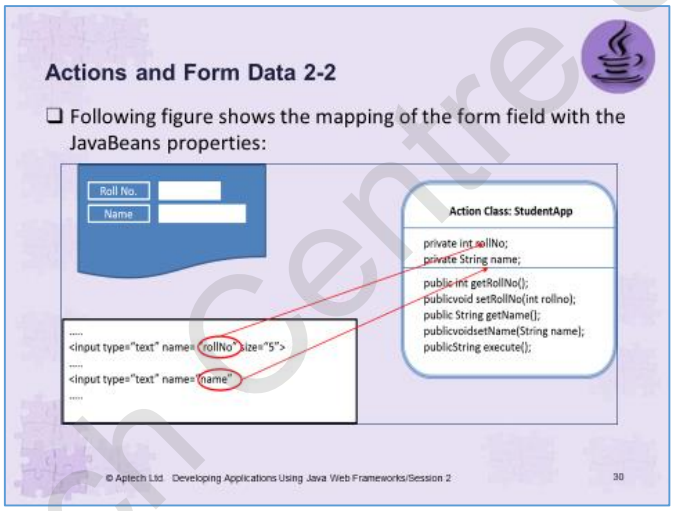
```
            }else{
               return ERROR;
            }
          }
      public String getName() {
       return name;
    }

    public void setName(String name) {
       this.name = name;
    }
}
```

As shown in the given code snippet, the `execute()` method can return String constants such as `SUCCESS` and `ERROR`, as the class is extended from `ActionSupport` class.

## Slides 29 and 30

Let us understand actions and form data.



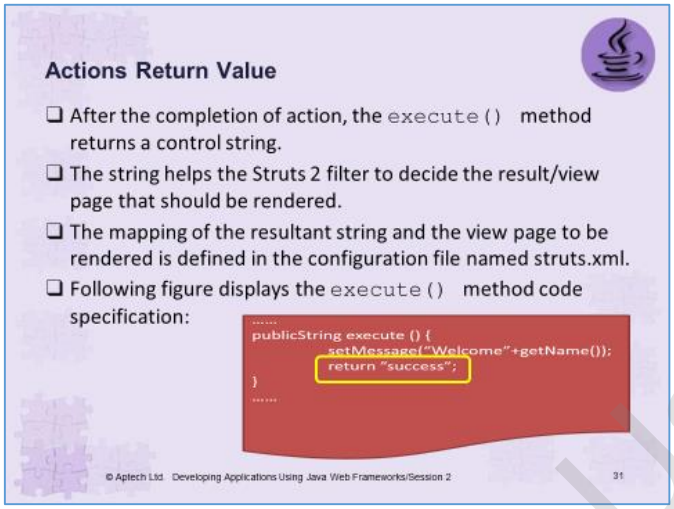

Use slides 29 and 30 to explain actions and form data.

Tell them that the data from the form fields is stored locally in the `Action` class by using the class instance variables or properties. The getter and setter methods help to retrieve and assign the data values to the properties respectively. The `execute()` method references the data using these properties.

Using the figure as shown on slide 30, explain that the `rollNo` and `name` parameters from the form are automatically assigned to the `rollNo` and `name` property of the JavaBean. In Struts 2, each request string or form value is a name-value pair. Struts 2 provides automatic type conversion between string and other data types using OGNL expression language.

## Slide 31

Let us understand Actions return value.

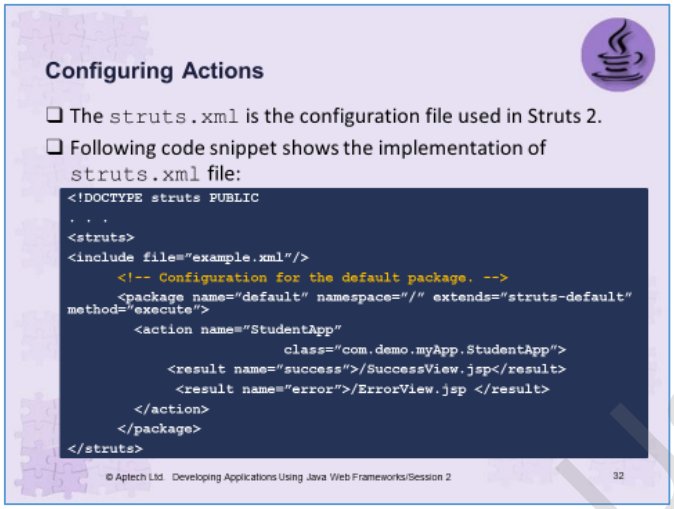



Use slide 31 to explain actions return value. After the completion of action, the `execute()` method returns a control string. This string helps the Struts 2 filter to decide the result/view page that should be rendered.

The mapping of the resultant string and the view page to be rendered is defined in the configuration file named `struts.xml`.

## Slide 32

Let us understand configuring actions.




Use slide 32 to explain configuring actions. Struts framework contains two main configuration files, `struts.xml` file and `struts.properties` file.

The `struts.properties` file is used to override the default values of `default.xml` file provided by struts framework.

The following code snippet shows an example for `struts.xml` file:

```
<?xml version="1.0" encoding="UTF-8" ?>
. . .
<struts>
<package name="default" extends="struts-default">
<action name="StudentApp" class="com.demo.myApp.StudentApp">
  <result name="success">/SuccessView.jsp</result>
   <result name="error">/ErrorView.jsp </result>
</action>
</package>
</struts>
```

The `<package>` tag is used to group together configuration information that share common attributes such as Interceptor stacks or URL namespace. Thus, packages help to group together the application's components based on commonality of function or domain. Packages group the following components into a logical configuration unit:

- Actions
- Result Types
- Interceptors

- `Interceptor stacks`

The `extends` attribute indicates the name of the parent package. It specifies the package name whose components will be inherited by the current package that is being defined. This attribute is similar to the `extends` keyword in Java. In the given code snippet, the `extends` attribute contains the value '`struts-default`'.

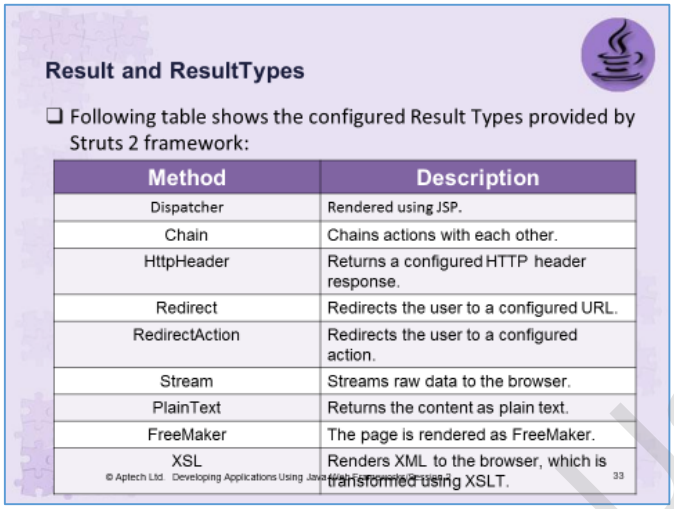
By extending the `struts-default` package, the action will by default inherit the commonly needed Struts 2 components ranging from complete Interceptor stacks to all the common result types.

The **<**`action`**>** tag maps an identifier with an action class. The framework uses action's name to process the request. This action name is mapped with the request URL. The attributes of the `<action>` tag are `name`, `class`, and `method`**.**

The `<result>` tag specifies the name of the result page. Every action element can have one or more result elements. The `<include>` tag allows the inclusion of other configuration files.

## Slide 33

Let us understand result and result types.




Use slide 33 to explain result and result types.

After processing of the action, the result is sent in two parts, the result type and the result. Result defines the next action after the processing of the action is completed. The action returns a String value such as ‘success’ or ‘error’, as a result which is used to select the result element.

The default result type `dispatcher` is used to dispatch to JSP pages. Struts allows to use other markup languages for the view to present the results that includes Velocity, Freemarker, XSLT, and Tiles. The default type, dispatcher result is used if no other result type is specified. It is used to forward to a servlet, JSP, HTML page, and so on, on the server. It uses the `RequestDispatcher.forward()` method.

Then, explain the different result types provided by Struts 2 framework as explained on slide 33.

## Slide 34

Let us understand configuring result.


Use slide 34 to explain configuring result.

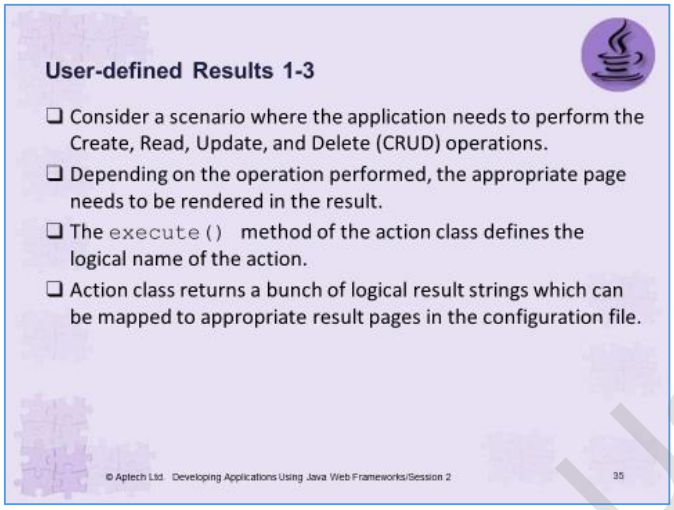
Tell the students when method of action class completes, it returns a String value. The value of the String 'success', 'input', and 'error', and so on is used to select a result element in `struts.xml` configuration file.

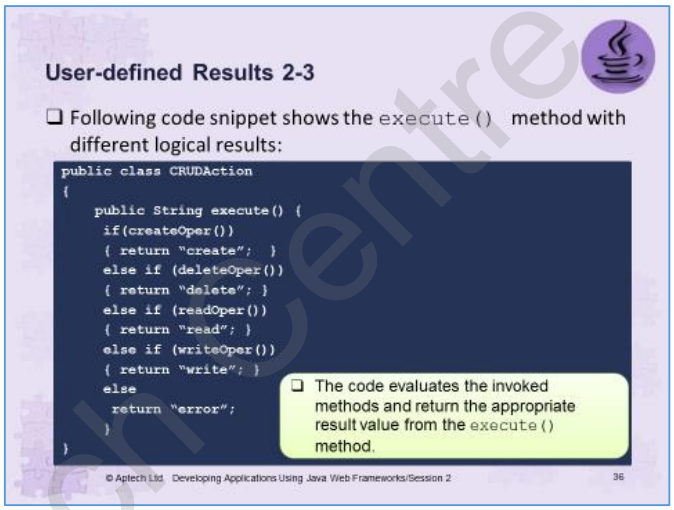
The String value matched with an action mapping will often have a set of results representing different outcomes. A standard set of result tokens are defined by the `ActionSupport` base class.

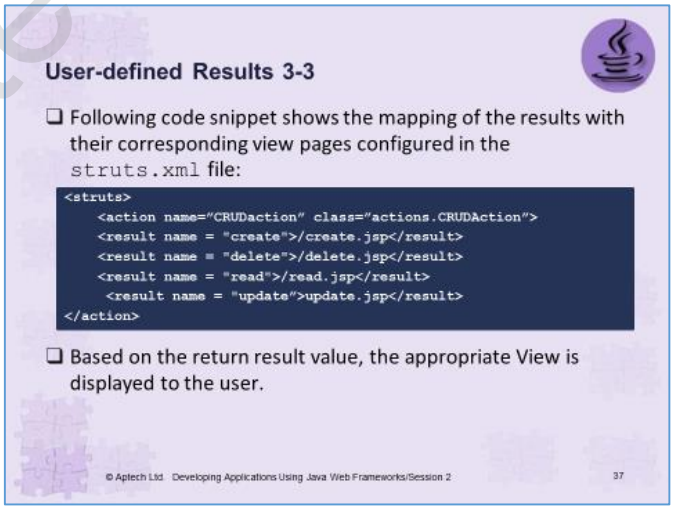
Then, explain the code snippet as given on slide 34 to set the default result types in `struts.xml` file. The code declares `dispatcher` as the default result type for the package. This means each request will be forwarded to a Web resource such as JSP to render the result to the client.

## Slides 35 to 37

Let us understand user-defined results.




Use slides 35 to 37 to explain user-defined results.

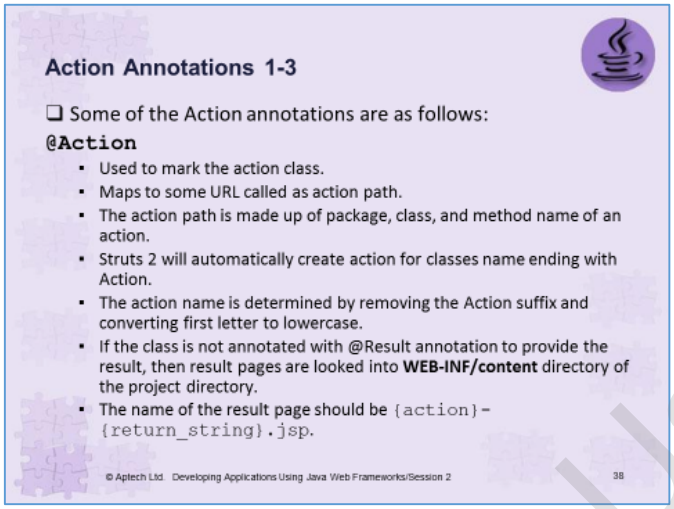
The `execute()` method of the action class defines the logical name of the action. Apart from the pre-defined CONSTANTS result names, the action class can also return a bunch of logical result strings which can be mapped to appropriate result pages in the configuration file.

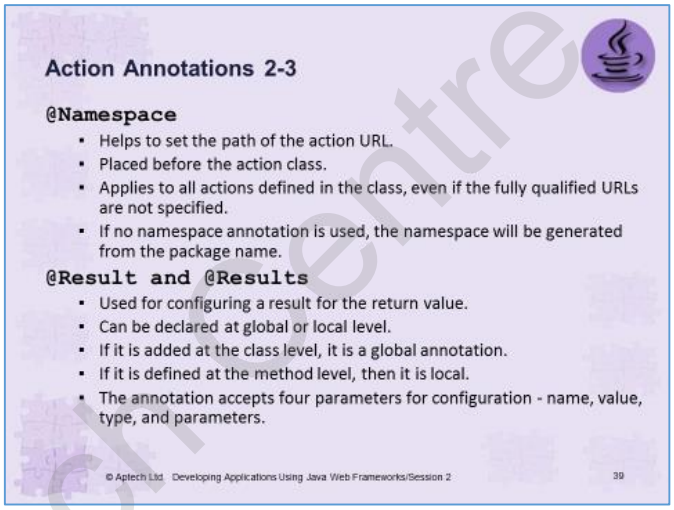
Then, explain the code snippet as given on slide 36 that evaluates the invoked methods and return the appropriate result value from the `execute()` method. Based on the return result value, the appropriate View is displayed to the user.

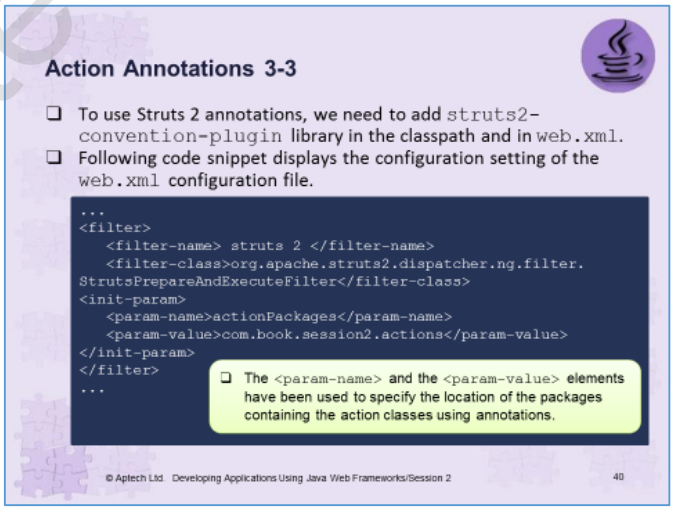
Then, explain the mapping of the results with their corresponding view pages configured in the `struts.xml` file as shown on slide 37.

## Slides 38 to 40

Let us understand Action annotations.




Use slides 38 to 40 to explain action annotations.

Use of the Zero Configuration terminology means departure from XML based configuration to an annotation-based configuration. The requirement for `struts.xml` is eliminated with the introduction of the annotation.

Annotations allow the developers to add meta-data information into the Java source code. Struts 2 uses annotations. Annotations are added on Java classes that implement the `Action` interface.

To use Struts 2 annotations, we need to add `struts2-convention-plugin` library in the classpath and in `web.xml`. Struts 2 filter configuration provide the Java package that contains action classes.

Some of the action annotations are as follows:

- `@Actions` Annotation - Group of `@Action annotations`, maps multiple URLs to the same action.
- `@Action` Annotation - Defines the URL of an action. This annotation is used to mark the action class. It maps to a URL called as action path.
- `@InterceptorRefs` Annotation - Group of `@InterceptorRef` annotations.
- `@InterceptorRef` Annotation – Interceptor or interceptors stack to be applied to an action.
- `@Results` Annotation - Group of `@Result annotations`.
- `@Result` Annotation - Defines a result for an action.
- `@ParentPackage` Annotation - Set the parent package of the actions.
- `@ExceptionMappings` - Group of `@ExceptionMapping annotations`.
- `@ExceptionMapping` - Defines an exception mapping.
- `@Namespace` - Helps to set the path of the action URL.

The following example shows the use of `@Action` annotation:

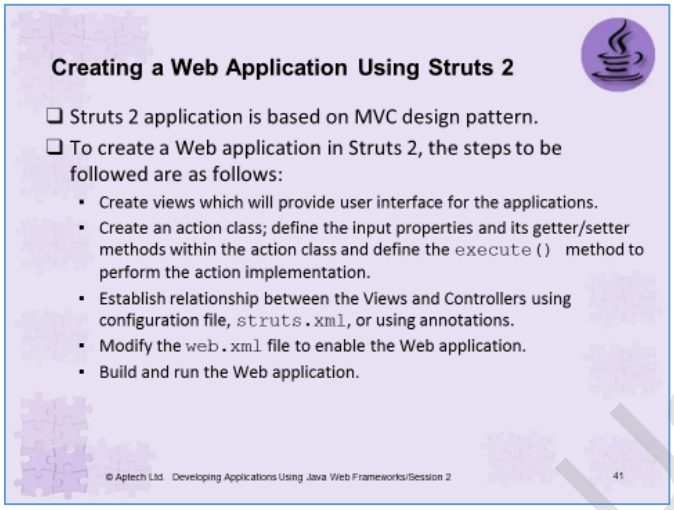
```
. . .
public class LoginAction extends ActionSupport{
{
private String name;
private String pwd;
public String getName() {
```

```
return name;
}
public void setName(String name) {
this.name = name;
}
public String getPwd() {
return pwd;
}
public void setPwd(String pwd) {
this.pwd = pwd;
}
// Applying action annotation
@Action(value="/login")
public String execute() throws Exception {
if("admin".equals(getName()) && "admin".equals(getPwd()))
return "SUCCESS";
else
return "ERROR" ;

}
}
```

If the class is not annotated with `@Result` annotation to provide the result, then result pages are looked into `WEB-INF/content` directory of the project directory. The name of the result page should be `{action}-{return_string}.jsp`. So, if `LoginAction` action class is returning '`SUCCESS`', the request will be forwarded to `WEB-INF/content/login-success.jsp` page. Thus, to avoid default naming conventions, the developers should apply proper annotation to the class with proper configuration.

## Slide 41

Let us understand creating a Web application using Struts 2.




Use slide 41 to explain creating a Web application using Struts 2.

Tell the students that when you click a hyperlink or submit an HTML form in a Struts 2 Web application, the input is collected by the Controller which is sent to a Java class called Actions. After the Action is executed, a Result selects a resource to render the response. The resource is basically a JSP, it can also be a PDF file, an Excel spreadsheet, or a Java applet window.

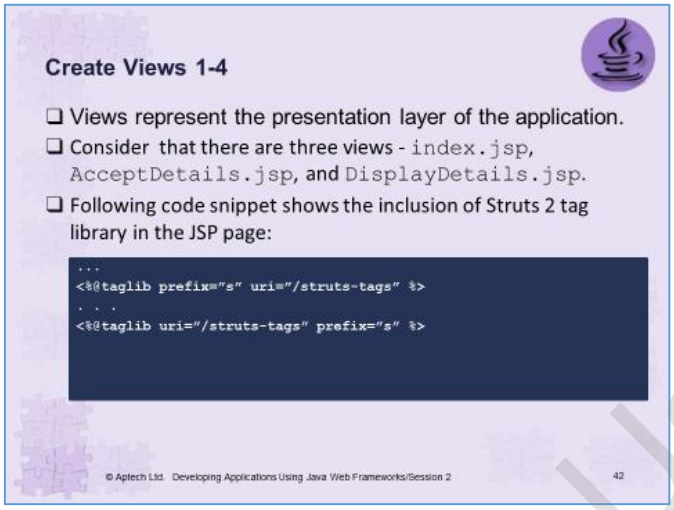
Struts 2 application is based on MVC design pattern. To create a Web application in Struts 2, you need to identify the Model, View, and Controller.

The steps to be followed are:

- Create views which will provide user interface for your applications.
- Create an action class; define the input properties and its getter/setter methods within the action class and define the `execute()` method to perform the action implementation.
- Establish relationship between the Views and Controllers using configuration file, `struts.xml`, or using annotations.
- Modify the `web.xml` file to enable the Web application with Struts 2 filter.
- Build and run the Struts 2 Web application.

## Slides 42 to 45

Let us understand creating Views.






Use slides 42 to 45 to explain creating Views.

Explain the code snippet as given on slides 42 and 43 demonstrating the `index.jsp` page. The **index.jsp** page uses Struts 2-specific HTML tag. This is the first page of the application where the user sees a Welcome Message and a link. The `<s:a href>` tag is used to emulate an HTML link. On clicking this link the user is taken to another JSP page.

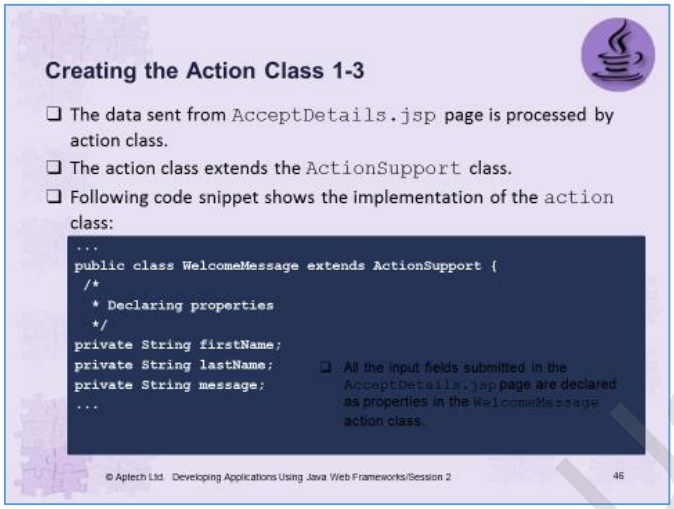
Then, explain the code snippet given on slide 44 that demonstrates the `AcceptDetails.jsp` page. The `<s:textfield/>` emulates an HTML text field. This text field allows the user to enter text. The `<s:submit/>` tag emulates an HTML Submit button. On clicking the Submit button, the `<s:form/>` tag is created and populated with the values from the `<s:textfield/>` tag.

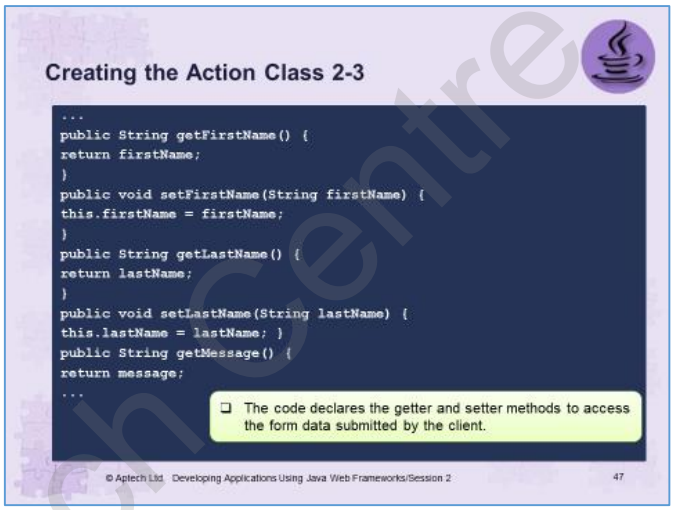
Finally, explain the code snippet given on slide 45 that accepts the user details. When the user submits the form, the **firstname** and **lastname** entered in the **AcceptDetails.jsp** page. The view is displayed only after the successful submission of the form by the user.

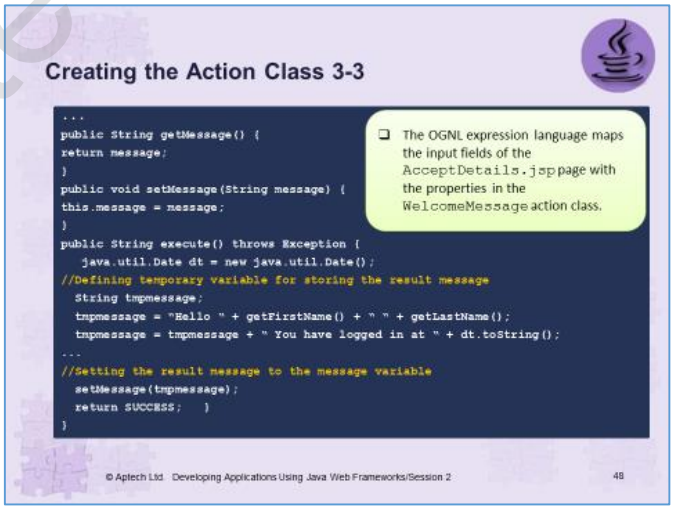
In this code, the `<s:property value="message"/>` tag is used to automatically find the message property and print the value in the message property.

## Slides 46 to 48

Let us understand creating the Action Class.



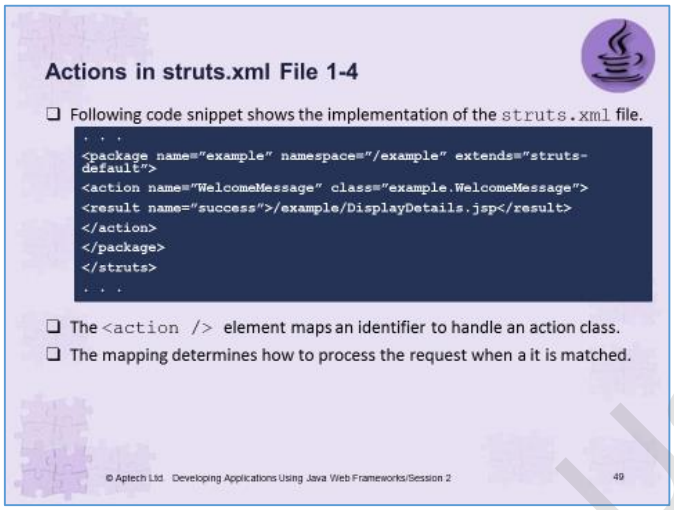

Use slides 46 to 48 to explain creating the Action class.

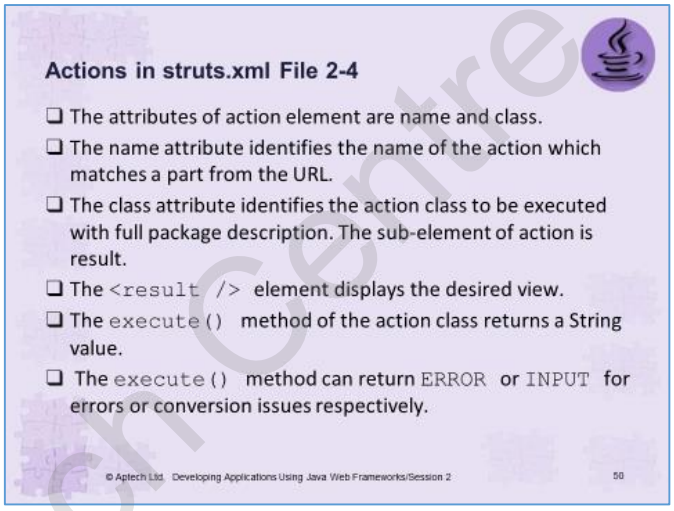
Tell the students that Action class responds to a user action when user clicks a URL. One or more of the Action class's methods are executed and a String result is returned. A specific JSP page is provided based on the value of the result.

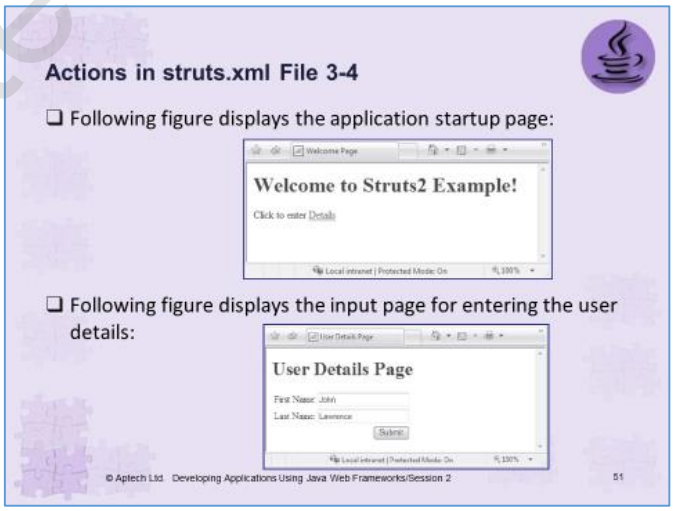
In the code, all the input fields submitted in the **AcceptDetails.jsp** page are declared as properties in the **`WelcomeMessage`** action class. The getter/setter methods for these properties have been declared. The OGNL expression language maps the input fields of the **AcceptDetails.jsp** page with the properties in the **`WelcomeMessage`** action class. Thus, for this reason the input field names of the **AcceptDetails.jsp** page are the same as the names of the properties in the **`WelcomeMessage`** action class.

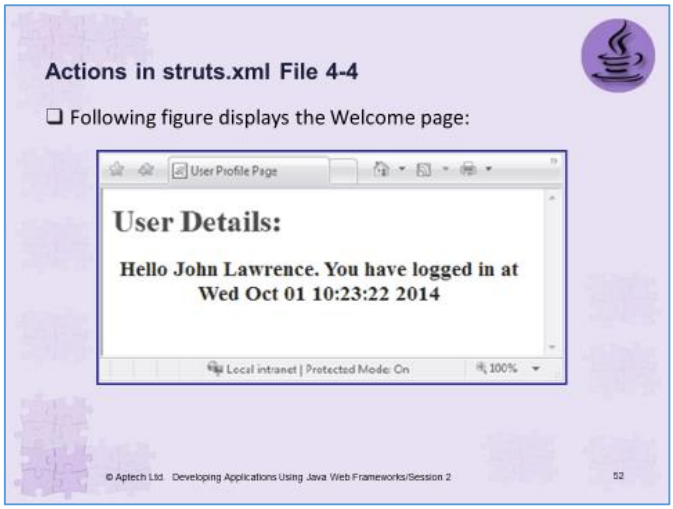
## Slides 49 to 52

Let us understand actions in `struts.xml` file.






Use slides 49 to 52 to explain configurations in `struts.xml` file.

The <`action /`> element maps an identifier to handle an action class. The mapping is used by the action's name and framework to determine how to process the request when a request is matched.

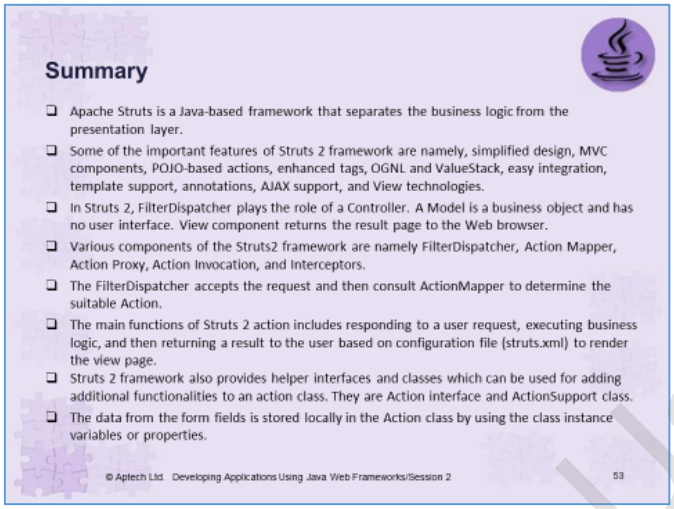
The attributes of action element are `name` and `class`. The `name` attribute identifies the name of the action which matches a part from the URL. The `class` attribute identifies the action class to be executed with the full package description. The sub-element of **action** is `result`.

The <`result /`> element displays the desired view. The `execute()` method of the action class returns a String value. This string value is used to select a <`result /`> element. For example, if the `execute()` method of the action class named **WelcomeMessage.java** returns **SUCCESS**, then this value is matched with the result named, **success**. After matching the value, the **DisplayDetails.jsp** page is invoked to display the result. The `execute()` method can return **ERROR** or **INPUT** for errors or conversion issues, respectively. These values should then be matched with the particular result name present in the <`result /`> element and a JSP page should be rendered.

The output of the code execution is given on slides 51 and 52.

## Slide 53

Let us summarize the session.




In slide 53, you will summarize the session. You will end the session, with a brief summary of what has been taught in the session.

### 2.3 Post Class Activities for Faculty

You should familiarize yourself with the topics of the next session. You should also explore the concept and working of the interceptors and different types of tags in Struts 2.

**Tips:**

You can also check the Articles/Blogs/Expert Videos uploaded on the OnlineVarsity site to gain additional information related to the topics covered in the next session. You can also connect to online tutors on the OnlineVarsity site to ask queries related to the sessions.

# Session 3 – Struts 2 - Interceptors and Tags

## 3.1 Pre-Class Activities

Before you commence the session, you should familiarize yourself with the topics of this session in-depth. You should revisit topics of the previous session for a brief review.

Here, you can ask students the key topics they can recall from previous session. Prepare a question or two which will be a key point to relate the current session objectives.

### 3.1.1 Objectives

By the end of this session, the learners will be able to:

- Describe the purpose of Interceptors in Struts 2 framework
- Explain different types of built-in Interceptors
- Explain how to configure Interceptors in Struts 2 application
- Describe Interceptors Stacks
- Explain the process of creating custom Interceptors in Struts 2 framework
- List the different types of tags used in Struts 2 View page
- Explain various Data tags
- Explain various Control tags
- Explain various User Interface (UI) tags

### 3.1.2 Teaching Skills

To teach this session successfully, you should be aware of Interceptors and different types of built-in Interceptors in Struts 2 framework. Familiarize yourself with the configuration of Interceptors in Struts 2 application and the process of creating custom Interceptors in Struts 2 framework.

You should also know about interceptor stack and different types of tags used in Struts 2 View page. Aware yourself with Data tags, Control tags, and User Interface (UI) tags.

For teaching in the class, you are expected to use slides and LCD projectors.

**Tips:**

It is recommended that you test the understanding of the students by asking questions in between the class.

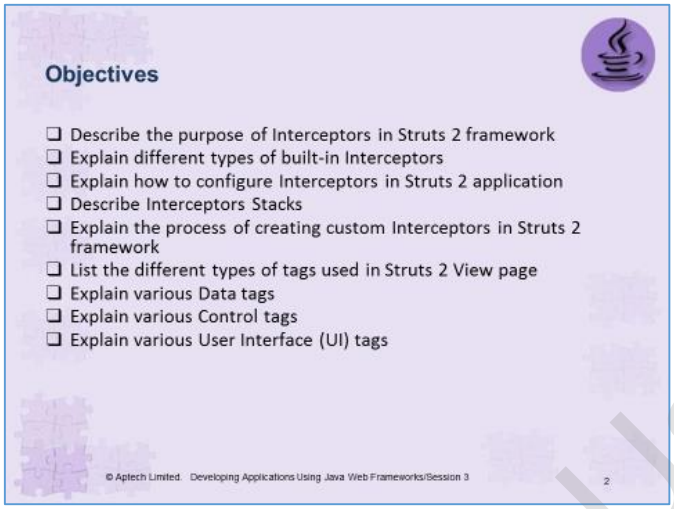
**In-Class Activities:**

Follow the order given here during In-Class activities.

**Overview of the Session:**

Give the students a brief overview of the current session in the form of session objectives.

Show the students slide 2 of the presentation.



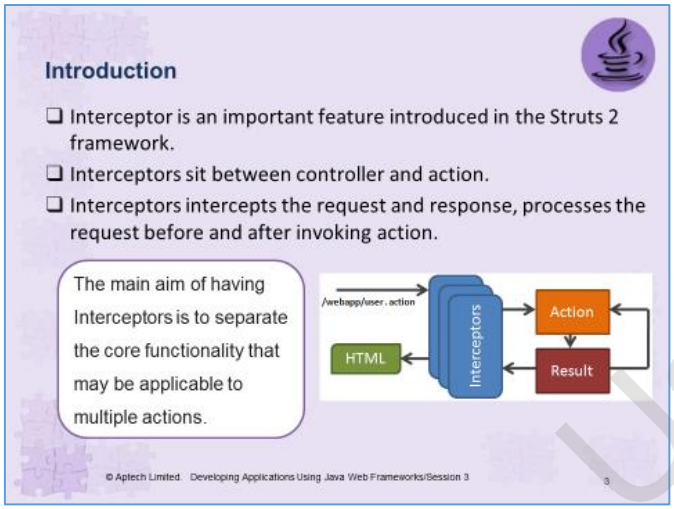

Tell the students that they will be introduced to interceptors and different types of built-in Interceptors in Struts 2 framework. Explain the configuration of Interceptors in Struts 2 application and the process of creating custom Interceptors in Struts 2 framework.

In this session, they will learn about Interceptor stack and different types of tags used in Struts 2 View page. Finally, discuss about Data tags, Control tags, and User Interface (UI) tags.

## 3.2 In-Class Explanations

### Slide 3

Let us introduce interceptors.




Use slide 3 to introduce interceptors.

Tell them that Interceptor is an important feature introduced in the Struts 2 framework. Interceptors sit between controller and action. They intercepts the request and response. In other words, Interceptors provide a mechanism to supply pre-processing and post-processing functionalities required by action classes.

Interceptor is an object which intercepts an action dynamically. It allows the developers to write a code which can be executed by the controller before and after invoking actions. Each and every Interceptor object is pluggable, so user can decide exactly which features an action need to support.

The main aim of having Interceptors is to separate the core functionality that may be applicable to multiple actions.

### In-Class Question:

After you finish explaining interceptors, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.

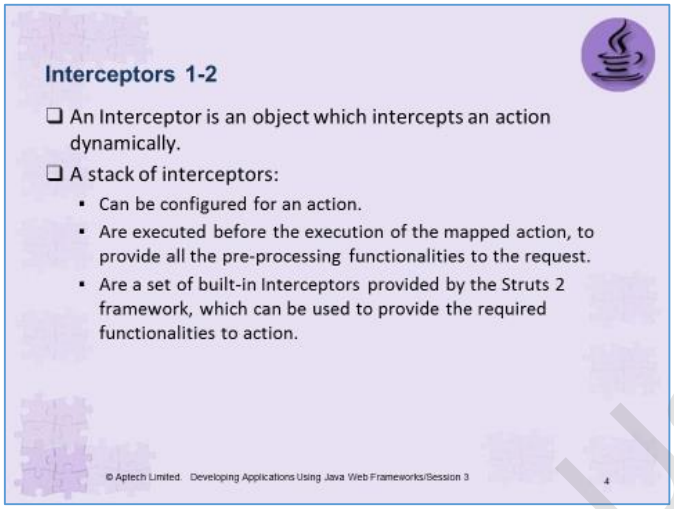
Which feature of Struts 2 intercepts the request and response and processes the request before and after invoking action?

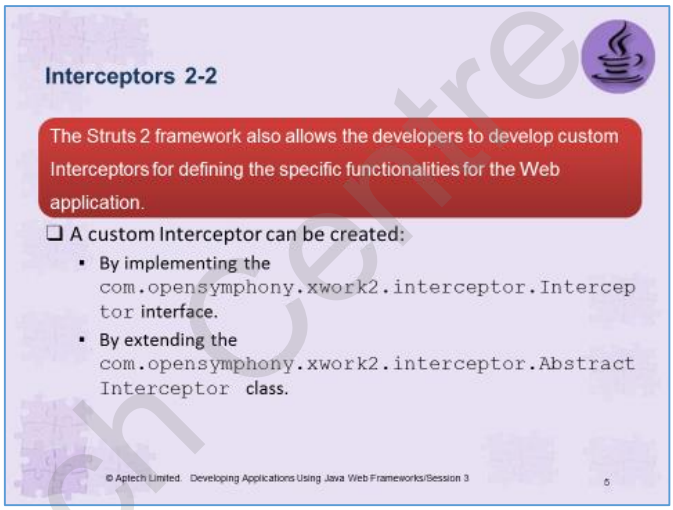
### Answer:

Interceptor

## Slides 4 and 5

Let us understand stack of interceptors and custom interceptors.




Use slides 4 and 5 to explain stack of interceptors and custom interceptors.

Tell them that struts 2 framework provide a default interceptor stack which fulfils the needs of most of the struts applications. These interceptors are referred to as built-in interceptors.

These built-in Interceptors can be used to provide the required functionalities to action. The Struts 2 framework also allows the developers to develop custom Interceptors for defining the specific functionalities for the Web application.

**Tips:**
A custom Interceptor class can be created by implementing the `com.opensymphony.xwork2.interceptor.Interceptor` interface or by extending the `com.opensymphony.xwork2.interceptor.AbstractInterceptor` class.

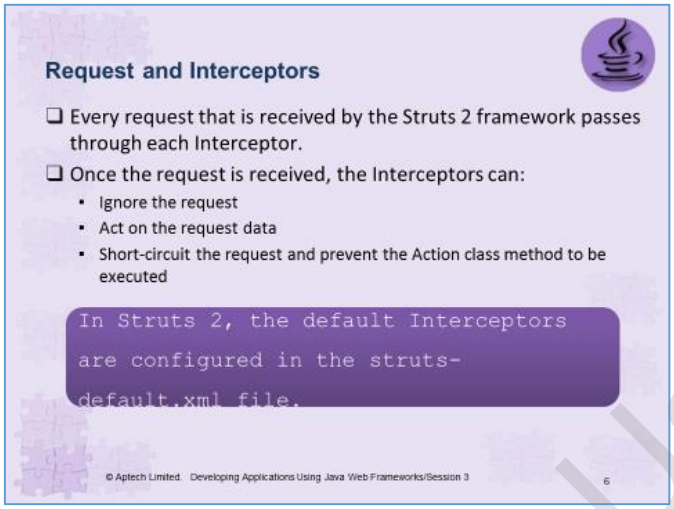
The following code snippet shows the syntax of interceptor interface used for creating custom interceptors:

```
public interface Interceptor extends Serializable{
   void destroy();
   void init();
   String intercept(ActionInvocation invocation) throws
Exception;
}
```

- `init()` method initializes the interceptor.
- `destroy()` method provides a facility for interceptor cleanup.
- `ActionInvocation` object provides access to the runtime environment. It allows access to the action itself and methods to invoke the action and determine whether the action has already been invoked.

## Slide 6

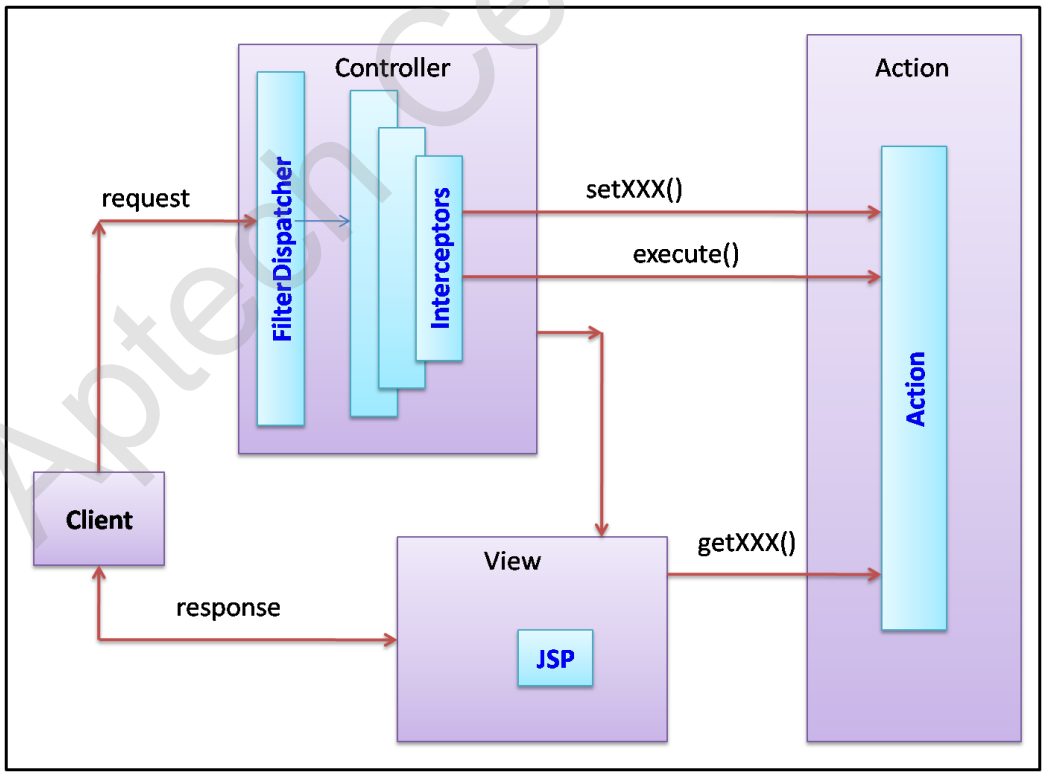
Let us understand about request and interceptors.




Use slide 6 to explain request and interceptors.

Interceptors are responsible for most of the request processing. Every request that is received by the Struts 2 framework passes through each Interceptor.

The following figure shows the request and response with configured interceptors:



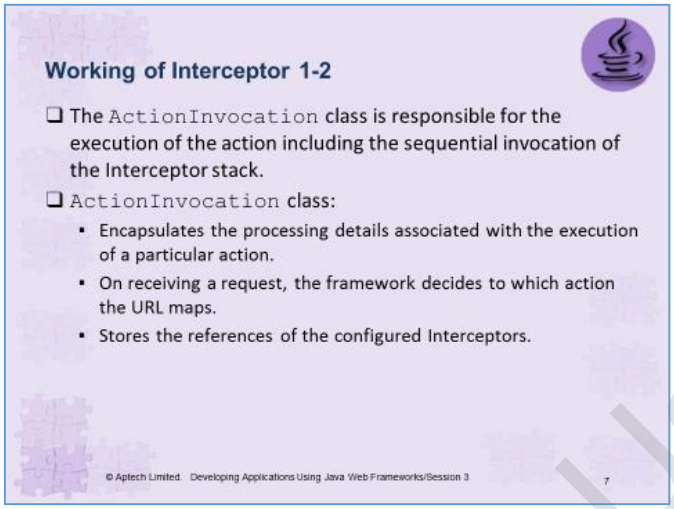
Some of the actions taken by interceptor on receiving the request are as follows:

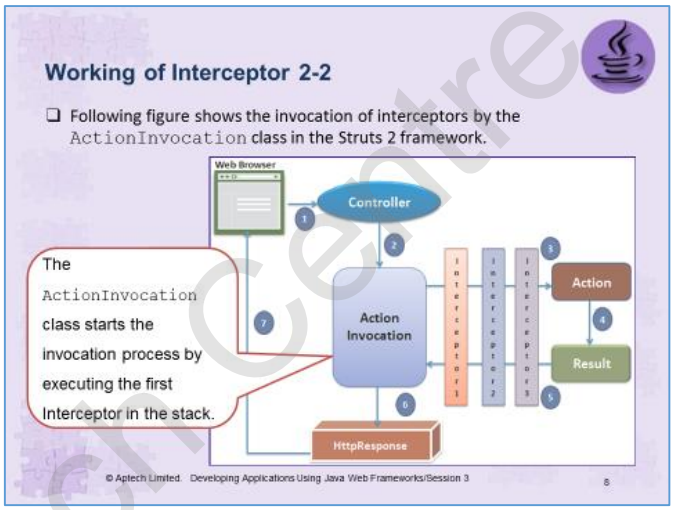
- Ignore the request
- Act on the request data
- Short-circuit the request and prevent the Action class method to be executed

In Struts 2, the default Interceptors are configured in the `struts-default.xml` file. The `struts-default.xml` is included in the application's configuration by default, so all of the predefined interceptors are provided in the Struts 2 framework.

## Slides 7 and 8

Let us understand how Interceptors works.




Use slides 7 and 8 to explain how Interceptors works.

**ActionInvocation Class**

Tell them that the `ActionInvocation` class is responsible for the execution of the interceptors. `ActionInvocation` class encapsulates the processing details associated with the execution of a particular action. On receiving a request, the framework decides to which action the URL maps.

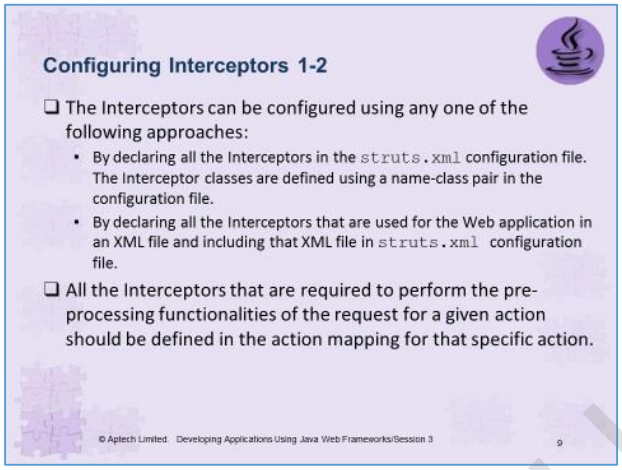
Struts 2 framework creates an object `ActionInvocation` that contains the action and all the interceptors configured for that action. Each interceptors are called before the action is called. Once the action is called and result is generated, each interceptors are again called in reverse order to perform post processing work. Interceptors can change the workflow of action. It can prevent the execution of action.

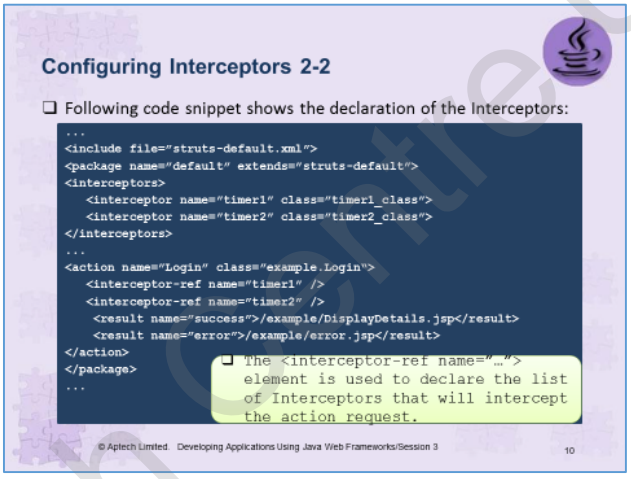
The `ActionInvocation` class stores the references of these Interceptors, besides storing other important information such as Servlet request objects and so on. Once the `ActionInvocation` class contains all the required information, it exposes the `invoke()` method to the framework. The framework invokes this method and the `ActionInvocation` class starts the invocation process by executing the first Interceptor in the stack.

All the Interceptors are invoked one after the other by recursively invoking the `invoke()` method of the `ActionInvocation` class. The `ActionInvocation` class checks the states and executes whichever Interceptor comes next. After all the Interceptors have been invoked, the `invoke()` method will cause the action to be executed.

## Slides 9 and 10

Let us understand how to configure interceptors.




Use slides 9 and 10 to explain how to configure interceptors. The following approach can be used to configure interceptors:

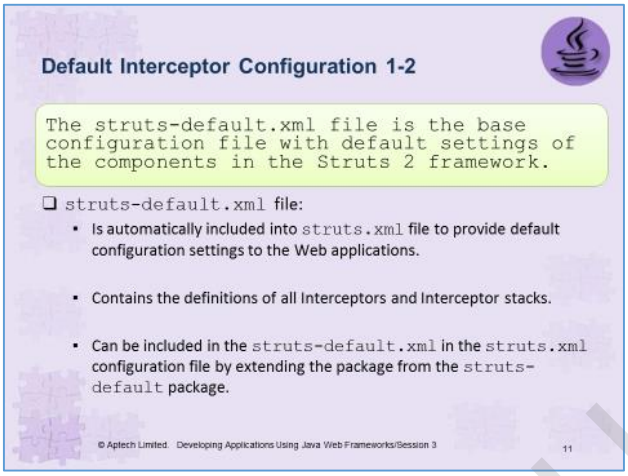
- By declaring all the Interceptors in the `struts.xml` configuration file.
- By declaring all the Interceptors that are used for the Web application in an XML file and including that XML file in `struts.xml` configuration file.

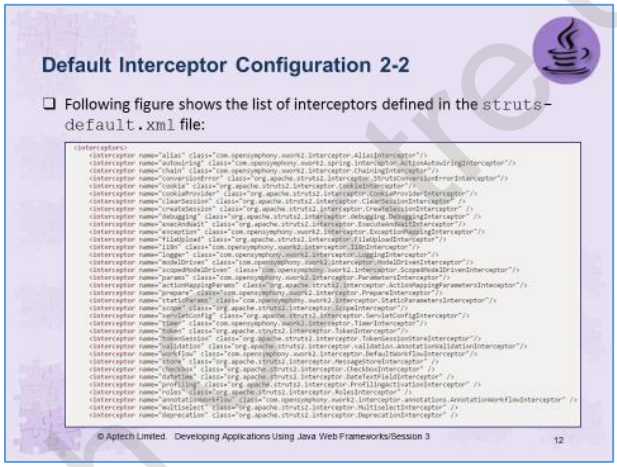
Then, explain them how to configure interceptors in `struts.xml` file as explained in the given code snippet on slide 10. In the code, every Interceptor class is provided a name which is used for reference. The `<interceptor-ref name="…">` element is used to declare the list of Interceptors that will intercept the action request. Interceptors are executed in the same order in which they are declared in the action mapping.

In the code, the Interceptor classes will be executed to provide all pre-processing functionalities before the execution of the **Login** action class.

## Slides 11 and 12

Let us understand default interceptor configuration.




Use slides 11 and 12 to explain default interceptor configuration.

The `struts-deafult.xml` file is the base configuration file with default settings of the components in the Struts 2 framework. This file is automatically included into `struts.xml` file to provide default configuration settings to the Web applications.

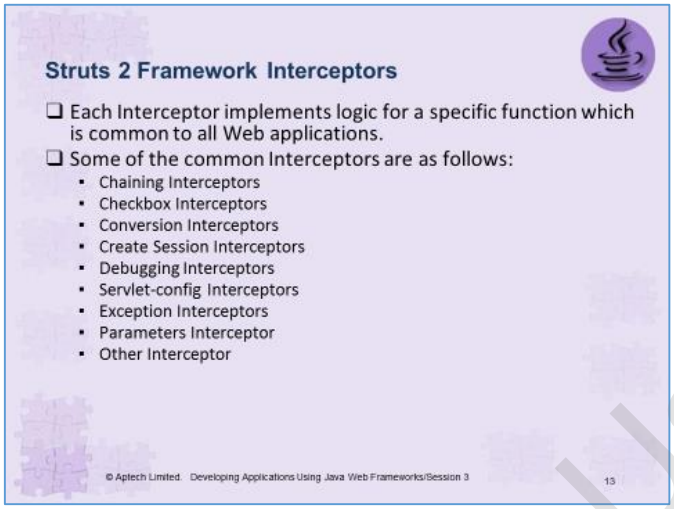
The `struts-default.xml` file contains the definitions of all Interceptors and Interceptor stacks. To use the Interceptors bundled with the framework, user can include `struts-default.xml` in the `struts.xml` configuration file by extending the package from the `struts-default` package. Discuss the list of interceptors mentioned in `struts-default.xml` as shown on slide 12.

**Tips:**

The `struts-default.xml` file is contained in the `struts2-core.jar` present in the `lib` folder of the Struts 2 software package.

## Slide 13

Let us understand Struts 2 framework interceptors.




Use slide 13 to explain Struts 2 framework interceptors.

Tell the students that Interceptor classes are defined using a key-value pair specified in the Struts configuration file.

Further, discuss the interceptor element defined in the `struts-default.xml` file. All interceptor elements are nested within the interceptors element and are associated with a name and a qualified class name.

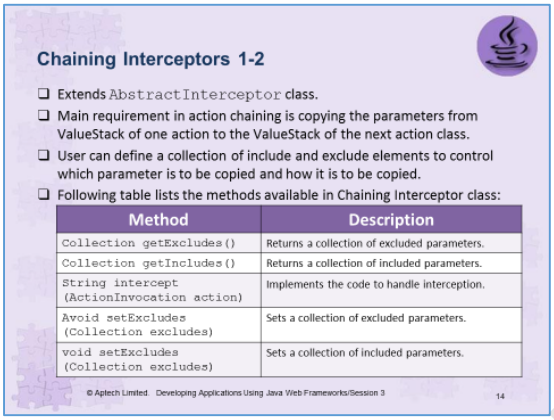
Following points are required to be taken care of when you are using framework Interceptors:

- `struts-default.xml` file should be included in the `struts.xml` file.
- Action mappings should be enclosed within the `<package>` element which extends `struts-default` package.

Then, list some of the common interceptors as mentioned on slide 13.

## Slides 14 and 15

Let us understand Chaining interceptors.





Use slides 14 and 15 to explain Chaining interceptors.

The Chaining class extends `AbstractInterceptor` class. The function of this Interceptor is to copy all the objects present in the ValueStack of the currently executing action class to the ValueStack of the next action class to be executed in action chaining.

Thus, the main requirement in action chaining is copying the parameters from ValueStack of one action to the ValueStack of the next action class. You can define a collection of include and exclude elements to control which parameter is to be copied and how it is to be copied.

The table as shown on slide 15 explains the methods of Chaining Interceptor class. If there are no objects in the ValueStack, the interceptor does nothing. It works with 'chain' result type in `struts.xml` file.

All objects including first action will be available in the target action's ValueStack.

The following additional code snippet shows how to configure the chain interceptors:

```
<package name="default" extends="struts-default">

<action name="ValidateAction" class="example.ValidateAction">
     <interceptor-ref name="basicStack"/>
     <result name="success" type="chain">action1</result>
    </action>

    <action name="LoginAction" class="example.Action">
     <interceptor-ref name="chain"/>
     <interceptor-ref name="basicStack"/>
     <result name="success">Success.jsp</result>
    </action>

</package>
```

**In-Class Question:**

After you finish explaining Chaining interceptors, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.

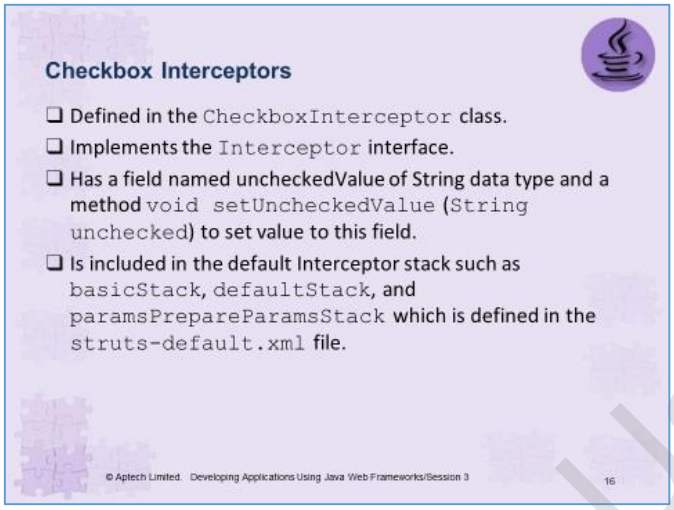
Which interceptor makes the previous `Action`'s properties available to the current `Action`?

**Answer:**

Chaining interceptors

## Slide 16

Let us understand Checkbox interceptor.




Use slide 16 to explain Checkbox interceptors.

Tell the students that Checkbox interceptors adds automatic checkbox handling code that detect an unchecked checkbox and add it as a parameter with a default value. The default value is usually false. The `org.apache.struts2.interceptor.CheckboxInterceptor` class is in the `defaultStack`. The class has a field named uncheckedValue of String data type and a method `void setUncheckedValue (String unchecked)` to set value to this field.

It is included in the default Interceptor stack such as `basicStack`, `defaultStack`, and `paramsPrepareParamsStack` which is defined in the `struts-default.xml` file.

Checkbox interceptors uses a specially named hidden field to detect an un-submitted checkboxes.

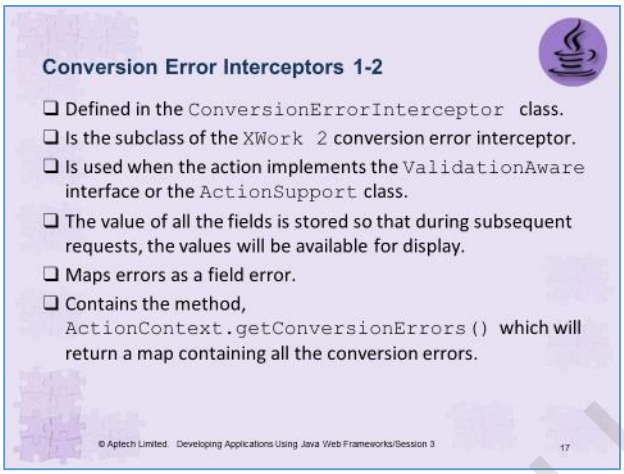
### Tips:

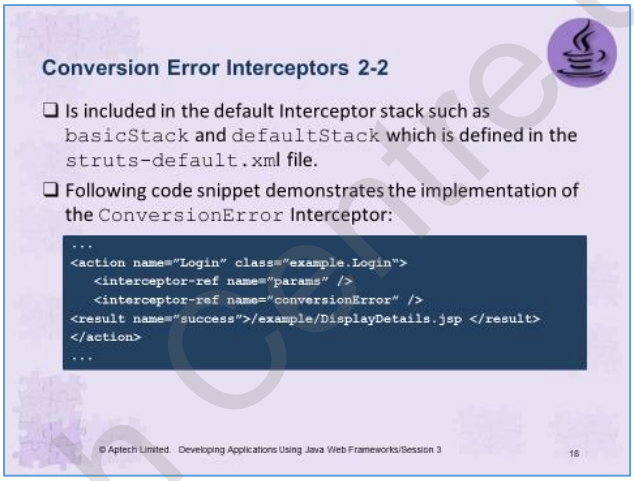
The `CheckInterceptor` contains a parameter named `setUncheckedValue`. The default value of an unchecked box is false which can be overridden by setting the '`uncheckedValue`' property.

For an example on CheckInterceptor, you can refer to this link:
http://www.mkyong.com/struts2/struts-2-scheckbox-checkbox-example/

## Slides 17 and 18

Let us understand ConversionError interceptors.




Use slides 17 and 18 to explain ConversionError interceptors.

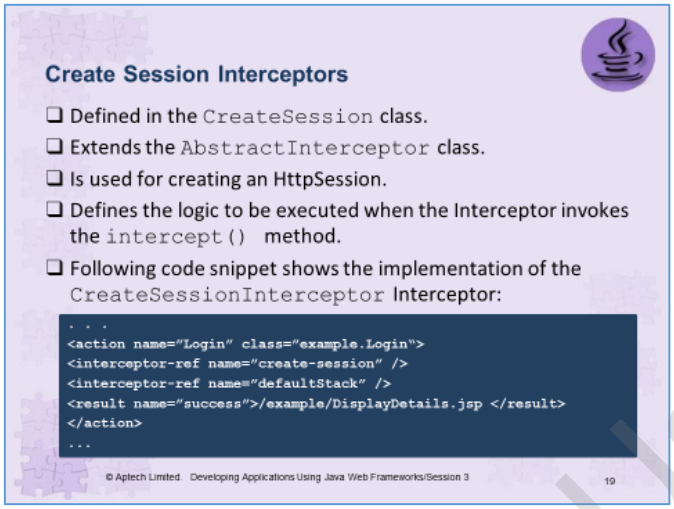
Mention that the Struts 2 Conversion Error interceptor is a subclass of the XWork 2 Conversion Error interceptor.Conversion Error interceptor adds conversion errors from the `ActionContext` to the `Action`'s field errors.

This interceptor adds any error found in the `ActionContext`'s `conversionErrors` map as a field error provided the action implements `ValidationAware`. The method, `ActionContext.getConversionErrors()` will return a map containing all the conversion errors. In addition, any field that contains a validation error has its original value saved. Thus, any subsequent requests for that value return the original value rather than the value in the action.

Then, explain the configuration of `ConversionError` interceptor as shown on slide 18.

## Slide 19

Let us understand Create Session interceptors.




Use slide 19 to explain Create Session interceptors.

Tell the students that this interceptor creates the `HttpSession` if it doesn't exist. `SessionMap` is also recreated and put in `ServletActionContext`. It does not have any parameters.

The following code snippet shows an example for Create Session interceptors:

```
<action name="LoginAction" class="example.LoginAction">
    <interceptor-ref name="createSession"/>
    <interceptor-ref name="defaultStack"/>
    <result name="enter">/example/DisplayDetails.jsp </result>
</action>
```

### In-Class Question:

After you finish explaining Create Session interceptors, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.

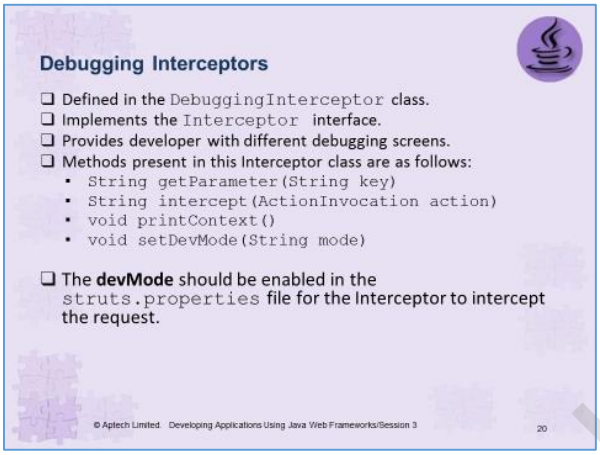
Which interceptors creates an HttpSession automatically?

### Answer:

Create Session interceptors.

## Slide 20

Let us understand Debugging interceptors.




Use slide 20 to explain Debugging interceptors.

Tell the students that Debugging interceptor provides several different debugging screens to provide insight into the data behind the page.

This interceptor is activated only when `devMode` is enabled in `struts.properties`. The 'debug' parameter is removed from the parameter list before the action is executed. All operations occur before the natural Result has a chance to execute.

Some of the parameters of the Debugging interceptor are as follows:

- `xml`: An XML document is dumped with parameters, context, session, and ValueStack.
- `console` - Shows a popup 'OGNL Console' that allows the user to test OGNL expressions against the value stack.
- `command` - Tests an OGNL expression and returns the string result.
- `browser` - Shows field values of an object specified in the `object` parameter.

Then, explain the methods present in the Debugging interceptor as explained on slide 20.

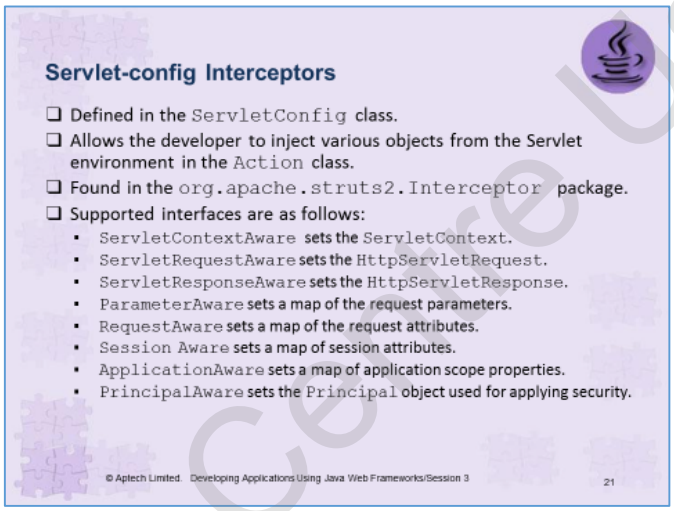
The following code snippet shows the struts.xml configuration file with debugging setting:

```
<?xml version="1.0" encoding="UTF-8" ?>
. . .
<constant name="devMode" value="true"/>
 <package name="default" extends="struts-default">
<action
```

```
ame="EmpDebugAction" class="techmyguru.actions.EmpDebugAction"
>
<interceptor-ref name="debugging"/>
  <result name="success">/WEB-INF/pages/EmpSuccess.jsp
</result>
</action>
</package>
</struts>
```

### Slide 21

Let us understand Servlet Config interceptor.




Use slide 21 to explain Servlet Config interceptor.

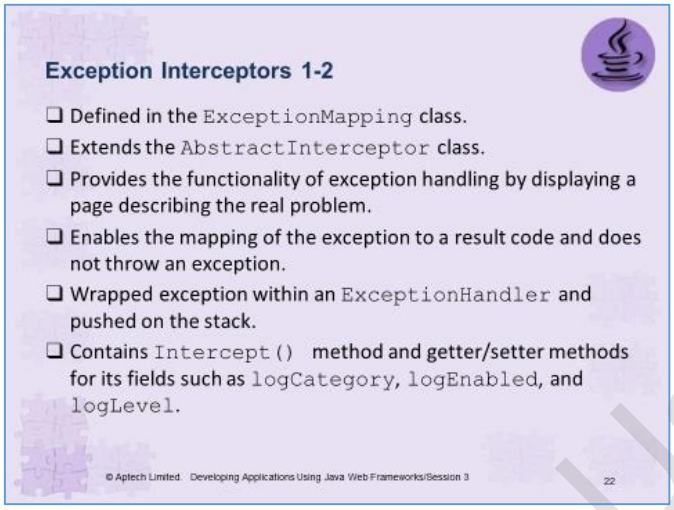
Tell the students that Servlet Config interceptor provides access to Maps representing `HttpServletRequest` and `HttpServletResponse`. Servlet Config interceptor sets action properties based on the interfaces an action implements. It is designed to set all properties an action needs if it is aware of servlet parameters, the servlet context, the session, and so on.

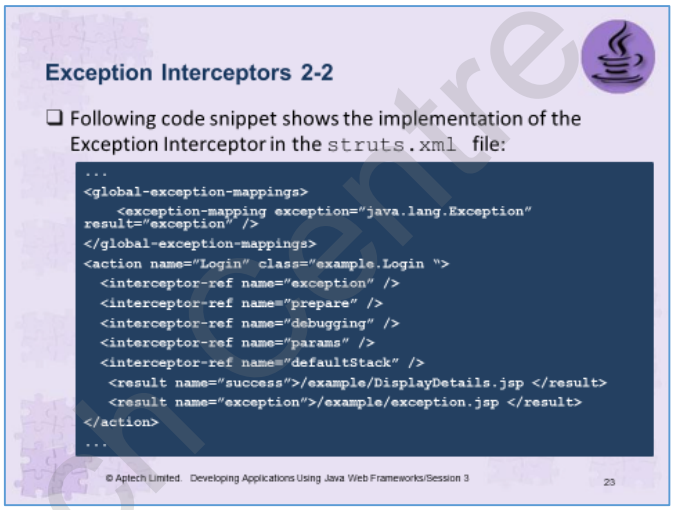
The following code snippet shows the configuration for Servlet Config interceptor.

```
<action name="someAction" class="com.examples.SomeAction">
    <interceptor-ref name="servletConfig"/>
    <interceptor-ref name="basicStack"/>
    <result name="success">good_result.ftl</result>
</action>
```

## Slides 22 and 23

Let us understand Exception interceptors.




Use slides 22 and 23 to explain Exception interceptors. Tell the students that Exception interceptor maps exceptions to a result. The Interceptor enables the mapping of the exception to a result code and does not throw an exception. The encountered exception is wrapped within an `ExceptionHandler` and pushed on the stack and thus gives access to the exception from within the user's result.

Tell them that besides the `intercept()` method, the class has getter/setter methods for its fields such as `logCategory`, `logEnabled`, and `logLevel`.

Parameters of Exception interceptors are as follows:

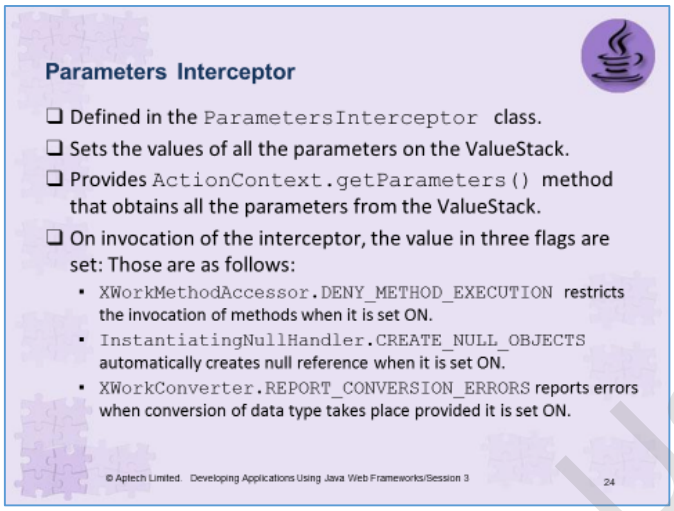
- `logEnabled (optional)` – Whether exceptions should also be logged. It has values such as boolean true or boolean.

- `logLevel` – Specifies the log level. It can use trace, debug, info, warn, error, or fatal. The default is debug.
- `logCategory` – The default category to use is `com.opensymphony.xwork2.interceptor.ExceptionMappingInterceptor`.

Then, explain the implementation of Exception interceptor in the `struts.xml` file as shown on slide 23.

## Slide 24

Let us understand Parameters interceptor.



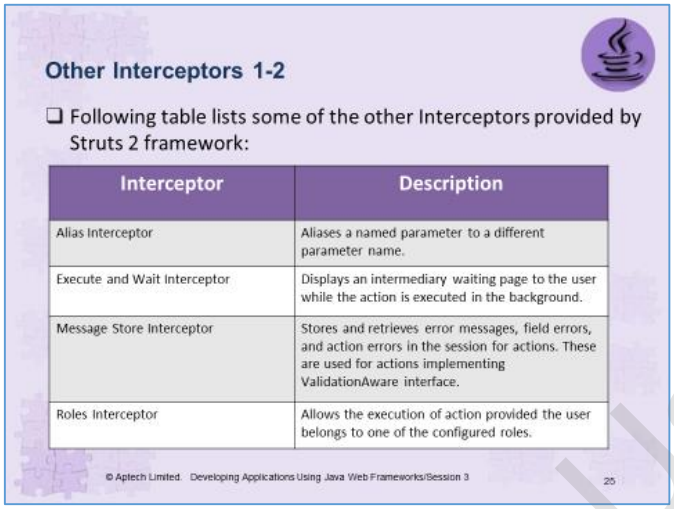

Use slide 24 to explain Parameters interceptor. Tell the students that Parameters interceptor sets the request parameters onto the ValueStack. The `ActionContext.getParameters()` method obtains all the parameters from the ValueStack. The `ValueStack.setValue(String, Object)` method is used for setting the value in the ValueStack. The values are applied to the actions present in the ValueStack after the form requests are submitted.

Parameters of Parameters interceptor are as follows:

- `ordered` - set to true if user wants the top-down property setter behavior.
- `acceptParamNames` - a comma delimited list of regular expressions to describe a whitelist of accepted parameter names.
- `excludeParams` - a comma delimited list of regular expressions to describe a blacklist of not allowed parameter names.
- `paramNameMaxLength` - the maximum length of parameter names. Parameters with longer names will be ignored. The default length is 100 characters.

## Slides 25 and 26

Let us understand other interceptors used in Struts 2.

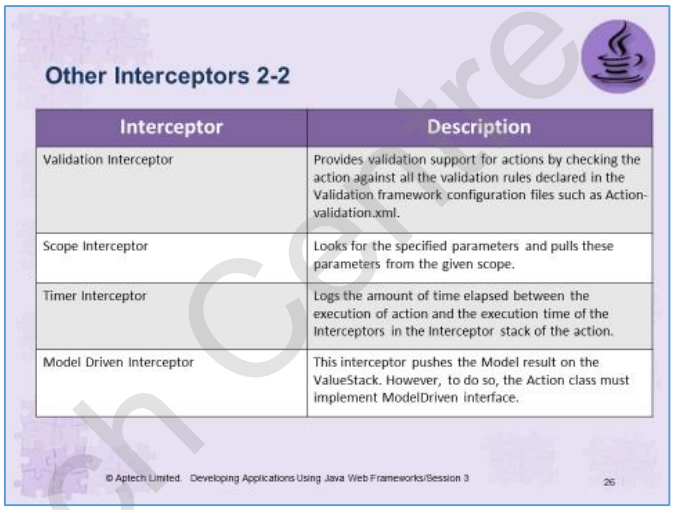


Other Interceptors 1-2

- Following table lists some of the other Interceptors provided by Struts 2 framework:

| Interceptor | Description |
|---|---|
| Alias Interceptor | Aliases a named parameter to a different parameter name. |
| Execute and Wait Interceptor | Displays an intermediary waiting page to the user while the action is executed in the background. |
| Message Store Interceptor | Stores and retrieves error messages, field errors, and action errors in the session for actions. These are used for actions implementing ValidationAware interface. |
| Roles Interceptor | Allows the execution of action provided the user belongs to one of the configured roles. |

 25



Other Interceptors 2-2

| Interceptor | Description |
|---|---|
| Validation Interceptor | Provides validation support for actions by checking the action against all the validation rules declared in the Validation framework configuration files such as Action-validation.xml. |
| Scope Interceptor | Looks for the specified parameters and pulls these parameters from the given scope. |
| Timer Interceptor | Logs the amount of time elapsed between the execution of action and the execution time of the Interceptors in the Interceptor stack of the action. |
| Model Driven Interceptor | This interceptor pushes the Model result on the ValueStack. However, to do so, the Action class must implement ModelDriven interface. |

 26

Use slides 25 and 26 to explain other interceptors used in Struts 2.

**Tips:**

The following link provides the example on Alias interceptor:
http://techmyguru.com/struts2/index.php?section=81

The following link provides the example on execAndWait interceptor:
http://www.journaldev.com/2296/struts2-execandwait-interceptor-example-for-long-running-actions

The following link provides the example on Scope interceptor:
http://struts.apache.org/docs/scope-interceptor.html

The following link provides the example on ModelDriven interceptor:
http://www.javatpoint.com/struts-2-modeldriven-interceptor-example

**In-Class Question:**

After you finish explaining other interceptors used in Struts 2, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.

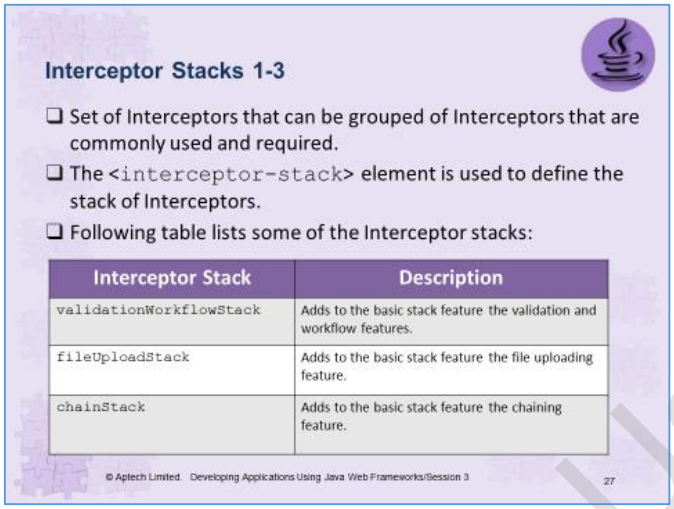
Which interceptor allows the execution of action provided the user belongs to one of the configured roles?

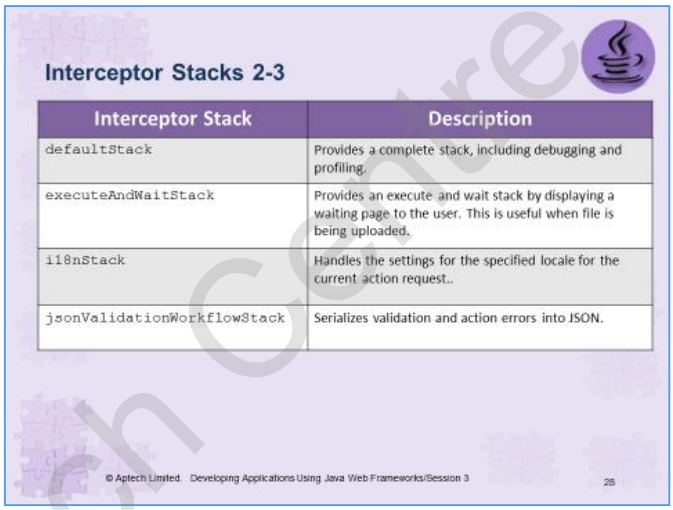
**Answer:**

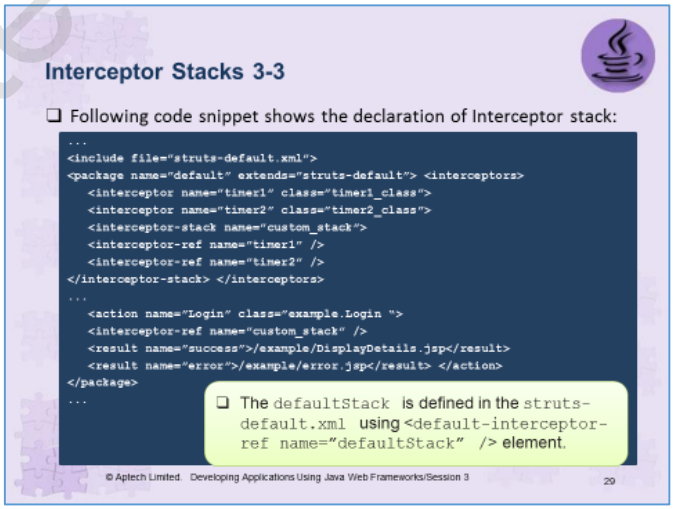
Roles Interceptor

## Slides 27 to 29

Let us understand Interceptor stacks.




Use slides 27 to 29 to explain interceptor stacks.

Interceptors are managed with interceptor stacks. The following code snippet shows an example for interceptor stacks:

```
<interceptor-stack name="mainStack">
   <interceptor-ref name="roles"/>
   <interceptor-ref name="token"/>
   <interceptor-ref name="prepare"/>
   <interceptor-ref name="alias"/>
   <interceptor-ref name="params"/>
   <interceptor-ref name="conversionError"/>
</interceptor-stack>
```

The given stake is called `mainStack`. Each `<interceptor-ref >` tag references either an interceptor or an interceptor stack that has been configured before the current interceptor stack. Therefore, the user have to ensure that the name is unique across all interceptor and interceptor stack configurations when configuring the initial interceptors and interceptor stacks.

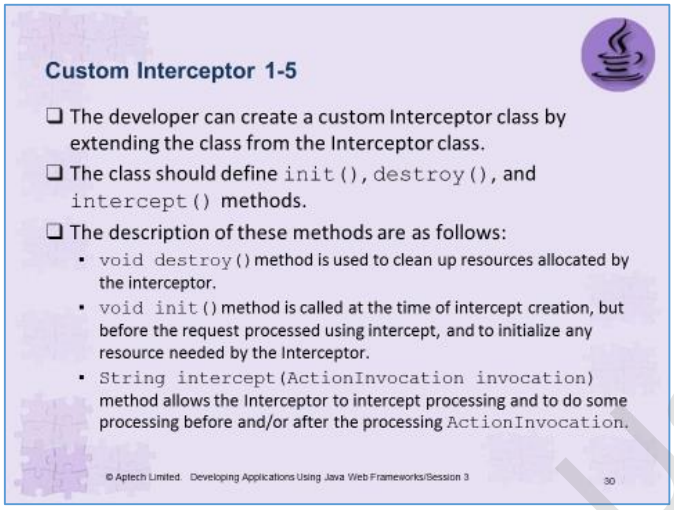
The following code snippet shows how to apply interceptor stacks to the action:

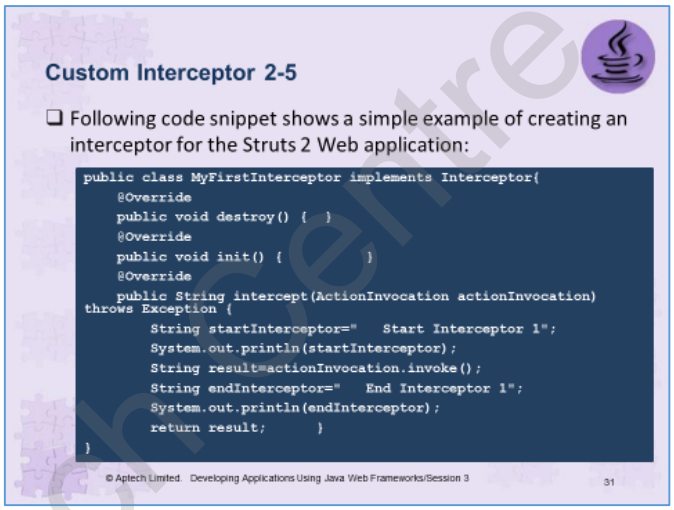
```
<action name="newyear" class="com. struts2.MyAction">
   <interceptor-ref name="mainStack"/>
   <result>view.jsp</result>
</action
```

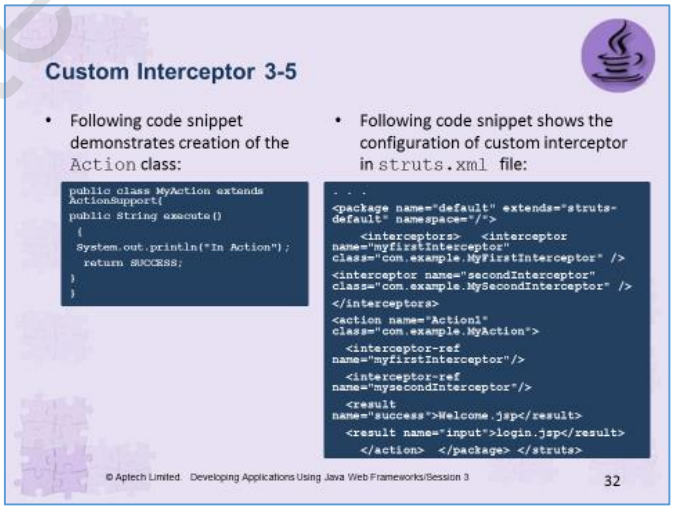
The registration of '`mainStack`' will register complete stake of all the six interceptors with '`newyear`' action. Interceptors will be executed in the order, in which they have been configured.

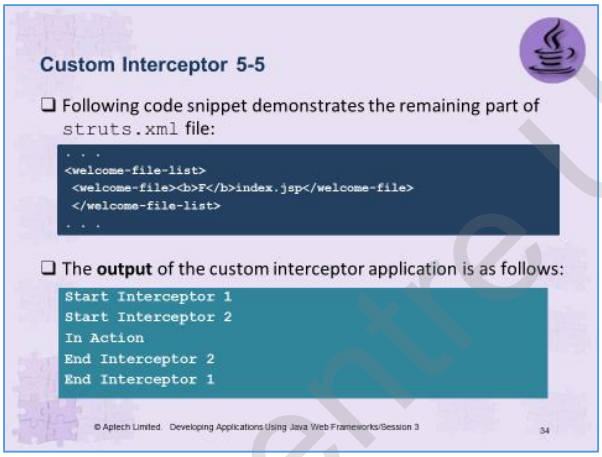
## Slides 30 to 34

Let us understand Custom interceptor.




Use slides 30 to 34 to explain custom interceptor.

The developer can create a custom Interceptor class by implementing the Interceptor interface. The class should define the following three methods: `init()`, `destroy()`, and `intercept()`. The `init()` and `destroy()` methods are used for initializing the interceptor and cleaning of the interceptor respectively. They are called once when the Struts 2 framework is initializing and when the framework is shutting down respectively. Interceptors are used across requests and needs to be thread-safe specially the `intercept()` method.

The code snippet as shown on slide 31 explains on creating an interceptor for the Struts 2 Web application. The code snippet defines three methods. The `init()` method will be invoked only once and will initialize the interceptor. The `intercept()` method defines the processing logic to be invoked at each request. Here, it returns `String` value, so the result View page will be invoked. The `intercept()` method can also return `invoke()` method which will result in the invocation of next interceptor configured in the action. Finally, the `destroy()` method is invoked. The `destroy()` method is invoked only once and destroys the interceptor.

Then, explain the code snippet as shown on slide 32 to create the Action class with the `execute()` method. Then, show them the configuration of custom interceptor as explained on slide 32. Finally, slide 33 shows the `index.jsp` page that call the appropriate action.

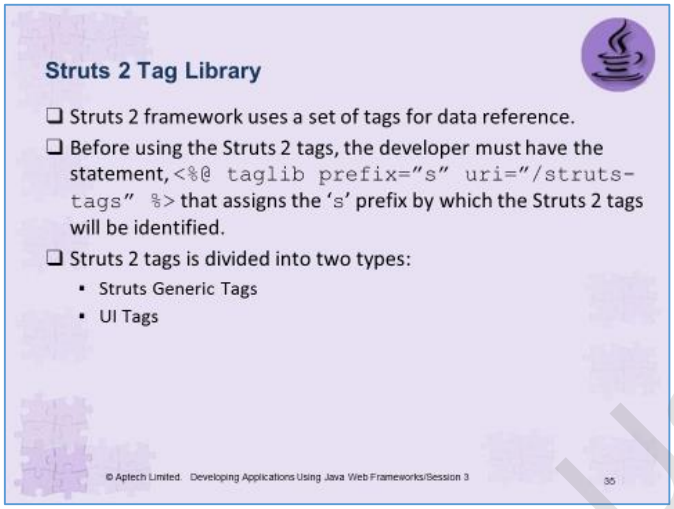
The output of the code is shown on slide 34.

**Tips:**

You can refer the following link to get additional example on configuring custom interceptor: http://www.javatpoint.com/struts-2-custom-interceptor-example-tutorial

## Slide 35

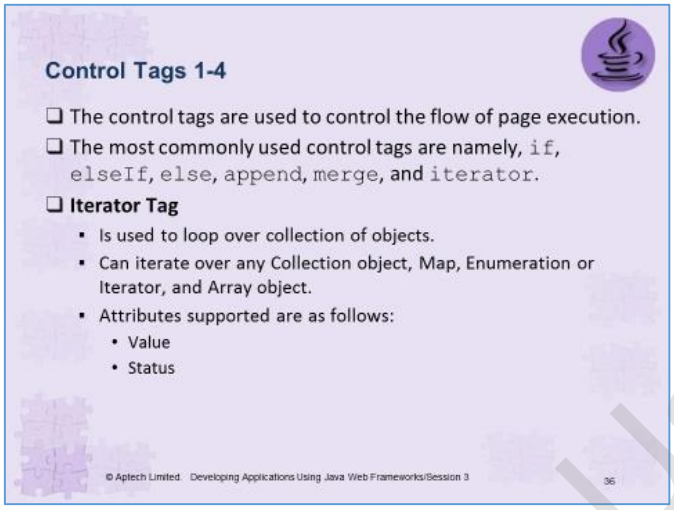
Let us understand Struts 2 tag library.

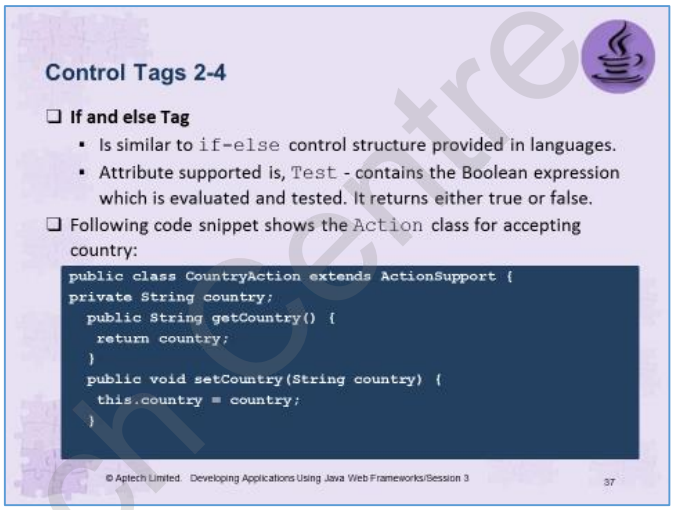



Use slide 35 to explain Struts 2 tag library. Tell the students that Generic tags are used for controlling the execution flow when the pages are rendered. Generic tags allows data extraction from places other than user's action or the value stack, such as Localization, JavaBeans, and including additional URLs or action executions.

## Slides 36 to 39

Let us understand Control tags.





Use slides 36 to 39 to explain control tags.

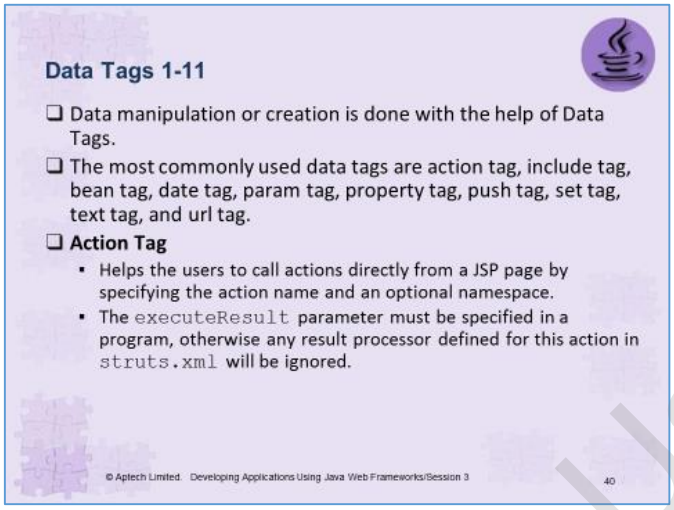
Other tags comes under Control Tags are subset, sort, and generator tags.

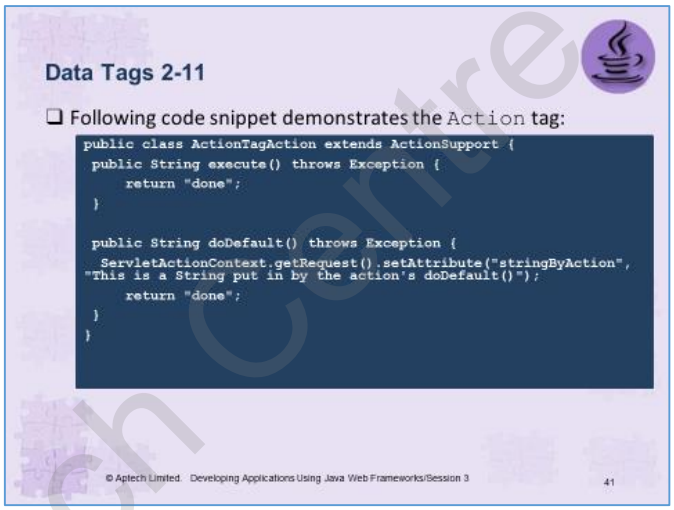
- **Subset tag** - Takes an iterator and outputs a subset of it. It delegates to `org.apache.struts2.util.SubsetIteratorFilter` internally to perform the subset functionality.
- **Sort tag** - Sorts a list using a comparator both passed in as the tag attribute. For example, if '`var`' attribute is specified, the sorted list will be placed into the `PageContext` attribute using the key specified by '`var`'.
- **Generator tag** - Generates an iterator based on the value attribute supplied. The following code snippet shows an example for generator tag:

```
<s:generator val="%{'111,222,333,444,555'}">
<s:iterator>
<s:property /><br/>
</s:iterator>
</s:generator>
```

## Slides 40 to 50

Let us understand data tags.




## Data Tags 4-11

❑ Following code snippet demonstrates the `actiontag.jsp`:

```
. . .
<s:action name="actionTagAction" executeResult="true" />
<div> … </div>
. . .
<s:action name="actionTagAction!specialMethod" executeResult="true">
<div> … </div>
. . .
<div> … </div>
<s:action name="actionTagAction!default" executeResult="false" />
<s:property value="#attr.stringByAction" />
```

❑ The jsp page invokes two methods namely, `execute()` and `doDefault()`.



## Data Tags 5-11

❑ **Include Tag**

- Is used to include a JSP file in another JSP page.

❑ Following code snippet demonstrates the use of `Include` tag:

```
 ...
<-- First Syntax -->
<include value="AptechJsp.jsp">
<-- Second Syntax -->
<include value="AptechJsp.jsp">
<param name="param1" value="value2">
<param name="param2" value="value2">
    </include>
<-- Third Syntax -->
<include value="AptechJsp.jsp">
<param name="param1">value1</param>
<param name="param2">value2</param>
</include>
...
```



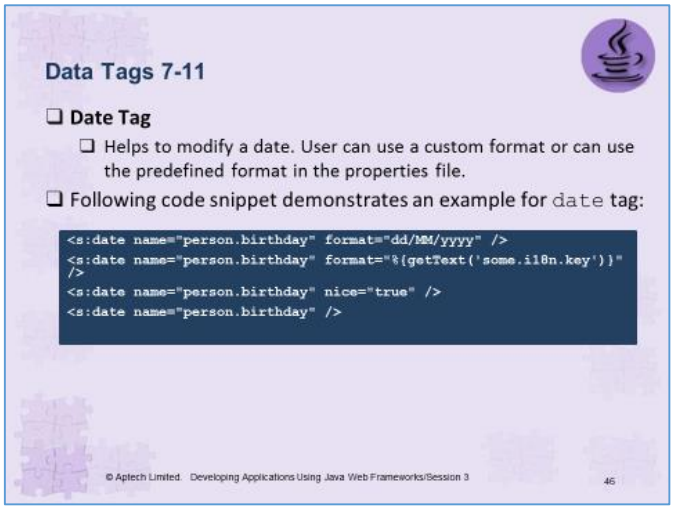
## Data Tags 6-11

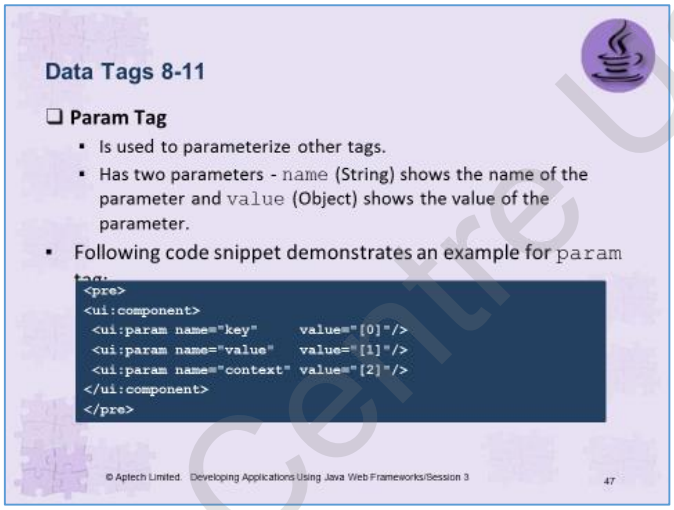
❑ **Bean Tag**

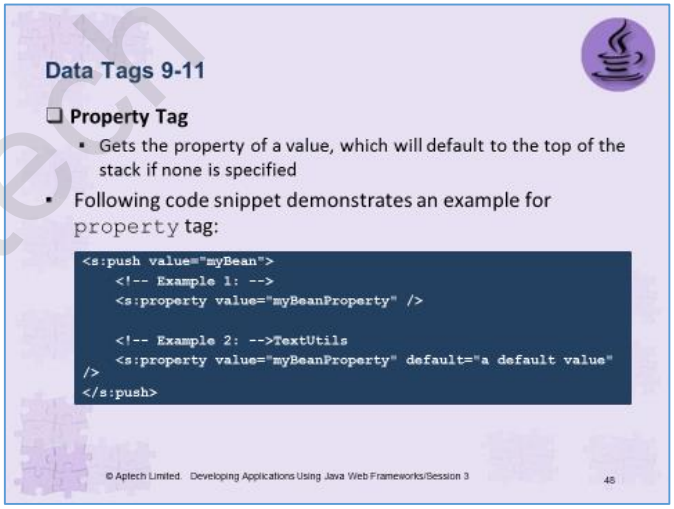
- ❑ Represents a class that applies to JavaBeans specification.

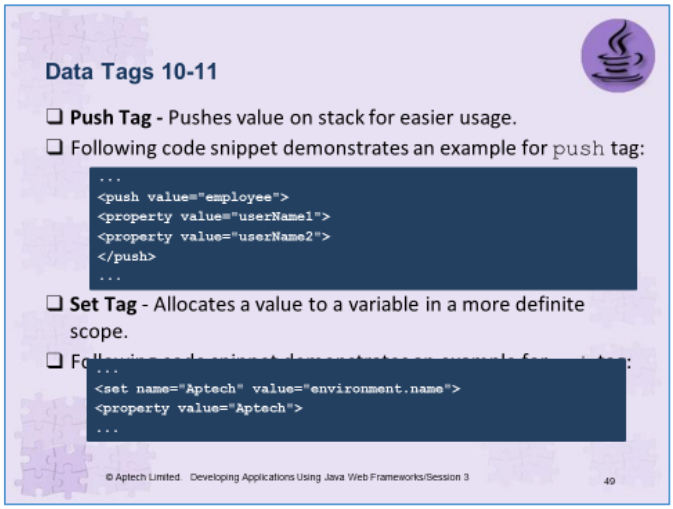
❑ Following code snippet demonstrates an example for `bean` tag:

```
...
<bean name="org.apache.struts2.util.number" var="number">
<param name="first" value="100">
<param name="last" value="125">
</bean>
...
```







Use slides 40 to 50 to explain data tags. Other types of data tags are as follows:

- `debug` tag - Outputs the content of the ValueStack.
- `i18n` tag - Gets a resource bundle and place it on the ValueStack. It allows the text tag to access messages from any bundle, and also the bundle associated with the current action.
- `a` tag - Creates an HTML `<a>`. It supports the same attributes.

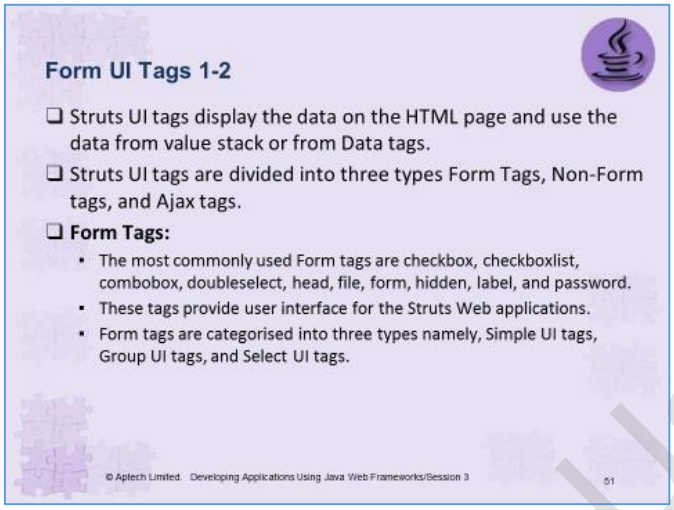
**Tips:**

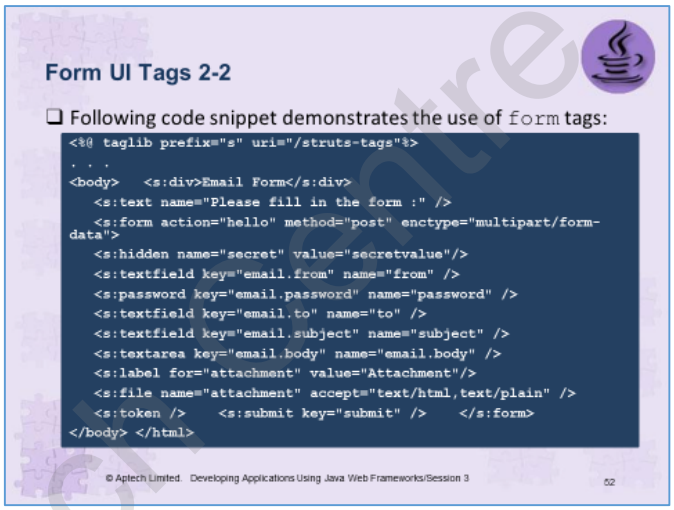
The following link shows the data tags with additional examples:

http://www.journaldev.com/2230/struts-2-data-tags-example-tutorial

## Slides 51 and 52

Let us understand Form UI tags.




Use slides 51 and 52 to explain Form UI tags.

Tell the students that the UI tags help in the rendering of the user interface required for the Struts based Web applications. Form tags are categorized into Simple UI tags, Group UI tags, and Select UI tags. Group UI tags helps to create radio button and checkbox.

Some of the Form tags are listed:

- `head` tag - Provides parts of the HEAD section for an HTML file.
- `file` tag - Provides an HTML file input element.
- form tag - Provides HTML an input form.
- `hidden` tag - Provides an HTML input element of type hidden, populated by the specified property from the ValueStack.

- `password` tag - Provides an HTML input tag of type password.
- `Checkbox` tag - Provides an HTML input element of type checkbox, populated by the specified property from the ValueStack.
- `radio` tag - Provides a radio button input field.
- `textfield` tag - Provides an HTML input field of type text.

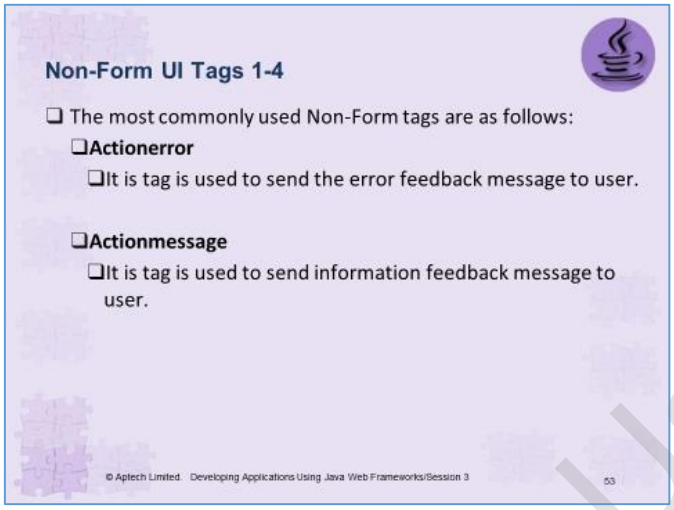
**Tips:**

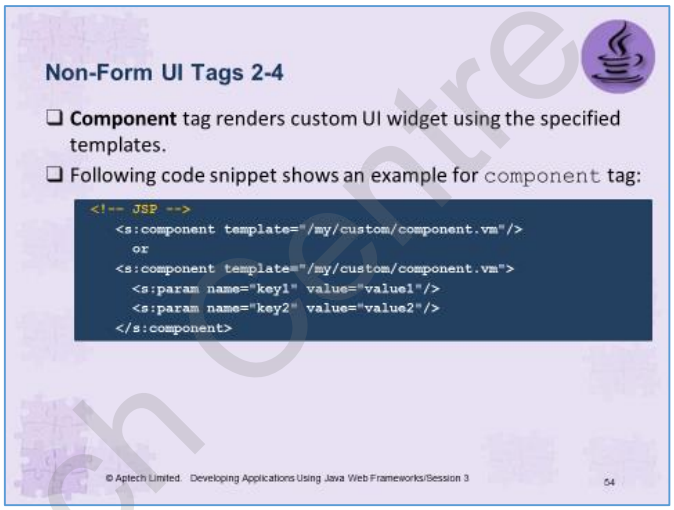
The following link shows the UI tags examples:

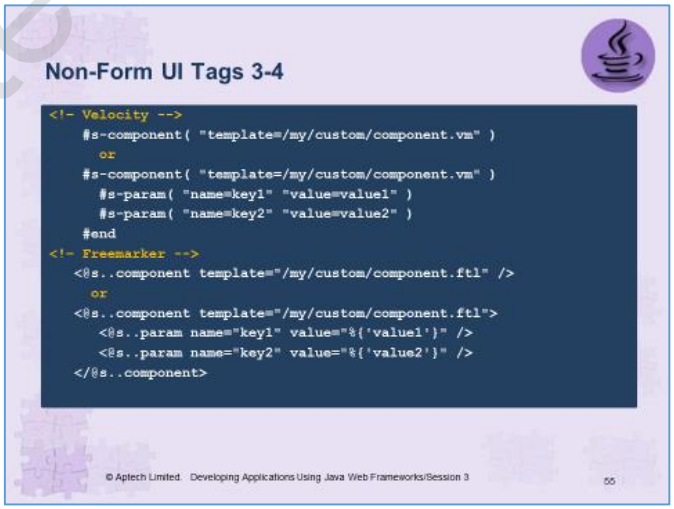
http://www.journaldev.com/2266/struts2-ui-tags-example-tutorial

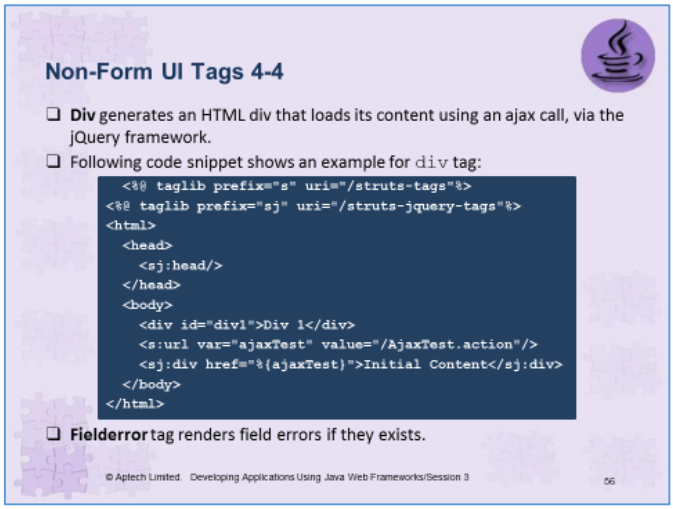
## Slides 53 to 56

Let us understand Non-form UI tags.



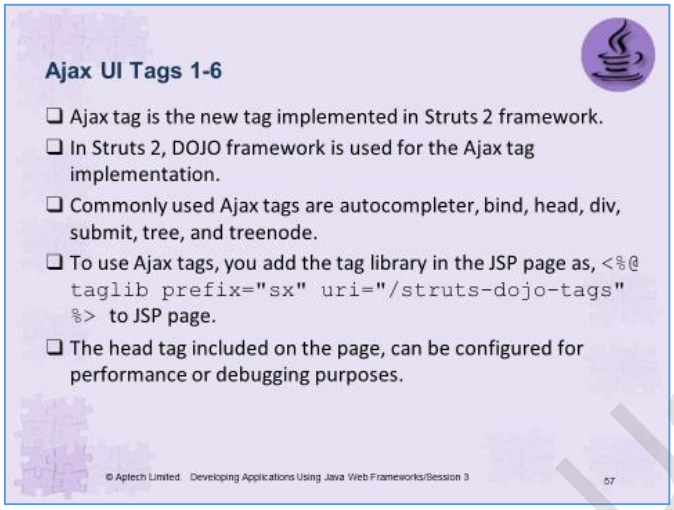


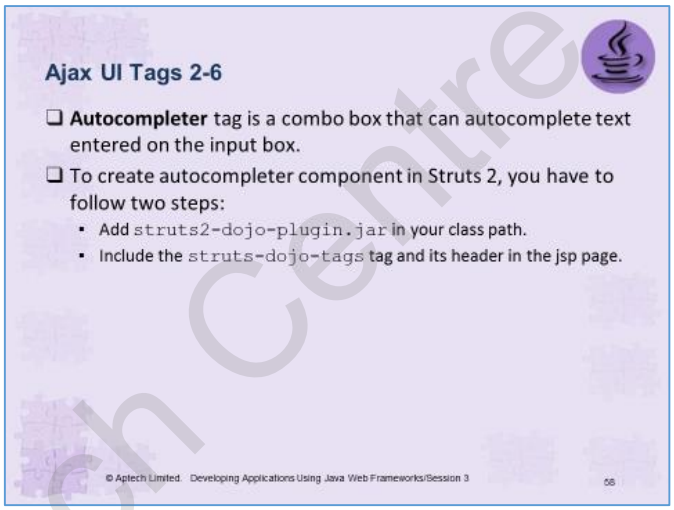

Use slides 53 to 56 to explain Non-form UI tags.

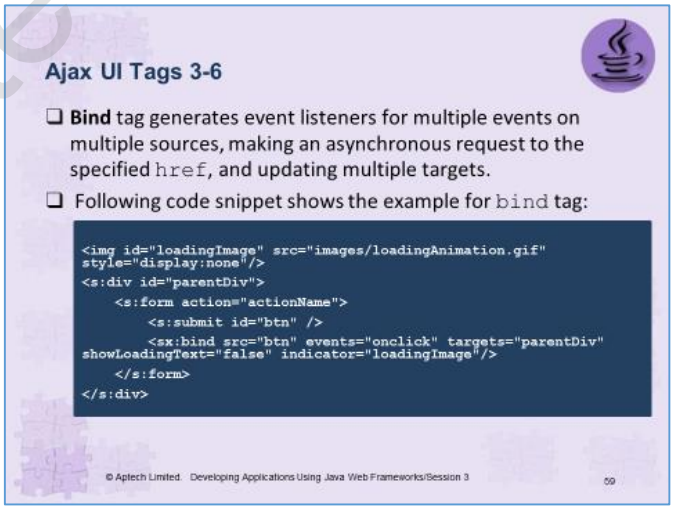
Explain the student about non-form UI tags such as `actionerror`, `actionmessage`, `component`, `div`, and `fielderror`.

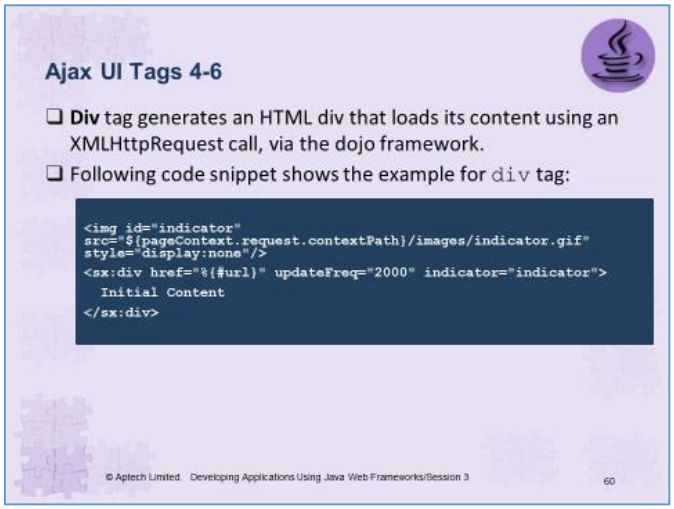
## Slides 57 to 62

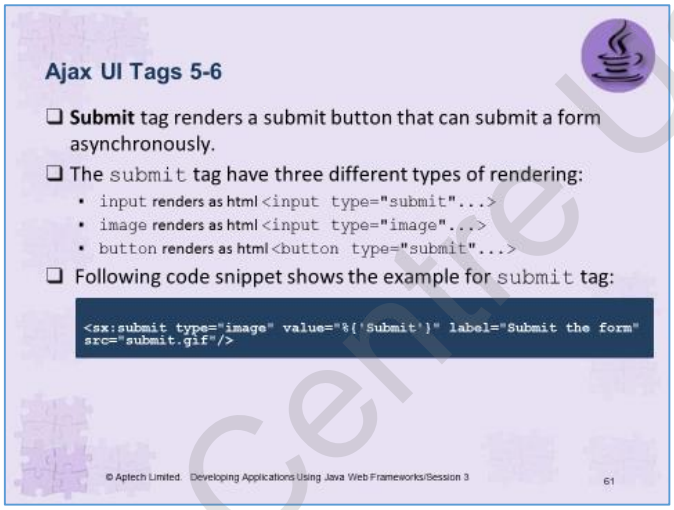
Let us understand Ajax UI tags.

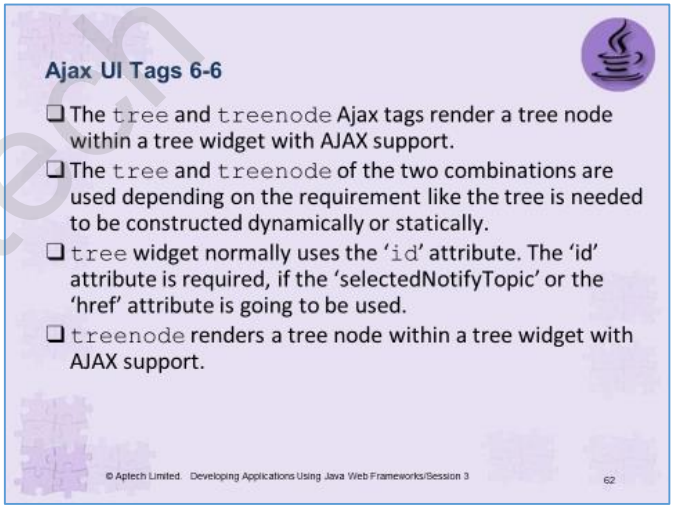





Use slides 57 to 62 to explain Ajax UI tags.

Ajax tag is the new tag implemented in Struts 2 framework. In Struts 2, DOJO framework is used for the Ajax tag implementation. Commonly used Ajax tags are `autocompleter`, `bind`, `head`, `div`, `submit`, `tree`, and `treenode`.

To use Ajax tags, you add the tag library in the JSP page as, `<%@ taglib prefix="sx" uri="/struts-dojo-tags" %>` to JSP page.

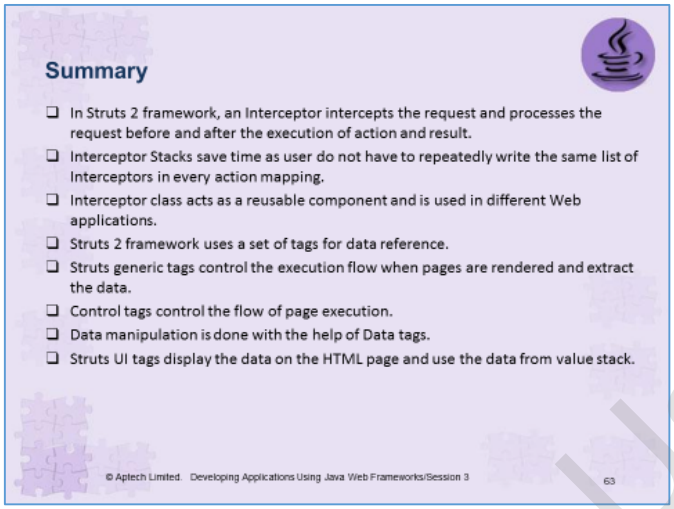
Then, explain the different AJAX tags as shown on slides 58 to 62.

The following is the additional code snippet for AJAX tags:

```
<%@ page contentType="text/html; charset=UTF-8"%>
<%@ taglib prefix="s" uri="/struts-tags"%>
<%@ taglib prefix="sx" uri="/struts-dojo-tags"%>
<html>
. . .
<s:head />
<sx:head />
</head>
<body>
   <s:form>
      <sx:autocompleter label="Favourite Colour"
         list="{'red','green','blue'}" />
      <br />
      <sx:datetimepicker name="deliverydate" label="Delivery
Date"
         displayFormat="dd/MM/yyyy" />
      <br />
      <s:url id="url" value="/hello.action" />
      <sx:div href="%{#url}" delay="2000">
           Initial Content
      </sx:div>
      <br/>
      <sx:tabbedpanel id="tabContainer">
        <sx:div label="Tab 1">Tab 1</sx:div>
        <sx:div label="Tab 2">Tab 2</sx:div>
      </sx:tabbedpanel>
   </s:form>
</body>
</html>
```

## Slide 63

Let us summarize the session.




In slide 63, you will summarize the session. You will end the session, with a brief summary of what has been taught in the session.

### 3.3 Post Class Activities for Faculty

You should familiarize yourself with the topics of the next session. You should also explore the OGNL expression language, internationalization, and Validation API in Struts 2.

**Tips:**

You can also check the Articles/Blogs/Expert Videos uploaded on the OnlineVarsity site to gain additional information related to the topics covered in the next session. You can also connect to online tutors on the OnlineVarsity site to ask queries related to the sessions.

# Session 4 – Struts 2 - OGNL, Validation, and Internationalization

## 4.1 Pre-Class Activities

Before you commence the session, you should familiarize yourself with the topics of this session in-depth. You should revisit topics of the previous session for a brief review.

Here, you can ask students the key topics they can recall from previous session. Prepare a question or two which will be a key point to relate the current session objectives.

### 4.1.1 Objectives

By the end of this session, the learners will be able to:

- Explain the use of ValueStack
- Explain OGNL expression language
- Explain the flow of data in and out of the Struts 2 framework
- Explain the use of converters and their types
- Explain Validator framework in Struts 2
- Explain different types of validators present in the Validator framework
- Explain the implementation of validation framework in Struts 2
- Explain internationalization
- Explain the use of i18N interceptor and resource bundle in internationalization
- Explain how to create the Struts 2 Web application with internationalization

### 4.1.2 Teaching Skills

To teach this session successfully, you should be aware of OGNL expression language and the flow of data in and out of the Struts 2 framework.

Aware yourself with the use of converters and there types in Struts 2. Also, familiarize yourself with different types of validators present in the Validator framework and the implementation of validation framework in Struts 2. You should also know about internationalization and the use of i18N interceptor and resource bundle in internationalization. Aware yourself with the creation of Struts 2 Web application with internationalization.

For teaching in the class, you are expected to use slides and LCD projectors.

**Tips:**

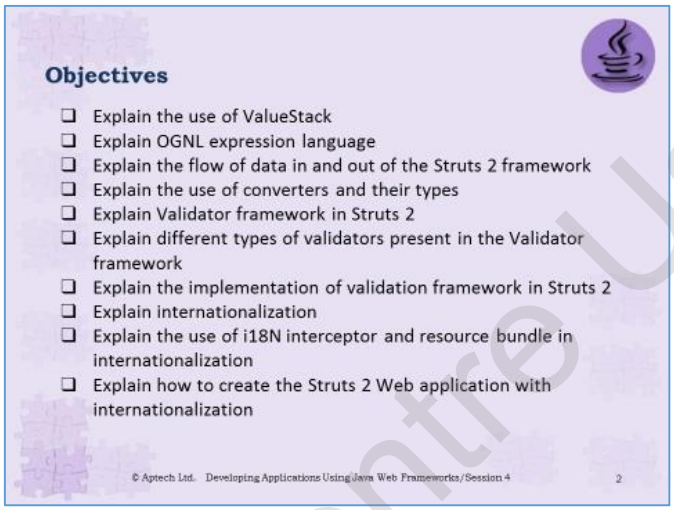
It is recommended that you test the understanding of the students by asking questions in between the class.

**In-Class Activities:**

Follow the order given here during In-Class activities.

**Overview of the Session:**

Give the students a brief overview of the current session in the form of session objectives.
Show the students slide 2 of the presentation.



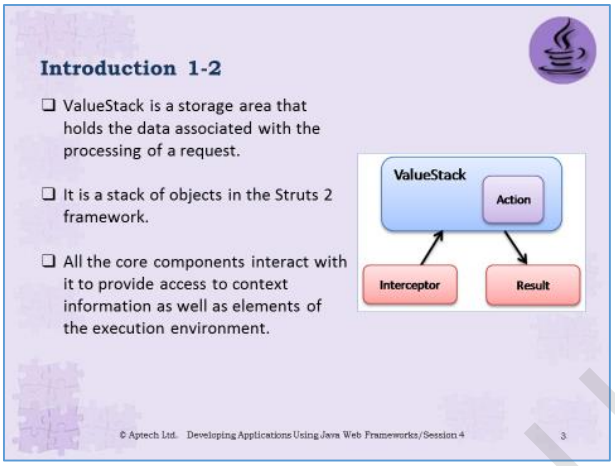

Tell the students that they will be introduced to OGNL expression language and the flow of data in and out of the Struts 2 framework.

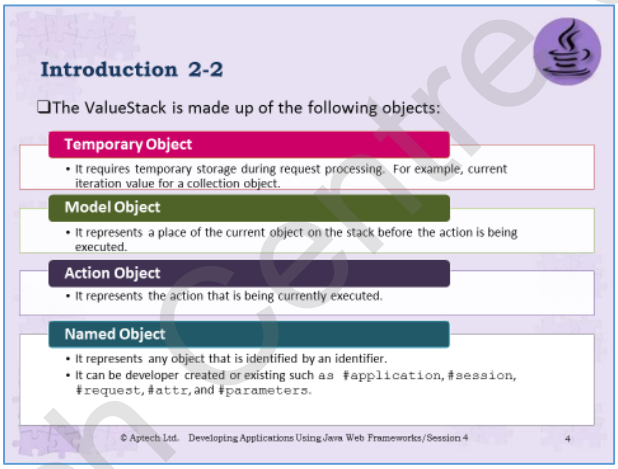
The session explains the use of converters, their types, and about Validator framework in Struts 2. In this session, they will learn different types of validators present in the Validator framework and the implementation of validation framework in Struts 2. Finally, the session explains about internationalization and the use of i18N interceptor and resource bundle in internationalization.

## 4.2 In-Class Explanations

### Slides 3 and 4

Let us introduce ValueStack.




Use slides 3 and 4 to introduce ValueStack.

ValueStack is a storage area that holds the data associated with the processing of a request. It is a stack of objects in the Struts 2 framework. All the core components interact with it to provide access to context information as well as elements of the execution environment.

Tell them ValueSatck stores different types of objects such as:

1. Temporary objects - Are used to store values of temporary variables used during execution of a page. For example, value of iteration variable used for looping over a collection.
2. Model objects - You can create Model objects to be used by action class.
3. Action objects - Represents the currently executed action class.

4. Named objects – Represents the object that is identified by an identifier. It can be developer created or existing such as `#application`, `#session`, `#request`, `#attr`, and `#parameters`. Each of these existing named objects corresponds to an equivalent HTTP scoped object collection.

Tell them that they can access objects from the ValueStack using JSP tags. A ValueStack returns the value of a property in the order it stacks the property. In other words, when the ValueStack is queried for a property value, it returns the value provided the property exists in the stack.

**Tips:**

Tell the students that they can get ValueStack object inside the action class using the following statement:
`ActionContext.getContext().getValueStack()`

Some of the methods to work with the ValueStack objects are as follows:

- `Object findValue(String expr)` – Finds a value by evaluating the given expression against the stack in the default search order.
- `CompoundRoot getRoot()` – Get the `CompoundRoot` which holds the objects pushed onto the stack.
- `Object peek()` – Get the object on the top of the stack without changing the stack.
- `Object pop()` – Get the object on the top of the stack and remove it from the stack.
- `void push(Object o)` – Puts the object onto the top of the stack.
- `void set(String key, Object o)` – Sets an object on the stack with the given key so it is retrievable by findValue(key k) method.
- `void setDefaultType(Class defaultType)` – Sets the default type to convert to if no type is provided when getting a value.
- `void setValue(String expr, Object value)` – Attempts to set a property on a bean in the stack with the given expression using the default search order.
- `int size()` – Gets the number of objects in the stack.

**In-Class Question:**

After you finish explaining ValueStack, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.

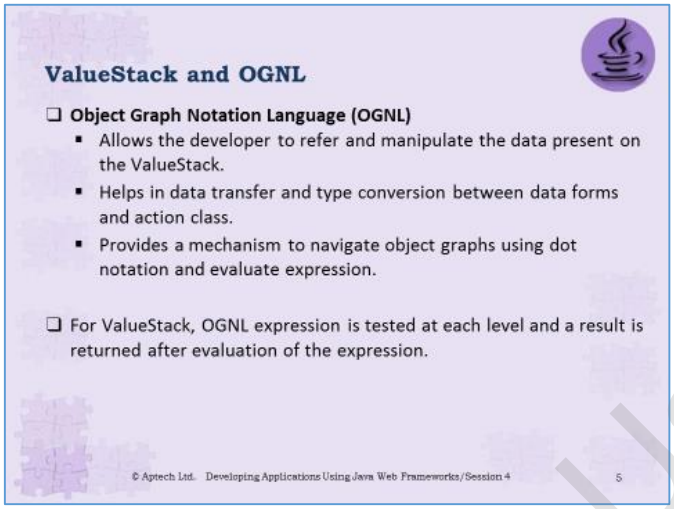
How to access the objects from ValueStack in a JSP page?

**Answer:**

JSP tags

## Slide 5

Let us understand ValueStack and OGNL.




Use slide 5 to explain ValueStack and OGNL.

Tell them that in ValueStack, you search for or evaluate a particular expression using Object Graph Navigation Language (OGNL) syntax. OGNL is a powerful expression language that allows assigning and manipulating the data present on the ValueStack.

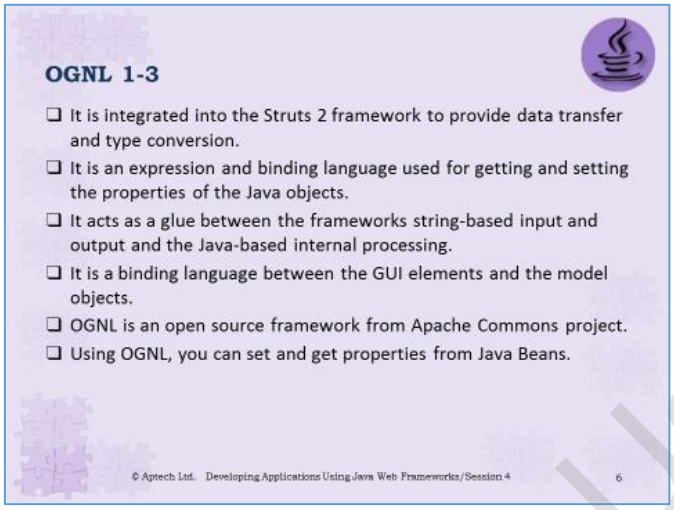
Tell the students that OGNL simplifies the accessibility of data stored in the `ActionContext`. ValueStack is set as the root object of OGNL in Struts 2.0 framework.

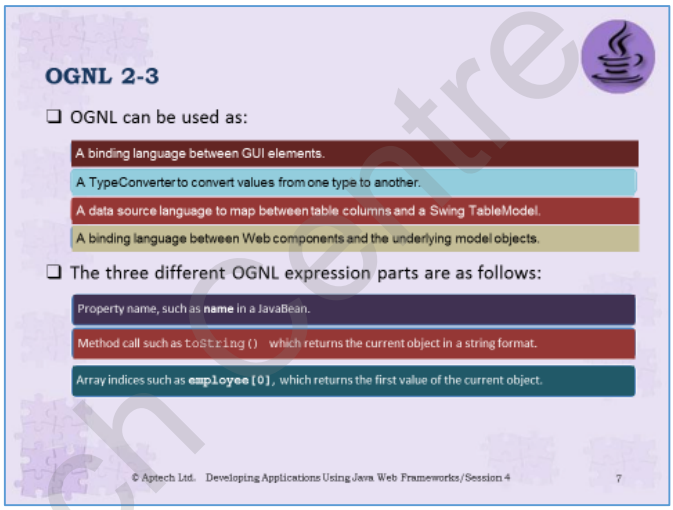
OGNL and ValueStack works together to handle a request. OGNL provides the Expression Language (EL), which can be a part of form tags and UI pages. EL maps the entered values to the destination properties of the Java Bean to which they have to be set.

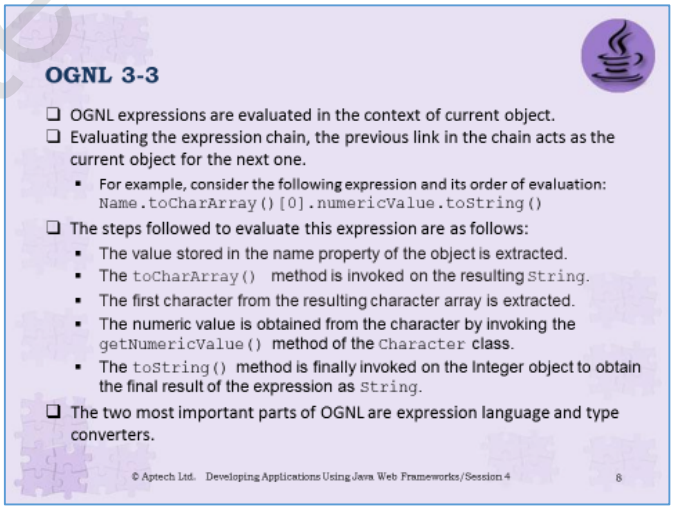
When a form is submitted, the values are transmitted as Strings via HTTP Protocol. String values is converted to the respective primitive or custom java types to set them in the beans. During the process of setting the values, OGNL consults the type converters available in Struts 2 and converts form values to the destination types and will set into the Java Beans.

## Slides 6 to 8

Let us understand OGNL.




Use slides 6 to 8 to explain OGNL.

Tell the students that during the request handling process in Struts 2, OGNL helps in three areas.

1. A binding language between GUI elements such as text fields, check boxes, radio button, and so on to model objects.
2. A Type Converter to convert values from one type to another. For example, from String to numeric.
3. A data source language to map between table columns and a Swing TableModel.
4. A binding language between Web components and the underlying model objects.

It does this by moving data from the request parameter into the action's JavaBean properties and back from action's Java bean properties out into the rendering HTML pages.

Tell them that the ValueStack is a set of several objects. However, for OGNL, it appears to be a single object.

**Working of OGNL and ValueStack**

OGNL and ValueStack work together in Struts 2 to handle a request. OGNL offers the expression language which can be part of form tags and UI pages. This expression language helps to map the entered values to the destination properties of the Java Bean to which they have to be set.

When a form is submitted, the values are transmitted as Strings through the HTTP Protocol. These String values have to be converted to the respective primitive or custom Java types to set them in the beans. During the process of setting the values, OGNL consults the type converters available in Struts 2 and helps in conversion of the form values to the destination types and will set into the JavaBeans.

Without the type converters, developers have to write the mapping beans which get the values from the form beans and then convert them to their appropriate data types. This leads to more code to handle a request and also puts more burden to test the code.

OGNL looks for the properties in the objects that are loaded in the ValueStack and when found, the values are converted and set to Model objects. This feature of Struts 2 is implemented by OGNL Expression Language (EL).

OGNL EL can refer many of the built-in objects present in a JSP page. It is the default expression language that is used to refer to data from various sections of the framework in a consistent manner. Struts 2 follows the MVC design pattern. The View component, which is responsible for displaying the model and other object in a JSP page, uses OGNL EL.

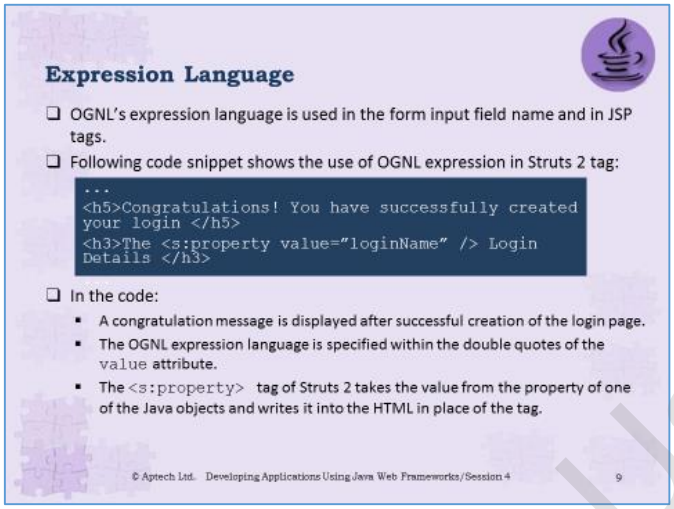
Following are the three different OGNL expression parts:

- Property name, such as name in a JavaBean.
- Method call such as `toString()` which returns the current object in a string format.
- Array indices such as `employee[0]`, which returns the first value of the current object.

Then, using slide 8, explain them that evaluation of OGNL expression.

## Slide 9

Let us understand EL.




Use slide 9 to explain EL.

Explain some of the examples of using OGNL EL as mentioned on slide 9.

Some of the additional EL notations are as follows:

- `<s:property value="#session.user.username" />` – Shows username property of the User object in the Session context.
- `<s:textfield name="username"/>` – A username property on the Value Stack.

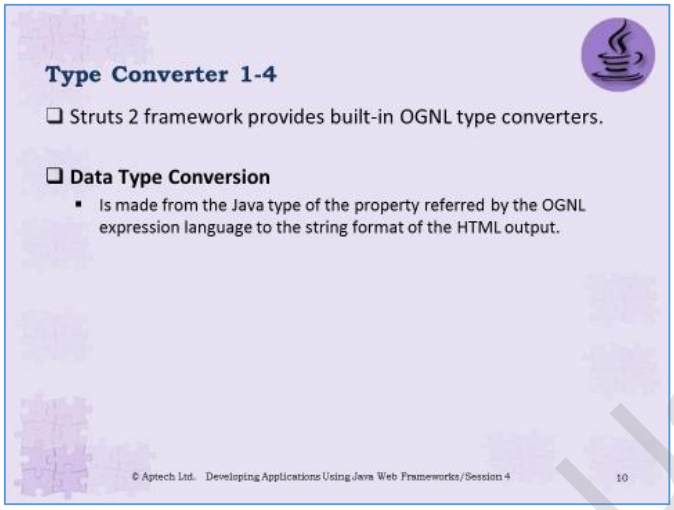
### Tips:

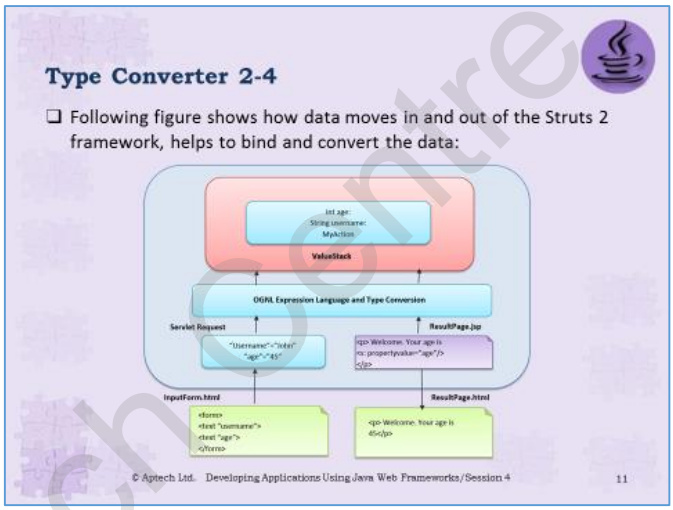
To work with few more examples on OGNL EL, you can refer to this link:

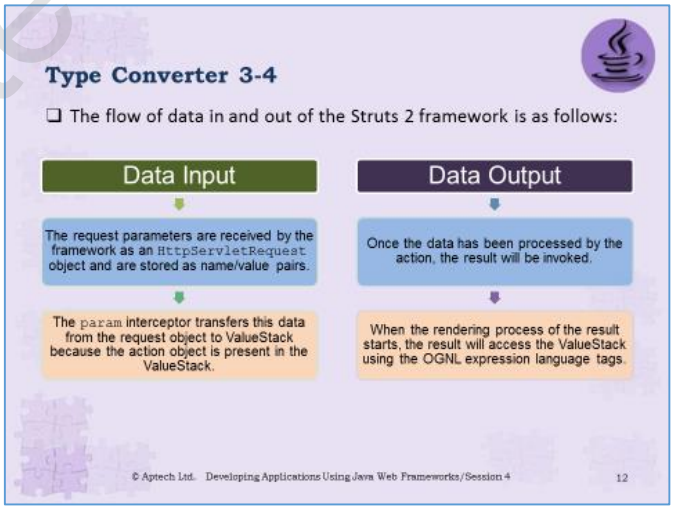
http://tutorials4u.net/struts2-tutorial/struts2_ognl_expression_language_example.html

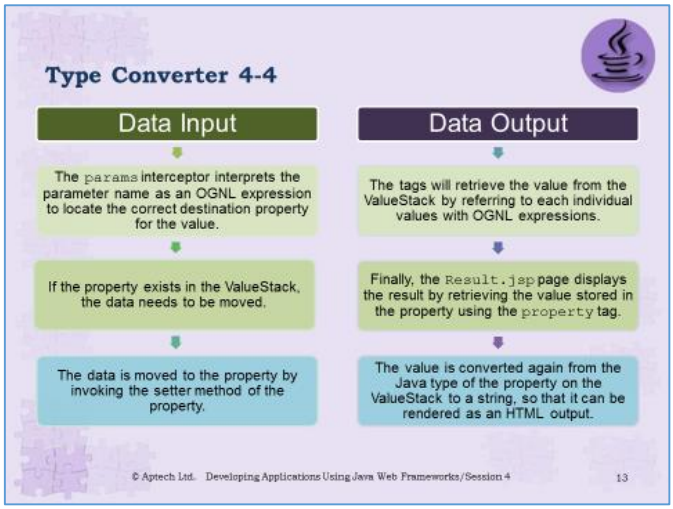
## Slides 10 to 13

Let us understand type converter.






Use slides 10 to 13 to explain type converter.

Tell them that even in a simple Web application, a data type conversion is made from the Java type of the property referred by the OGNL expression language to the string format of the HTML output.
Thus, always an appropriate conversion of data takes place when the data moves in and out of the Java environment and HTML. Struts 2 framework provides built-in OGNL type converters.

Then, explain the flow of data in and out of the Struts 2 framework using the figure given on slide 11. Explain the process of data in and out using the points given on slides 12 and 13.

**Data Input**

The form, `InputForm.html` displays two input fields, which are valid HTML expressions. The field names are valid OGNL expression. Once the value is entered, the form is submitted to the framework. The request parameters are received by the framework as an `HttpServletRequest` object and are stored as name/value pairs. The name/value pair exists in a string format.

The `param` interceptor transfers this data from the request object to ValueStack because the action object is present in the ValueStack. The most important activity that takes place before transferring the data is mapping of the property name with actual property on the ValueStack. The `params` interceptor interprets the parameter name as an OGNL expression to locate the correct destination property for the value.

The OGNL resolves its expression against a default object, which is a type of virtual object created by the ValueStack. ValueStack holds the objects and their properties. In other words, ValueStack exposes the properties of the object it contains. Therefore, when the expression age is evaluated as an OGNL expression against the ValueStack, it checks whether the ValueStack has a `user` property. If the property is present, then it checks

whether the property named user has an `age` property. Each of the properties is separated by a period. If the age property exists in the ValueStack, the data needs to be moved.

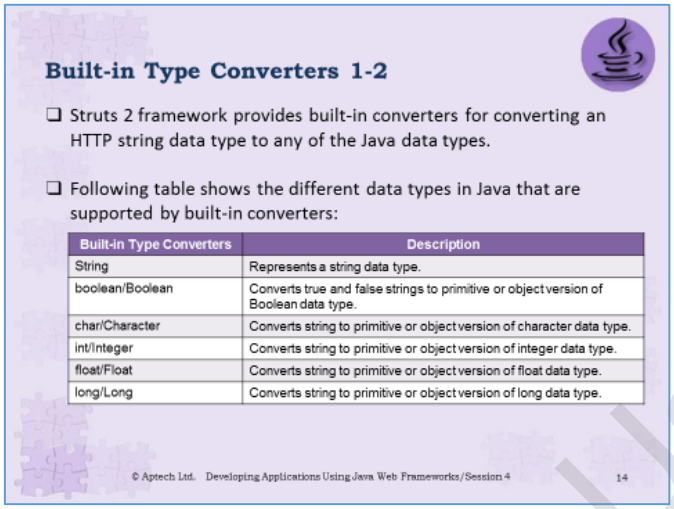
Thus, the data is moved to the property by invoking the setter method of the property. However, since the value is in string format, it needs to be converted to the appropriate Java data type, that is, int. In order to do this conversion, OGNL type converters come into action and check whether this type conversion, that is, string to `int` is possible. If it is possible, then the type converter converts and sets the value on the object named user.

**Data Out**

Once the data has been processed by the action, the result will be invoked, which will provide a new result view of the application to the user. During data processing of the request, the object remains on the ValueStack. When the rendering process of the result starts, the result will access the ValueStack using the OGNL expression language tags. The tags will retrieve the value from the ValueStack by referring to each individual values with OGNL expressions. Finally, the `Result.jsp` page displays the result by retrieving the value stored in the property using the property tag. The value is converted again from the Java type of the property on the ValueStack to a string, so that it can be rendered as an HTML output. Thus, the integer data type is converted back to String data type.

## Slides 14 and 15

Let us understand built-in type converters.

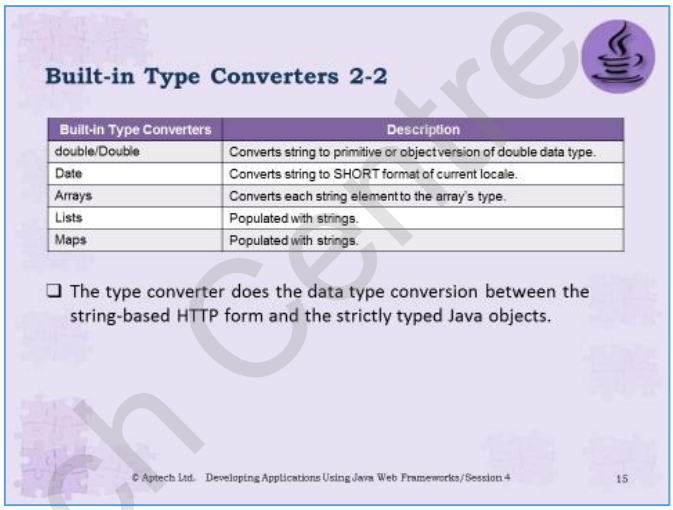


**Built-in Type Converters 1-2**

- Struts 2 framework provides built-in converters for converting an HTTP string data type to any of the Java data types.
- Following table shows the different data types in Java that are supported by built-in converters:

| Built-in Type Converters | Description |
|---|---|
| String | Represents a string data type. |
| boolean/Boolean | Converts true and false strings to primitive or object version of Boolean data type. |
| char/Character | Converts string to primitive or object version of character data type. |
| int/Integer | Converts string to primitive or object version of integer data type. |
| float/Float | Converts string to primitive or object version of float data type. |
| long/Long | Converts string to primitive or object version of long data type. |





**Built-in Type Converters 2-2**

| Built-in Type Converters | Description |
|---|---|
| double/Double | Converts string to primitive or object version of double data type. |
| Date | Converts string to SHORT format of current locale. |
| Arrays | Converts each string element to the array's type. |
| Lists | Populated with strings. |
| Maps | Populated with strings. |

- The type converter does the data type conversion between the string-based HTTP form and the strictly typed Java objects.



Use slides 14 and 15 to explain built-in type converters.

The different data types in Java that are supported by built-in converters are as follows:

- String - Represents a string data type.
- boolean/Boolean - Converts true and false strings to primitive or object version of Boolean data type.
- char/Character - Converts string to primitive or object version of character data type.
- int/Integer - Converts string to primitive or object version of integer data type.
- float/Float - Converts string to primitive or object version of float data type.
- long/Long - Converts string to primitive or object version of long data type.
- double/Double - Converts string to primitive or object version of double data type.

- Date - Converts string to SHORT format of current locale.
- Arrays - Converts each string element to the array's type.
- Lists and Maps - Populated with strings.

Consider an example where the framework will look for a converter of a particular type when it locates a Java property targeted by a given OGNL expression. The setter method provided in the action class can change from `setId(String Id) to setId(int Id).`

The type converter does the data type conversion between the string-based HTTP form and the strictly typed Java objects.

**In-Class Question:**

After you finish explaining built-in type converters, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.

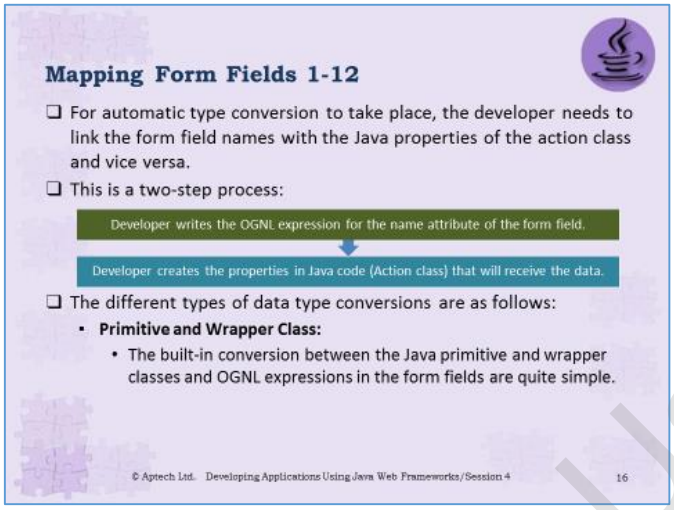
Which converter is used for converting an HTTP string data type to any of the Java data types?

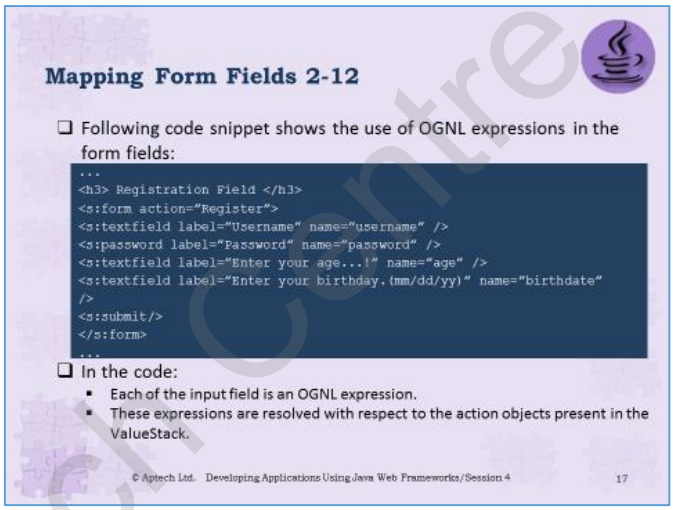
**Answer:**

Built-in type converters.

## Slides 16 to 27

Let us understand how to map form fields.




## Mapping Form Fields 4-12

```
this.password = password;
}
public String getUsername() {  return username;
public void setUsername(String username) {  this.username = username;
}
public Double getAge() { return age;
}
public void setAge(Double age) {  this.age = age;
}
public Date getBirthdate() {  return birthday;
}
public void setBirthdate(Date birthday) {  this.birthday = birthday;
}
}
...
```

- In the code:
  - When the **Submit** button on the form is clicked, the request is sent to the framework as a name-value pair.
  - The framework places the `Action` object in the ValueStack.



## Mapping Form Fields 5-12

- **Handling Multiple Values:**
  - Multiple parameter values coming from a request can be mapped to a single parameter name.
  - These multiple values can be mapped to Arrays, Lists, or Maps.

| Array | Lists | Maps |
|---|---|---|
| • Struts 2 provides support for converting data to Java arrays.<br>• OGNL executes the type conversion for each element of the array.<br>• The framework provides unique names to OGNL, and are correctly interpreted as specific elements in an array. | • Struts 2 framework supports the conversion of sets of request parameter to properties of various collection types such as Lists.<br>• Lists need not be initialized.<br>• Elements in a List will be treated as String objects if the type is not specified. | • Struts 2 also support the conversion of a set of values from the request parameter to a Map property.<br>• Map stores the value in a key-value pair.<br>• The data types can be specified for the key-value pair in the Map. |



## Mapping Form Fields 6-12

- Following code snippet shows the use of the array in the struts form:

```
...
<s:form action="ArraysDataTransferTest">
<s:textfield name="marks" label="Marks"/>
<s:textfield name="marks" label="Marks"/>
<s:textfield name="marks" label="Marks"/>
<s:textfield name="names[0]" label="names"/>
<s:textfield name="names[1]" label="names"/>
<s:textfield name="names[2]" label="names"/>
<s:submit/> </s:form>
...
```

- In the code:
  - There are two arrays named, **marks** and **names**.
  - The two array properties named, **marks** and **names** will receive data from the first and the last three form fields respectively.

## Mapping Form Fields 7-12

❑ Following table displays the request parameter name with its value:

| RequestParameter Name | Parameter Value |
|---|---|
| Marks | 75, 65, 55 |
| name[0] | Angel |
| name[1] | Jessica |
| name[2] | John |



## Mapping Form Fields 8-12

❑ Following code snippet shows the implementation of the properties in the action class that will receive the data:

```
...
private int[] marks ;
public int[] getMarks() { return marks;
}
public void setMarks(int[] marks){ this.marks = marks; }
private String[] names = new String[10];
public String[] getNames() { return names;
}
public void setNames(String[] names){ this.names = names; }
...
```

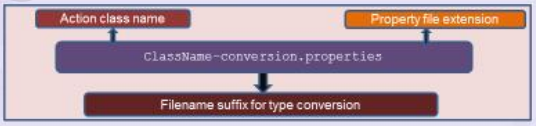
❑ In the code:

- Arrays are displayed using the getter and setter methods.
- The framework, when it transfers the **marks** property, first, searches for the **marks** property and resolves the mapping of the property with the request parameter.



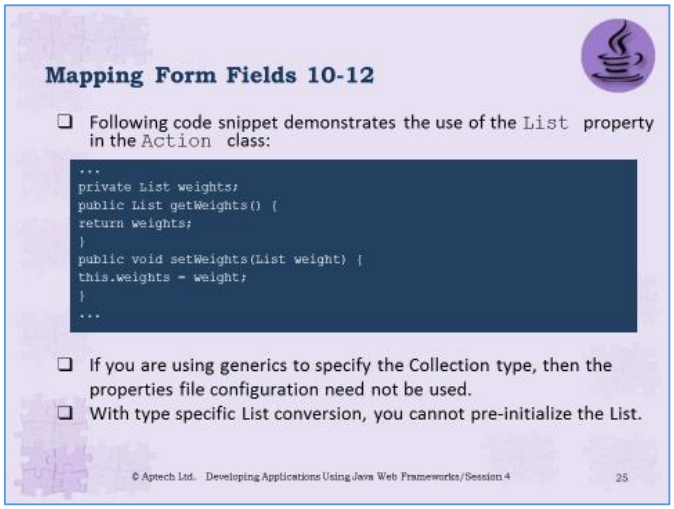
## Mapping Form Fields 9-12

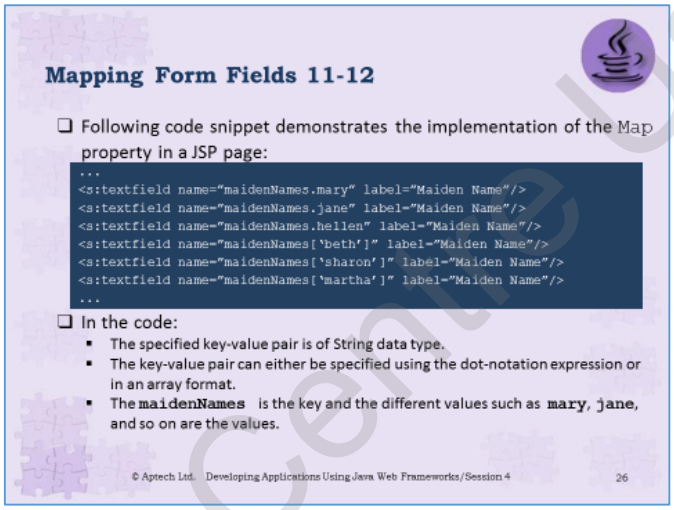
❑ Following figure displays the creation of a properties file according to the naming convention:

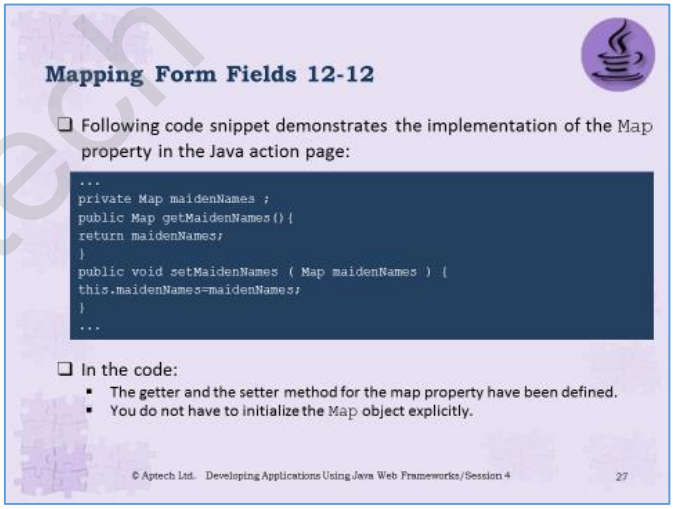



❑ Following code snippet shows the JSP page containing the weights property, which will be used as a List in the action class:

```
...
<s:textfield name="weights[0]" label="weights"/>
<s:textfield name="weights[1]" label="weights"/>
<s:textfield name="weights[2]" label="weights"/>
...
```



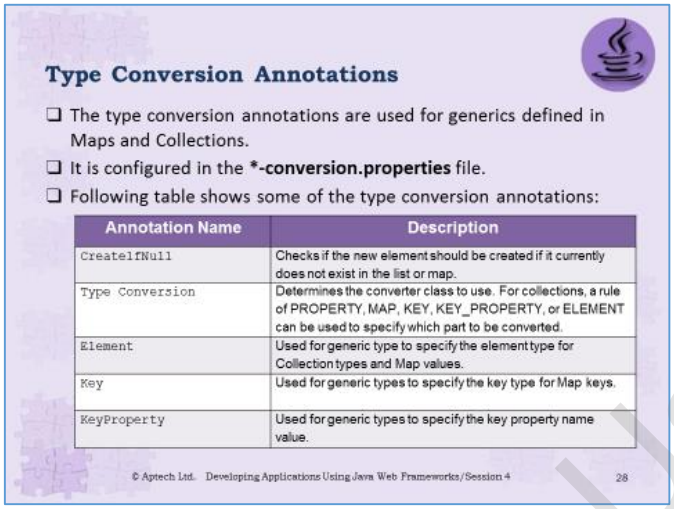




Use slides 16 to 27 to explain how to map form fields.

Discuss examples provided on the slides for conversion in Struts 2.

## Slide 28

Let us understand type conversion annotations.




Use slide 28 to explain type conversion annotations. Conversion annotation is also a type conversion annotation.

Tell them that to use annotation-based type conversion, annotate the class or interface with the **Conversion Annotation**.

Conversion Annotation – Is a marker annotation for type conversions at Type level. The conversion annotation is used at Type level. The code snippet for conversion annotation is as follows:

```
@Conversion()
   public class ConversionAction implements Action {
}
```

Then, explain the `TypeConversion` annotation. This annotation is used for class and application wide conversion rules. It can be applied at property as well as method level. It has an attribute named `type` which takes an enum value. The value of the attribute determines whether the conversion should be applied at application level or class level. The enum constants are `ConversionType.CLASS` and `ConversionType.APPLICATION`. It also has another attribute named `rule`. The `rule` attribute takes a property from an enum. It specifies the conversion rule which can be a property, a collection, or a map.

The following code snippet shows the use of `TypeConversion` annotation:

```
@Conversion()
public class ConversionAction implements Action {
 private String convertInt;

  private String convertDouble;
  private List users = null;

  private HashMap keyValues = null;

  @TypeConversion(type = ConversionType.APPLICATION, converter
= "com.opensymphony.xwork2.util.XWorkBasicConverter")
  public void setConvertInt( String convertInt ) {
      this.convertInt = convertInt;
  }

  @TypeConversion(converter =
"com.opensymphony.xwork2.util.XWorkBasicConverter")
  public void setConvertDouble( String convertDouble ) {
      this.convertDouble = convertDouble;
  }

  @TypeConversion(rule = ConversionRule.COLLECTION, converter
= "java.util.String")
  public void setUsers( List users ) {
      this.users = users;
  }

  @TypeConversion(rule = ConversionRule.MAP, converter =
"java.math.BigInteger")
  public void setKeyValues( HashMap keyValues ) {
      this.keyValues = keyValues;
  }

  @TypeConversion(type = ConversionType.APPLICATION, property
= "java.util.Date", converter =
"com.opensymphony.xwork2.util.XWorkBasicConverter")
  public String execute() throws Exception {
     return SUCCESS;
  }
}
```

Then, explain them about other annotations such as:

**Element Annotation**

Sets an element for the type conversion. It can be applied at field level or method level.

**Key Annotation**

Sets the Key for type conversion.

**In-Class Question:**

After you finish explaining type conversion annotations, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.

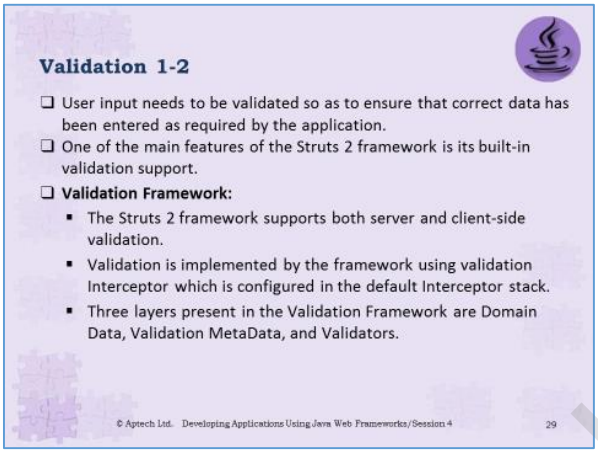
Which annotation is used for generic types to specify the key type for Map keys?

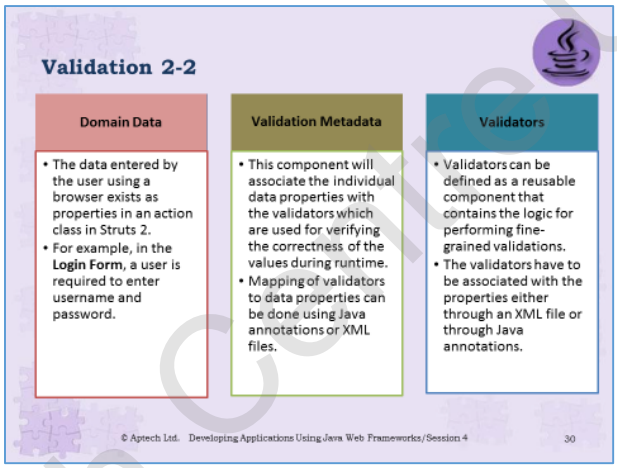
**Answer:**

Key annotation.

## Slides 29 and 30

Let us understand validation.




Use slides 29 and 30 to explain Validation.

All Web applications require user input which range from a simple username and password to data entered in a complex form. Values entered in the application are required to be validated before they are stored in the database.

User input needs to be validated so as to ensure that correct data has been entered as required by the application. The task of validating user input is complex. One of the main features of the Struts 2 framework is its built-in validation support. Thus, this helps to reduce the code specification and makes the tasks of validating the user input easy and simple.

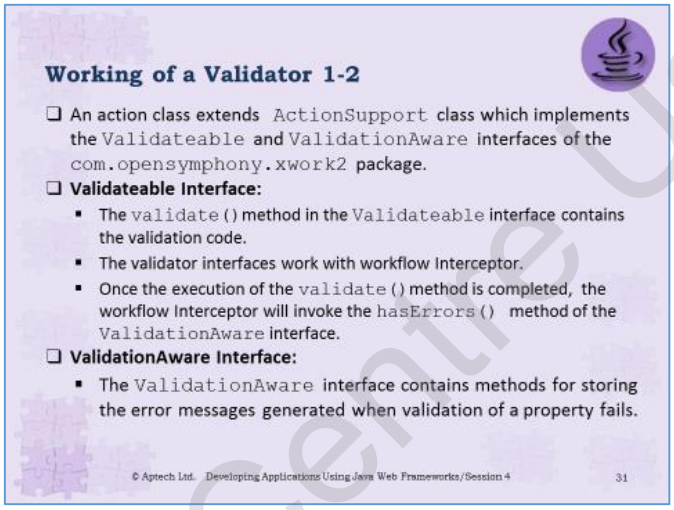
The Struts 2 framework supports both server and client side validation. It provides a wide range of validation rules and pre-defined validators which can be applied to form fields for validations.

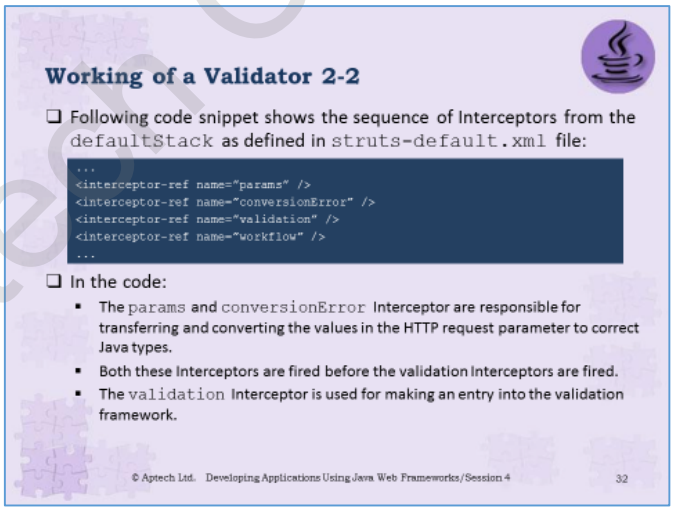
Developers can also create custom validators based on the validation requirements. For using pre-defined validators, no initial configuration is required. Validation is implemented by the framework using validation Interceptor which is configured in the default Interceptor stack.

Then, explain them three layers present in the Validation framework as mentioned on slide 30.

### Slides 31 and 32

Let us understand the working of a validator.




Use slides 31 and 32 to explain the working of a validator.

An action class extends `ActionSupport` class which implements the `Validateable` and `ValidationAware` interfaces of the com.opensymphony.xwork2 package.

The `validate()` method in the `Validateable` interface contains the validation code whereas the `ValidationAware` interfaces contains methods for storing the error messages generated when validation of a property fails.

Then, explain them about `Validateable` and `ValidationAware` interfaces as mentioned on slide 31.

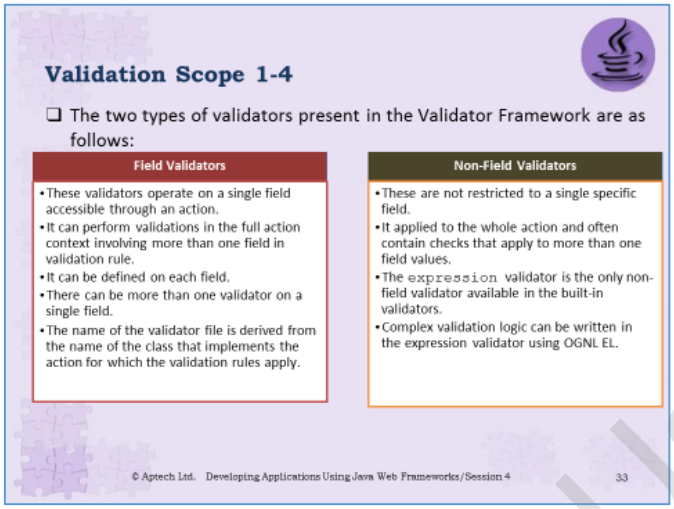
The code snippet as shown on slide 32 explains the sequence of Interceptors from the `defaultStack` as defined in `struts-default.xml` file. In the given code, `params` and `conversionError` Interceptors are responsible for transferring and converting the values in the HTTP request parameter to correct Java types. Both these Interceptors are fired before the validation Interceptors are fired. The validation Interceptor is used for making an entry into the validation framework. When this Interceptor fires, all the validations are checked that have been defined in the validation metadata. If conversion and validation errors are encountered, then they are stored or added in the methods of the `ValidationAware` interface.

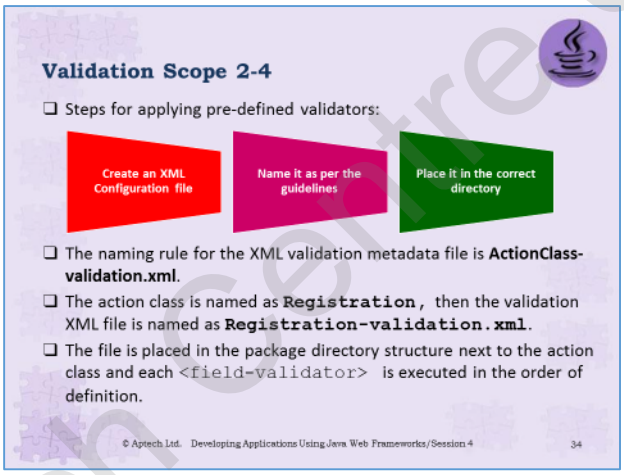
The `workflow` Interceptor works in two phases. In the first phase, it invokes the `validate()` method if present in the current action and performs the basic validation. In the second phase, it checks for errors in the `hasErrors()` method of the `ValidationAware` interface. If there are no errors, then the control passes to the rest of the action invocation process. When the `defaultStack` is used, both the validation and the workflow Interceptors fire every time. This means that both the forms of validation can be used. Thus, first the validation Interceptor runs all the validations that are defined with the validation framework metadata.

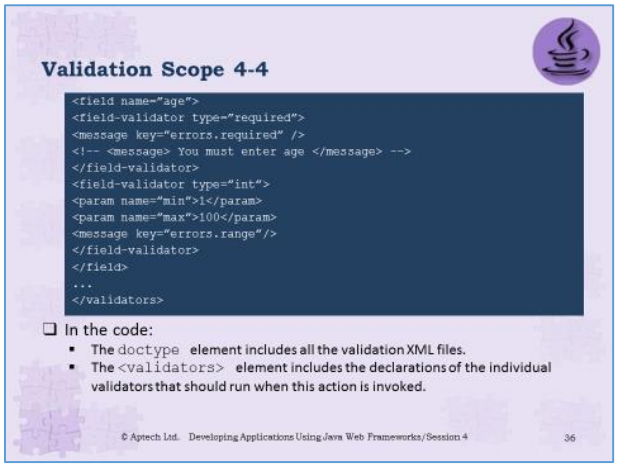
Next, when the workflow Interceptor runs, it checks whether the `validate()` method is implemented in the action class. The `validate()` method can contain some extra validation code.

## Slides 33 to 36

Let us understand validation scope.




Use slides 33 to 36 to explain validation scope. Tell the students that field validators act on single fields attainable through an action. Non-field validators are domain specific and provide custom implementations.

Slides 33 and 34 describes about field validators and non-field validators.

Then, tell them about the steps for applying pre-defined validators or field validators as explained on slides 35 and 36.

The name of the validator file is derived from the name of the class that implements the action for which the validation rules apply. The naming rule for the XML validation metadata file is `ActionClass-validation.xml`. Thus, if the action class is named as Registration then the validation XML file is named as `Registration-validation.xml`. The file is placed in the package directory structure next to the action class. Each `<field-validator>` is executed in the order of definition.

**Tips:**

- If you want to create to apply single validation to an action class, then the validation xml file will be named as (Action class name)-validation.xml.

- If you want to apply multiple set of validations for an action class, then validation file should be named as (Action class name)-Alias-validation.xml.

**In-Class Question:**

After you finish explaining validation scope, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.

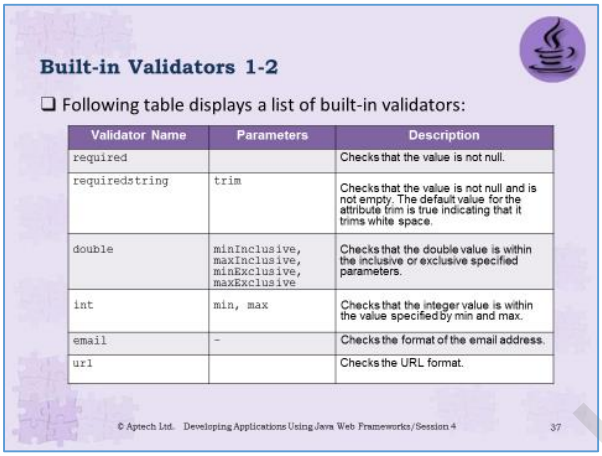
Which validators operate on a single field accessible through an action?

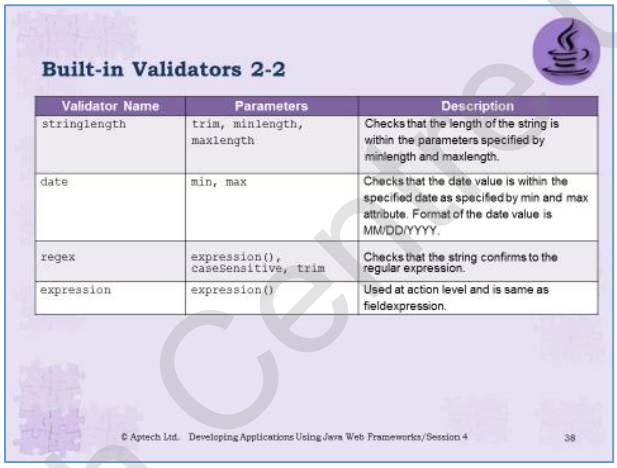
**Answer:**

Field validators.

## Slides 37 and 38

Let us understand built-in validators.




Use slides 37 and 38 to explain built-in validators. The framework provides built-in validators to handle the validation requirements. The validators are present in XWork jar file which contains an XML file that declares all the built-in validators.

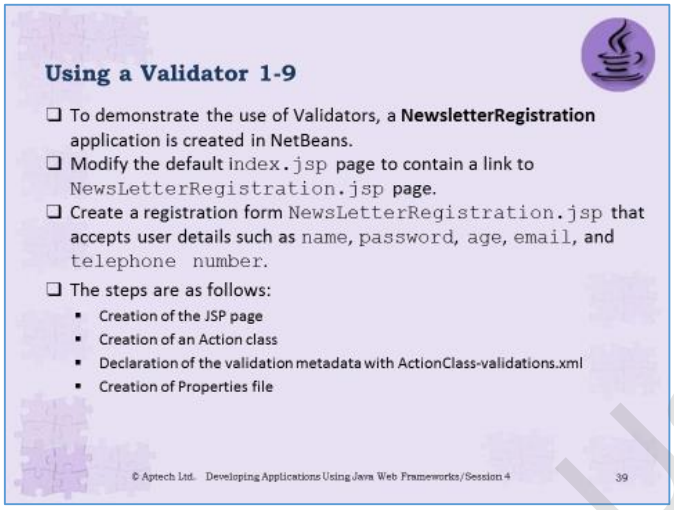
**Tips:**

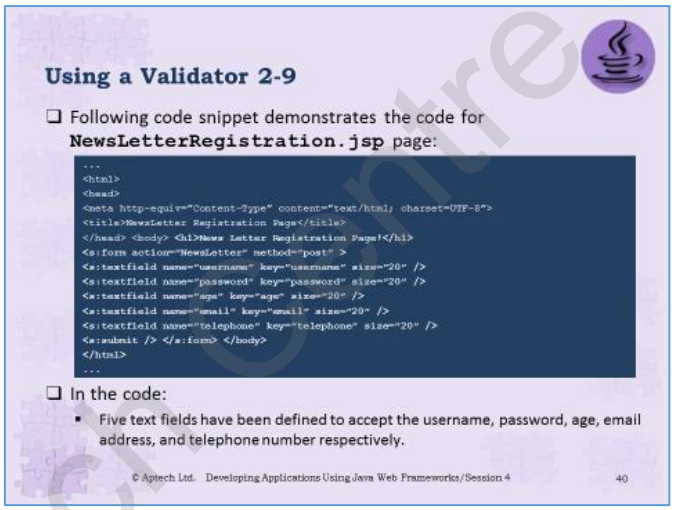
The other types of built-in validators are as follows:

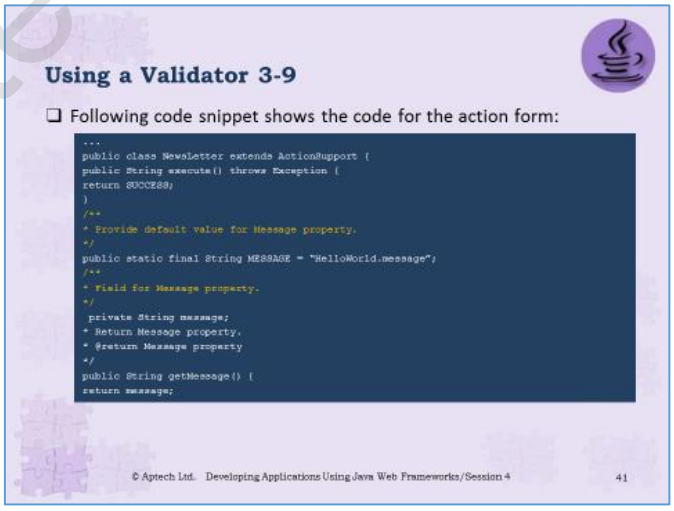
- `ConditionalVisitorField Validator` – Forwards validation to the `VisitorFieldValidator` only if the expression will evaluate to true.
- `VisitorFieldValidator` – Forwards validation to object properties of user's action using the object's own validation files. It can handle either simple Object properties, Collections of Objects, or Arrays.
- `Short validators` – Field Validators that checks if the short specified is within a certain range.

## Slides 39 to 47

Let us understand how to use a validator.



## Using a Validator 4-9

```
}
public void setMessage(String message) {
this.message = message;
}
private String username;
private String password;
private String telephone;
private String email;
private int age;
public String getUsername() {
return username;
}
public void setUsername(String userName) {
this.username = userName;
}
public String getPassword() {
return password;
}
public String getTelephone() {
return telephone;
}
public void setTelephone(String telephone) {
```



## Using a Validator 5-9

```
this.telephone = telephone;
}
public String getEmail() {
return email;
}
public void setEmail(String email) {
this.email = email;
}
public int getAge() {
return age;
}
public void setAge(int age) {
this.age = age;
}
}
...
```

- In the code, the action class receives the data from the JSP page and sets the property value.



## Using a Validator 6-9

- Following code snippet shows the NewsLetter action's validation metadata file, **NewsLetter-validation.xml**:

```
...
<!DOCTYPE validators PUBLIC

<validators> <field name="username">
<field-validator type="requiredstring">
  <param name="trim">true</param>
  <message key="errors.required" />
</field-validator> </field> <field name="password">
<field-validator type="requiredstring">
   <param name="trim">true</param>
  <message key="errors.required" /> </field-validator>
</field> <field name="age">
<field-validator type="required"> <message key="errors.required" />
<!-- <message> You must enter age </message> -->
</field-validator>
<field-validator type="int">
<param name="min">1</param>
<param name="max">100</param>
<message key="errors.range"/>
```

## Using a Validator 7-9

```
</field-validator>
</field>
<field name="email">
<field-validator type="requiredstring">
<message key="errors.required" />
</field-validator>
<field-validator type="email">
<message key="errors.invalid" />
</field-validator>
</field>
<field name="telephone">
<field-validator type="requiredstring">
<message key="errors.required" />
</field-validator>
</field>
</validators>
```

- In the code:
  - Name of the file is derived from the name of the class that implements the action for which the validation rules apply.
  - The file is placed in the package directory next to the action class.
  - The `doc` type element must be included in the validation XML file.

© Aptech Ltd. Developing Applications Using Java Web Frameworks/Session 4 45

## Using a Validator 8-9

- Following code snippet demonstrates the implementation of **properties** file:

```
HelloWorld.message= Welcome to Struts Validation ...
username=Username
age=Age
email=Email
telephone=Telephone
password=Password
errors.invalid=${getText(fieldName)} is invalid.
errors.required=${getText(fieldName)} is required.
errors.number=${getText(fieldName)} must be a number.
errors.range=${getText(fieldName)} is not in the range ${min} and ${max}.
```

- In the code:
  - The different error messages are stored for each of the text fields to be validated.
  - For example, if the `requiredstring` validator encounters a null value for the `username` property of the action class, the respective error message is retrieved and displayed.

© Aptech Ltd. Developing Applications Using Java Web Frameworks/Session 4 46

## Using a Validator 9-9

- Following figure shows the application startup page:

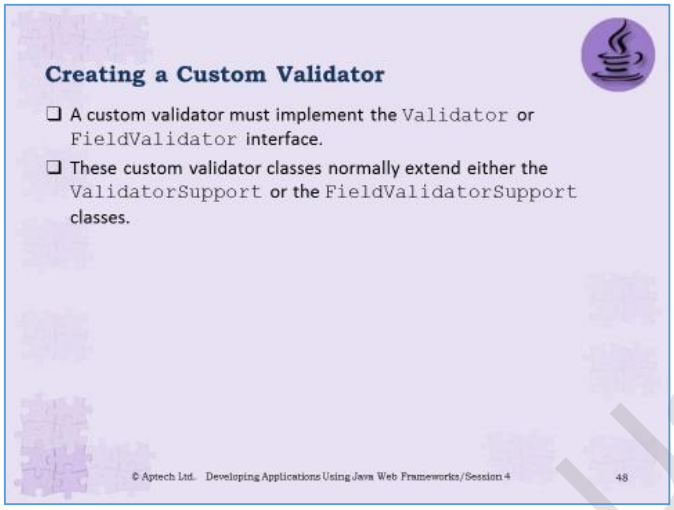

- Following figure displays the error page showing the validation messages:


© Aptech Ltd. Developing Applications Using Java Web Frameworks/Session 4 47

Use slides 39 to 47 to explain how to use a validator.

## Slide 48

Let us understand how to create a custom validator.




Use slide 48 to explain how to create a custom validator. Tell the students that user can define their own custom validation by implementing the `Validateable` interface in the action class.

The method used in `Validateable` interface is `validate()`.`ValidationAware` interface can accept the field level or action class level error messages. It is overridden in the action class to define the validation logic.

Some methods of `ValidationAware` interface are as follows:

- `void addFieldError(String fieldName,String errorMessage)` – adds the error message for the specified field.
- `void addActionError(String errorMessage)` – adds an Action-level error message for this action.
- `void addActionMessage(String message)` – adds an Action-level message for this action.
- `void setFieldErrors(Map<String,List<String>> map)` – sets a collection of error messages for fields.
- `void setActionErrors(Collection<String> errorMessages)` – sets a collection of error messages for this action.
- `void setActionMessages(Collection<String> messages)` – sets a collection of messages for this action.

- `boolean hasErrors()` – checks if there are any field or action errors.
- `boolean hasFieldErrors()` – checks if there are any field errors.
- `boolean hasActionErrors()` – checks if there are any Action-level error messages.
- `boolean hasActionMessages()` – checks if there are any Action-level messages.

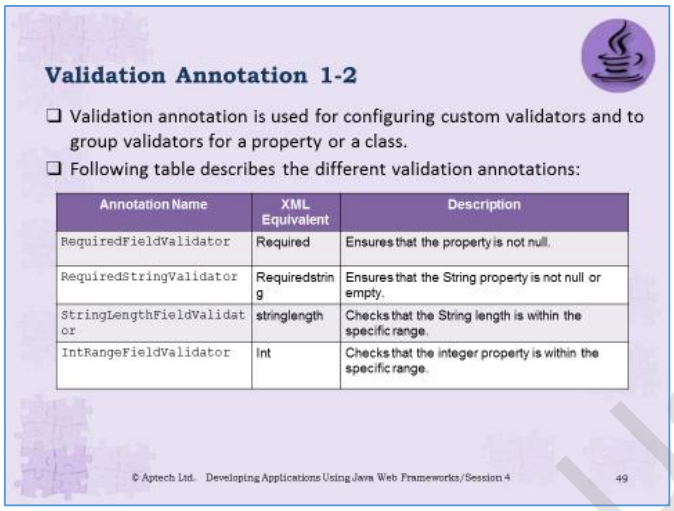
**Tips:**

You can work with an example on custom validator by referring this link:

http://ddubbya.blogspot.in/2011/02/creating-custom-struts2-validator.html

## Slides 49 and 50

Let us understand validation annotation.

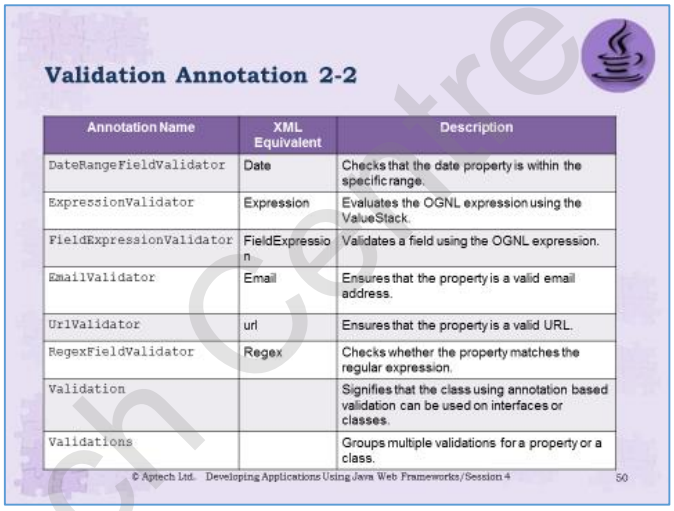


**Validation Annotation 1-2**

- Validation annotation is used for configuring custom validators and to group validators for a property or a class.
- Following table describes the different validation annotations:

| Annotation Name | XML Equivalent | Description |
|---|---|---|
| RequiredFieldValidator | Required | Ensures that the property is not null. |
| RequiredStringValidator | Requiredstring | Ensures that the String property is not null or empty. |
| StringLengthFieldValidator | stringlength | Checks that the String length is within the specific range. |
| IntRangeFieldValidator | Int | Checks that the integer property is within the specific range. |





**Validation Annotation 2-2**

| Annotation Name | XML Equivalent | Description |
|---|---|---|
| DateRangeFieldValidator | Date | Checks that the date property is within the specific range. |
| ExpressionValidator | Expression | Evaluates the OGNL expression using the ValueStack. |
| FieldExpressionValidator | FieldExpression | Validates a field using the OGNL expression. |
| EmailValidator | Email | Ensures that the property is a valid email address. |
| UrlValidator | url | Ensures that the property is a valid URL. |
| RegexFieldValidator | Regex | Checks whether the property matches the regular expression. |
| Validation | | Signifies that the class using annotation based validation can be used on interfaces or classes. |
| Validations | | Groups multiple validations for a property or a class. |



Use slides 49 and 50 to explain validation annotation.

Some other types of validation annotations are as follows:

- `DoubleRangeFieldValidator` – Checks that a double field has a value within a specified range.
- `VisitorFieldValidator` – Invokes the validation for a property's object type.
- `CustomValidator` – Use this annotation for user's custom validator types.

**Tips:**

For more information on validation annotations, you can refer to this link:
http://www.codejava.net/frameworks/struts/struts2-form-validation-basic-example-using-annotations

## Slides 51 to 53

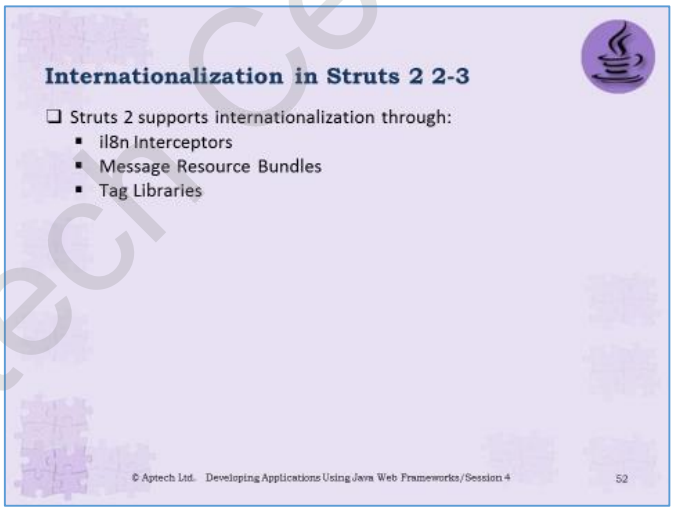
Let us understand internationalization in Struts 2.

### Internationalization in Struts 2 1-3

- **Internationalization**
  - Is the technique for developing applications that support multiple languages and data formats without the need to re-engineer the application.
  - Sometimes, the term 'Internationalization' is abbreviated as `I18N`, because there are 18 letters between the first 'I' and the last 'N.'
- Following table shows the list of few country and language codes:

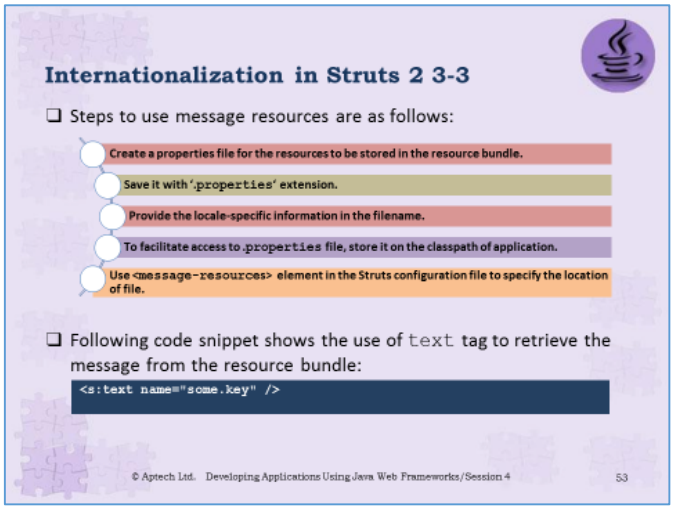
| Country Name | Country Code | Language | Language Code |
|---|---|---|---|
| Italy | IT | Italian | it |
| China | CN | Chinese | zh |
| France | FR | French | fr |
| Germany | DE | German | de |
| Japan | JP | Japanese | ja |
| Spain | ES | Spanish | es |
| United States | US | English | en |





### Internationalization in Struts 2 2-3

- Struts 2 supports internationalization through:
  - il8n Interceptors
  - Message Resource Bundles
  - Tag Libraries




Use slides 51 to 53 to explain internationalization in Struts 2.

Internationalization is the technique for developing applications that support multiple languages and data formats without the need to re-engineer the application. By designing application ahead of time, developers need not write programming logic again when a new language or country is to be supported by the application. Sometimes the term Internationalization is abbreviated as i18N, because there are 18 letters between the first 'I' and the last 'N.'

Each language and country has unique codes representing the language and country respectively. These codes are assigned by International Organization for Standardization (ISO).

Then, tell them the i18n interceptor provides multi-lingual support for your application. It handles setting locale for the action. It can be used if user wants to set their location locale and get data according to the locale provided. The i18n allows the developer to set the locale for the action. A locale is an identifier and represents the local details of the user to the application. The local can be created by `java.util.Locale` class. When a user session begins, the Locale object is sent in the HTTP request as parameter.

There are two parameters defined for i18n interceptor. Both are optional.

- **parameterName** - It specifies the name of the HTTP request parameter. It is set to request_locale by default.
- **attributeName** – It specifies the name of the session key to store the locale. It is WW_TRANS_I18N_LOCALE by default.

Another component required for internationalization is Message Resource Bundle. Resource bundle is a flat file that contains key-value pairs. These bundles can be used as central repository for contents that have multiple uses. In order to use resource bundles one needs to create a property file that contains key/value pairs.

For example, to create a Struts 2 Web application for two locales, English and French. So, two resource bundles need to be created.

**English Resource Bundle**

| Key | Value |
|---|---|
| key_name | value_value |
| key_name | value_value |

**French Resource Bundle**

| Key | Value |
|---|---|
| key_name | value_value |
| key_name | value_value |

When we pass localization key, Struts 2 framework looks for resource bundles at various places in the given order:

- `{ActionClassName}.properties` (it should be in the same package with Action Class)
- `Interface.properties` (every interface and sub-interface)
- `BaseClass.properties` (all the way to Object.properties)
- ModelDriven's model (if implements ModelDriven)
- `package.properties` in the class package and then to the parent packages till the root
- Global resource properties configured in struts property file

To use message resources, the steps to be followed are:

1. Create a properties file for the resources to be stored in resource bundle.
2. Save it with '`.properties`' extension. Struts application use `ApplicationResources.properties` as standard way to refer to this file.
3. Provide the locale-specific information in the filename. For example, to create a Spanish version, name the file as `Applicationresources _ es.properties`. The class `PropertyMessageResource` of Struts will transparently select the required locale-specific resource bundle. In case of unavailability, Struts will load the resource from the default resource bundle.

4. To facilitate access to .properties file, store it on the classpath of application. (For example, in `/WEB-INF/classes/ directory`). This is because Java's class loader is used by Struts to load the properties file.
5. Use `<message-resources>` element in the Struts configuration file to specify the location of file. The value of parameter attribute of message-resources element is set to the location of .properties file. The key attribute of element can be used to specify each additional resource bundles.

**In-Class Question:**

After you finish explaining internationalization in Struts 2, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.

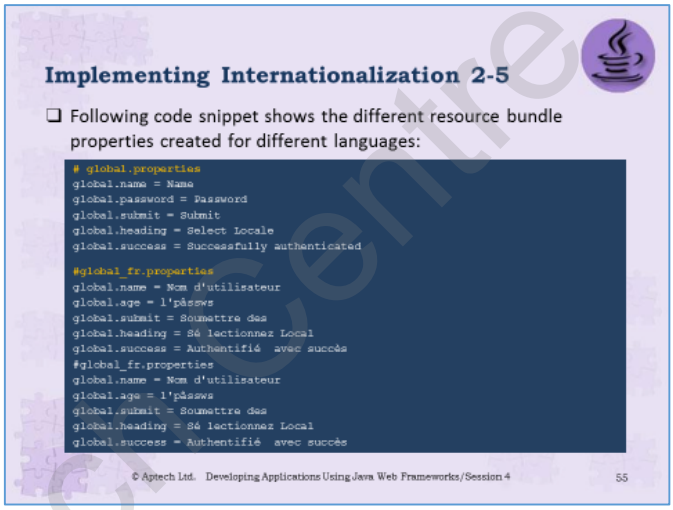
Which technique helps in developing applications that support multiple languages and data formats without the need to re-engineer the application?

**Answer:**

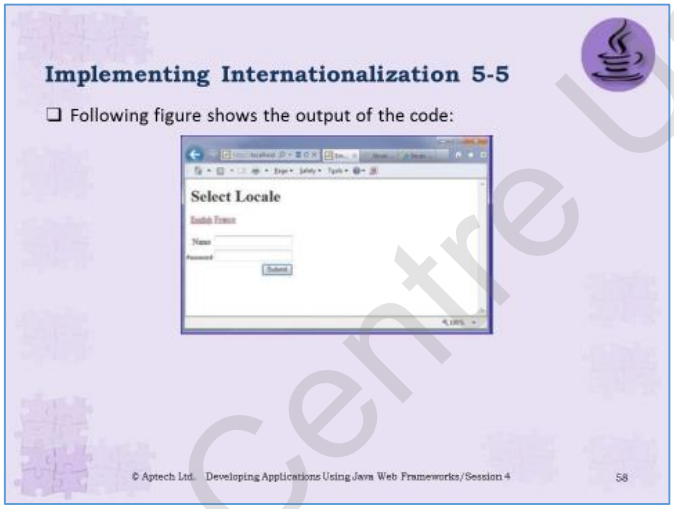
Internationalization.

## Slides 54 to 58

Let us understand how to implement internationalization.

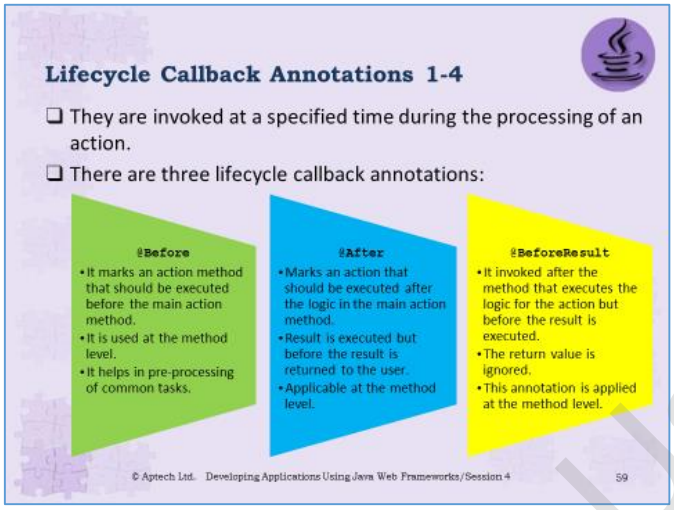


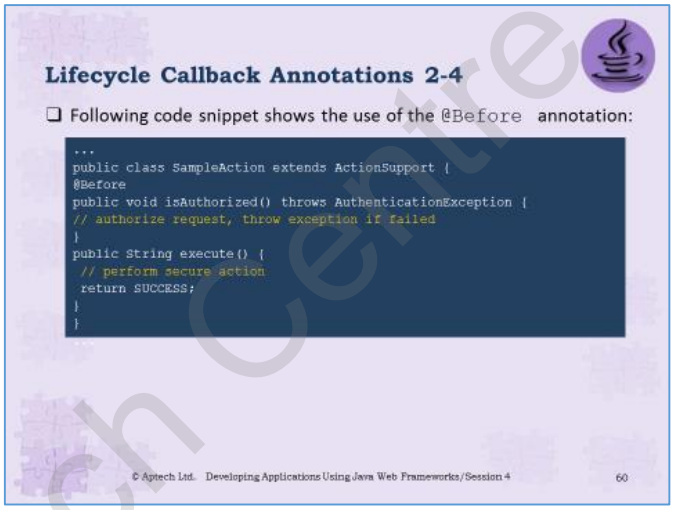


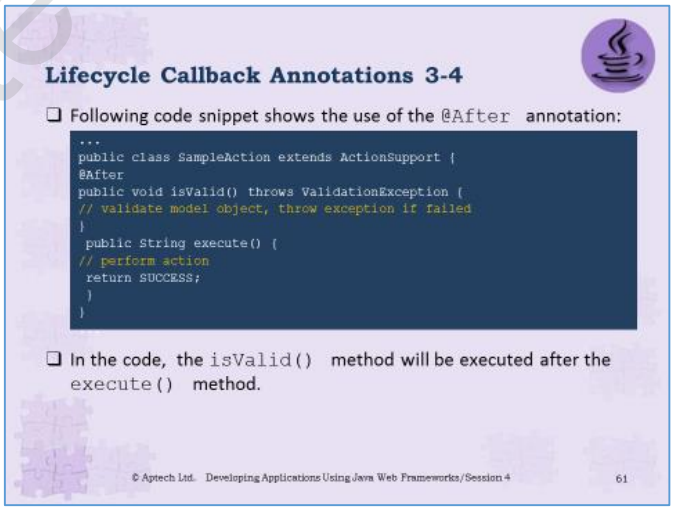

Use slides 54 to 58 to explain how to implement internationalization.

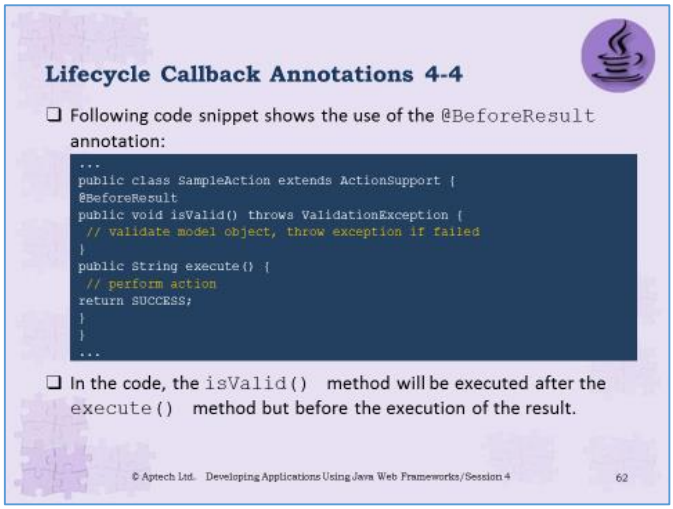
## Slides 59 to 62

Let us understand lifecycle callback annotations.






Use slides 59 to 62 to explain lifecycle callback annotations.

Life cycle callback annotations are invoked at a specified time during the processing of an action.

There are three life cycle callback annotations. To use these annotations, you have to specify the `AnnotationWorkflwInterceptor` to your Interceptor task.

Then, explain different types of life cycle callback interceptors as explained on slide 59.

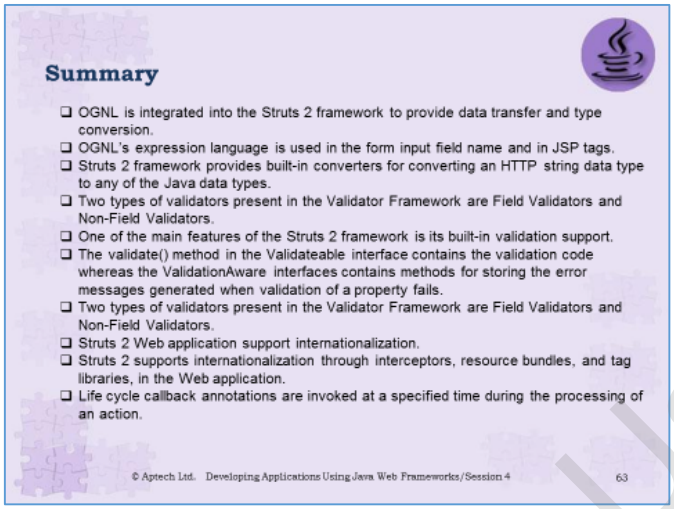
Then, using slide 60, explain `@Before` annotation that marks an action method that should be executed before the main action method.

Then, using slide 61, explain `@After` annotation that marks an action that should be executed after the logic in the main action method and the result is executed before it is returned to the user.

Then, using slide 62, explain the `@BeforeResult` annotation that will be invoked after the method that executes the logic for the action, but before the result is executed. The return value is ignored. This annotation is applied at the method level.

## Slide 63

Let us summarize the session.




In slide 63, you will summarize the session. You will end the session, with a brief summary of what has been taught in the session.

### 4.3 Post Class Activities for Faculty

You should familiarize yourself with the topics of the next session. You should also explore the JavaServer Faces technology introduced in the next session.

**Tips:**

You can also check the Articles/Blogs/Expert Videos uploaded on the OnlineVarsity site to gain additional information related to the topics covered in the next session. You can also connect to online tutors on the OnlineVarsity site to ask queries related to the sessions.

# Session 5 – Introduction to JavaServer Faces

## 5.1 Pre-Class Activities

Before you commence the session, you should familiarize yourself with the topics of this session in-depth. You should revisit topics of the previous session for a brief review.

Here, you can ask students the key topics they can recall from previous session. Prepare a question or two which will be a key point to relate the current session objectives.

### 5.1.1 Objectives

By the end of this session, the learners will be able to:

- Explain JSF framework
- Explain the components of JSF framework
- Explain the different types of configuration files used in JSF applications
- Describe JSF architecture and JSF lifecycle
- Explain the process of developing JSF application
- Explain UI components and the renderers in JSF
- Explain the concept of managed beans in JSF
- Explain basic tag, converter tag, and validator tag in JSF

### 5.1.2 Teaching Skills

To teach this session successfully, you should be aware of JSF framework and its components. Familiarize yourself with different types of configuration files used in JSF applications. Aware yourself with JSF architecture and JSF lifecycle. You should know the process of developing JSF application and also about UI components and the renderers in JSF.

Aware yourself with the concept of managed beans and about basic tag, converter tag, and validator tag in JSF.

For teaching in the class, you are expected to use slides and LCD projectors.

**Tips:**

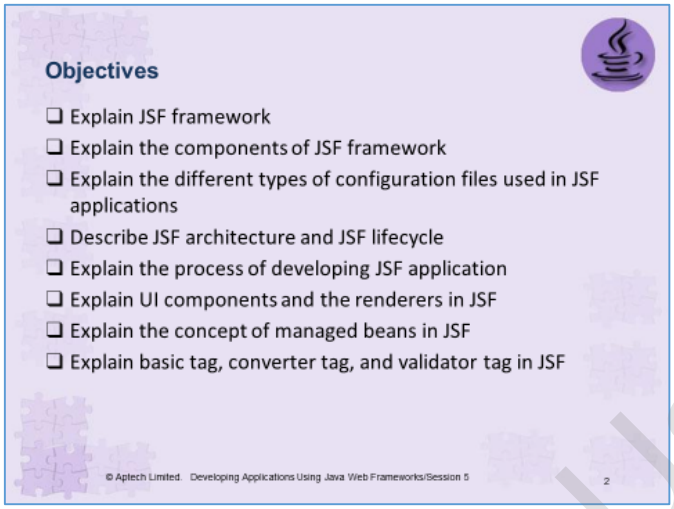
It is recommended that you test the understanding of the students by asking questions in between the class.

**In-Class Activities:**

Follow the order given here during In-Class activities.

## Overview of the Session:

Give the students a brief overview of the current session in the form of session objectives.
Show the students slide 2 of the presentation.




Tell the students that they will be introduced to JSF framework and its components. The session explains the different types of configuration files used in JSF applications. You should also know the process of developing JSF application and also about UI components and their renderers in JSF.

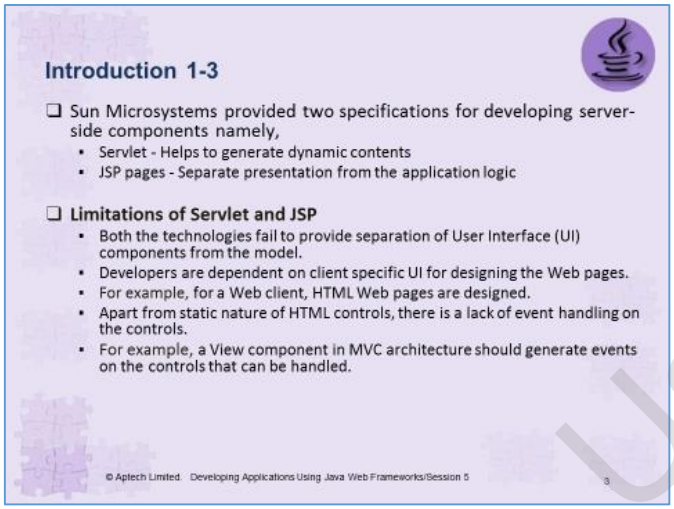
Further, the session will also explain them about JSF architecture and JSF lifecycle. Finally, discuss the process of developing JSF application.

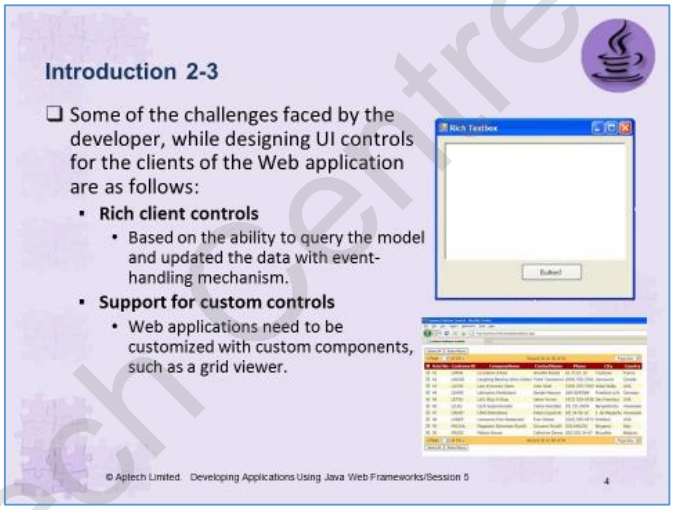
Finally, discuss about the concept of managed beans and about basic tag, converter tag, and validator tag in JSF.

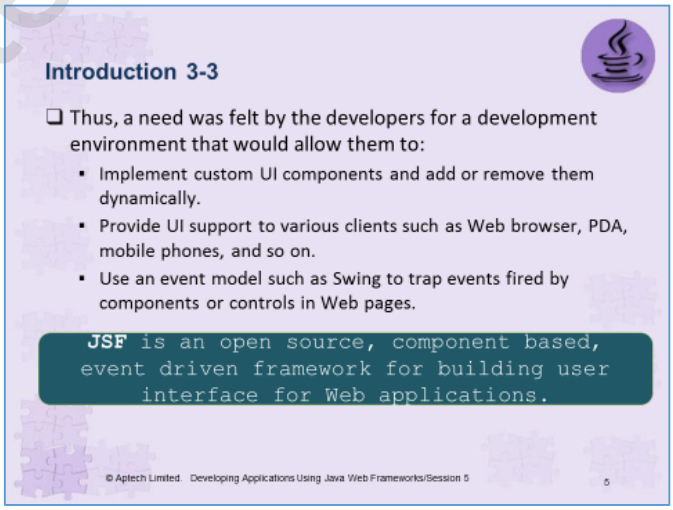
## 5.2 In-Class Explanations

### Slides 3 to 5

Let us understand the limitations of Servlet and JSP.




Use slides 3 to 5 to explain the limitations of Servlet and JSP.

Tell them that Sun Microsystems provided two specifications for developing server-side components namely, Servlet and JSP.

Servlets help to generate dynamic contents, whereas JSP pages separate presentation from the application logic. However, both the technologies fail to provide separation of User Interface (UI) components from the model. Thus, developers are dependent on client specific UI for designing the Web pages. For example, for a Web client, HTML Web pages are designed. As HTML controls are static in nature, which means they are not powered with the dynamic functionalities of handling the data. Thus, developers access the data from the form controls using HTTP request and response objects.

Apart from static nature of HTML controls, there is a lack of event handling on the controls. For example, a view component in MVC architecture should generate events on the controls that can be handled. For example, Swing controls are developed with MVC architecture. When value changes in a UI component, the model gets updated based on the event fired on the control.

Then, discuss the challenges faced by the developer while designing UI controls for the Web applications designed using Servlet and JSP technology as discussed on slide 4.

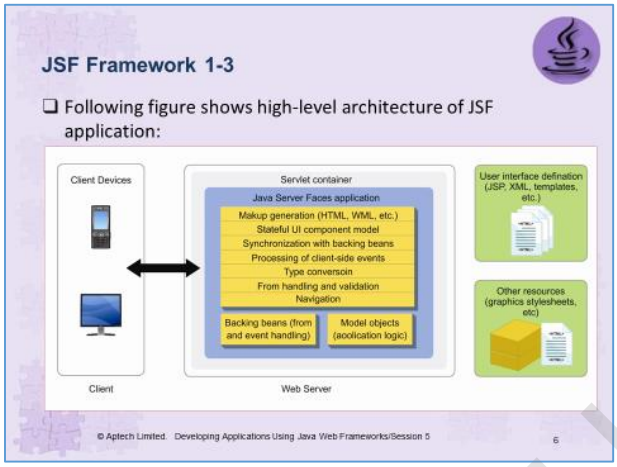
Thus, to deal with such requirements as mentioned on slide 4, the Sun Microsystems provided a server-side UI component framework named as JavaServer Faces (JSF). JSF is an open source, component based, event driven framework for building user interface for Web applications.

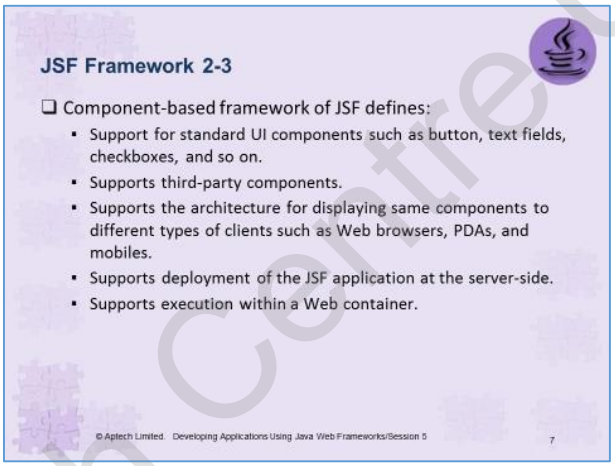
Then, discuss the main components of JSF technology that are as follows:

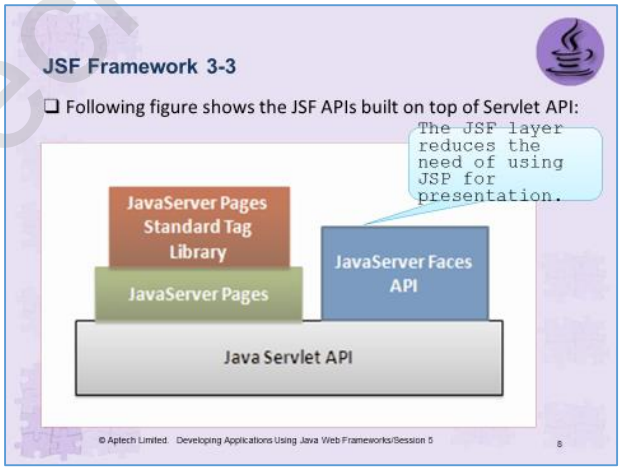
- A well-defined programming model that provides various UI components
- API to manage the state of UI controls on server-side with event-handling, validation, data conversion, and so forth.
- JSP custom tag libraries for providing UI components on a JSP page.

## Slides 6 to 8

Let us understand JSF framework.




Use slides 6 to 8 to explain JSF framework.

Tell the students that JSF helps to:

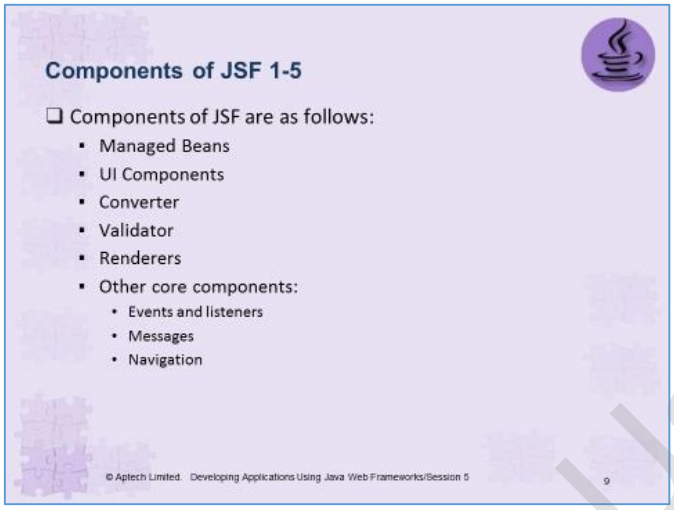
1. Create a Web page.
2. Drop components onto a Web page by adding component tags.
3. Bind components on a page to server-side data.
4. Wire component generated events to server-side application code.
5. Save and restore application state beyond the life of server requests.
6. Reuse and extend components through customization.

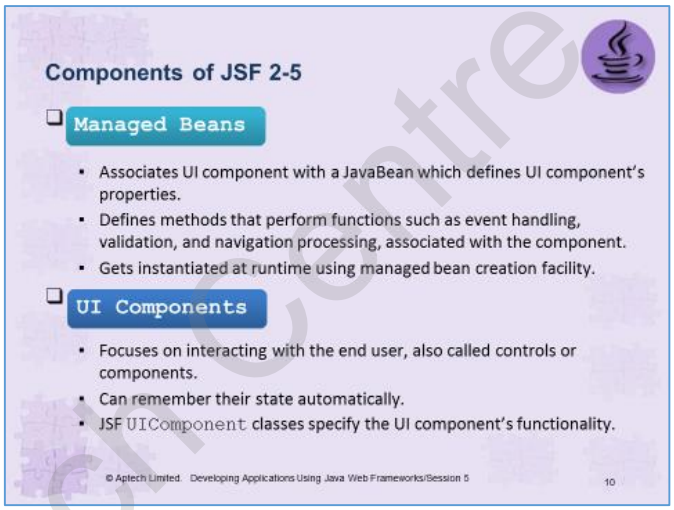
Using slides 6 and 7, explain the architecture of JSF. The component-based framework of JSF defines the support for standard UI components such as button, text fields, check boxes, and so on. It also supports third-party components. Components are based on event-driven programming model in which events are handled at the server-side. JSF supports the architecture for displaying same components to different types of clients such as Web browsers, PDAs, and mobiles. JSF applications are deployed at the server-side and are executed within a Web container.

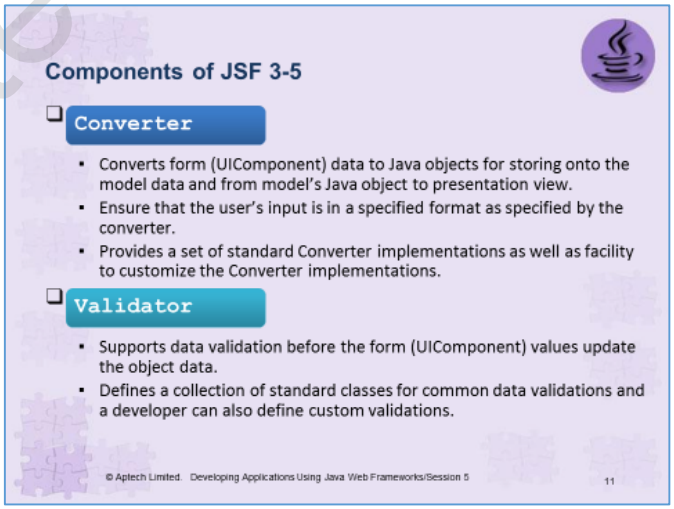
Using slide 8, explain the JSF API that are built on top of Servlet API. Tell them that the JSF layer reduces the need of using JSP for presentation. This enables the use of component classes for creating the presentations for different types of client devices.

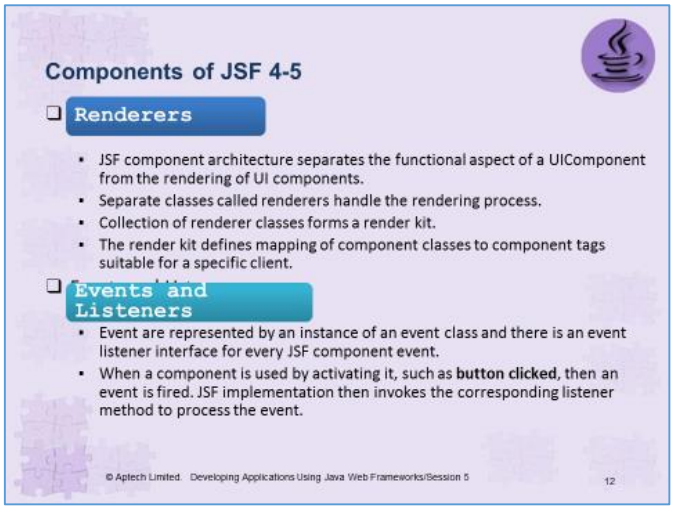
## Slides 9 to 13

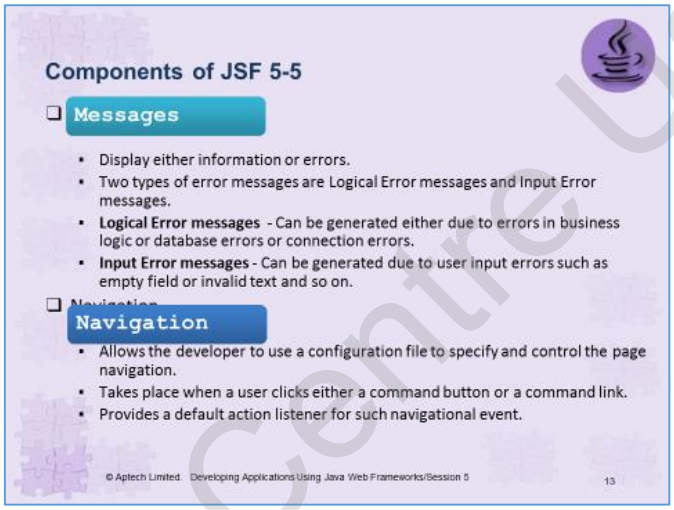
Let us understand the components of JSF.





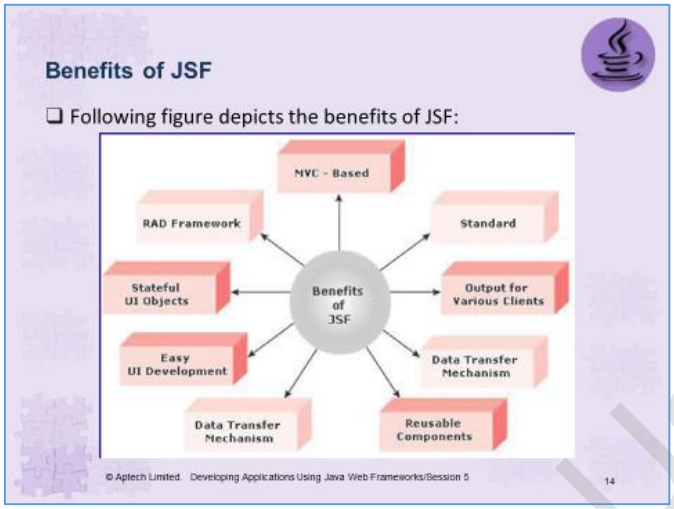

Use slides 9 to 13 to explain the components of JSF.

Explain them that the components of JSF that are as follows:

- Managed Beans
- UI components
- Converter
- Validator
- Renderers
- Events and listeners
- Navigation

## Slide 14

Let us understand the benefits of JSF.



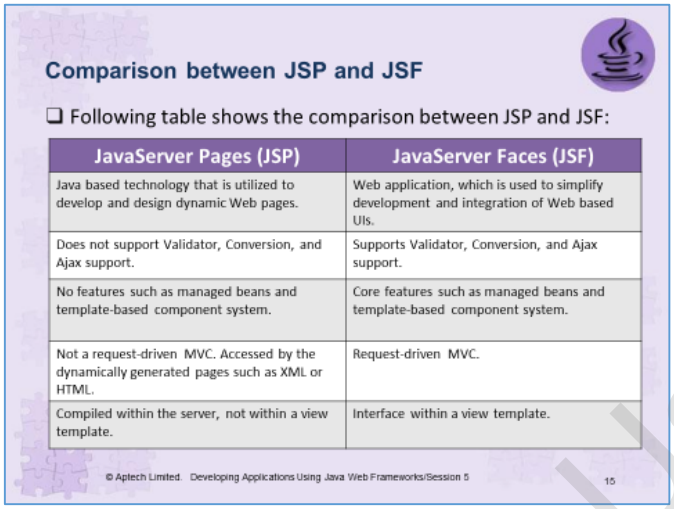

Use slide 14 to explain the benefits of JSF.

The other advantages of JSF are as follows:

1. JSF application can map HTTP requests to component-specific event handling and can manage UI elements as stateful objects on the server.
2. JSF technology helps the user to build Web applications that implement the separation of behavior and presentation that is normally given by client-side UI architectures.
3. JSF technology leverages familiar UI-component and Web-tier concepts without limiting the user to a particular scripting technology or markup language.
4. The JSF technology APIs are layered directly on top of the Servlet API. These APIs enables several important application use cases, such as using another presentation technology instead of JSP pages, creating custom components directly from the component classes, and generating output for various client devices.
5. JSF has inbuilt template layout features.

## Slide 15

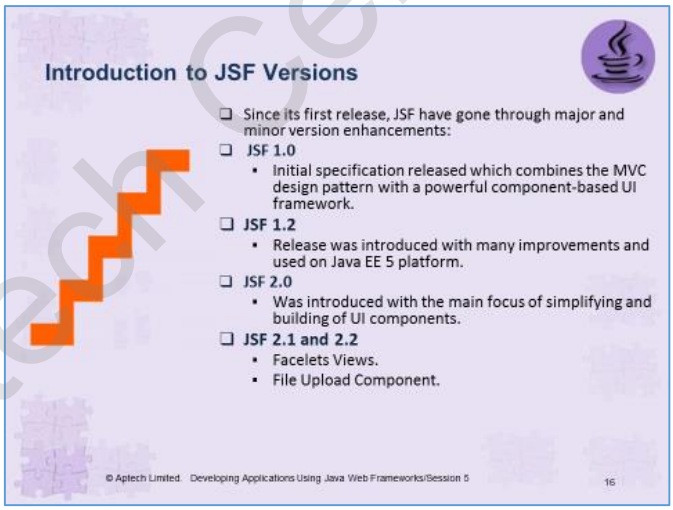
Let us compare JSP and JSF.




Use slide 15 to explain the difference between JSP and JSF.

## Slide 16

Let us introduce JSF versions.



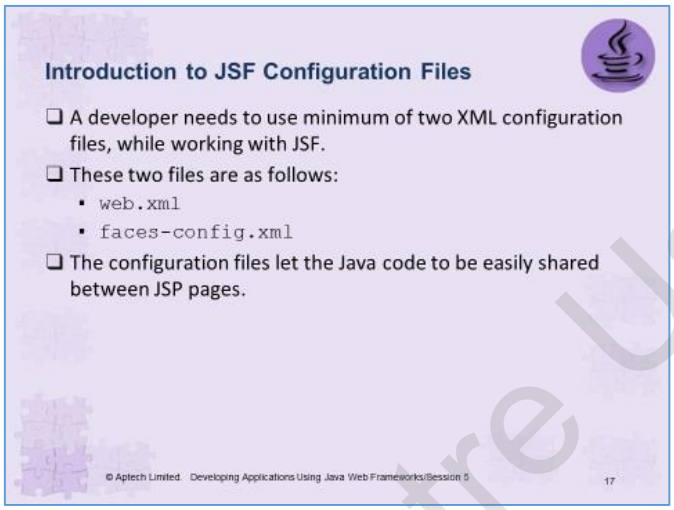

Use slide 16 to explain JSF versions.

Since its first release, JSF have gone through major and minor version enhancements. JSF 1.0 was the initial specification released which combines the MVC design pattern with a powerful component-based UI framework. This simplifies the development of Web applications on Java EE platform. JSF 1.2 release was introduced with many improvements and used on Java EE 5 platform.

The later version JSF 2.0 was introduced with the main focus of simplifying and building of UI components. In version 2.0, JSF is released in two minor version namely, JSF 2.1 and JSF 2.2.

## Slide 17

Let us introduce JSF configuration files.



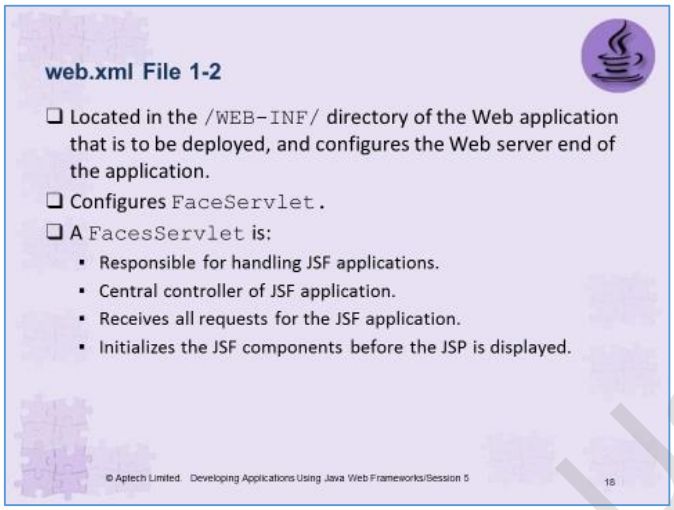

Use slide 17 to explain JSF configuration files.

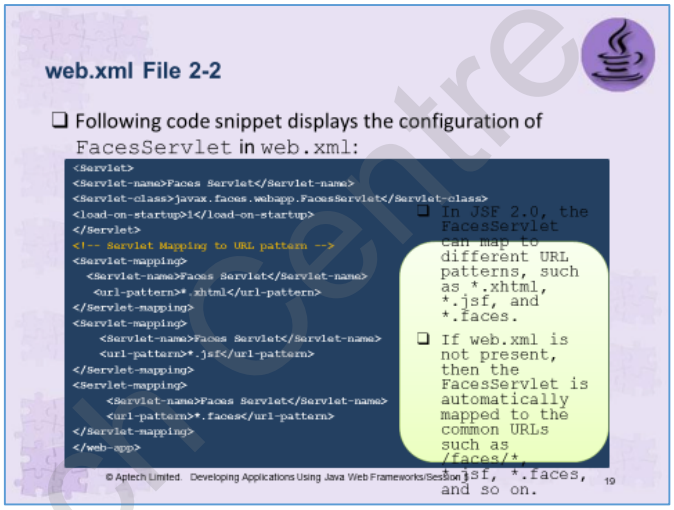
JSF applications use one or more XML descriptor files to configure applications. A developer needs to use minimum of two XML configuration files, while working with JSF. These two files, namely, `web.xml` and `faces-config.xml` are crucial to the flexibility and provide flexible coupling for the JSF architecture. The configuration files let the Java code to be easily shared between JSP pages.

JSF 2.0 introduced the use of annotations to configure the Web application components based on JSF framework. This makes the use of `faces-config.xml` optional.

## Slides 18 and 19

Let us understand `web.xml` File.




Use slides 18 and 19 to explain `web.xml` file.

Tell the students that JSF needs the central configuration `web.xml` to configure the controller of the JSF application. The `web.xml` is present in the directory `WEB-INF` of the application.

Using slide 18, explain that in `web.xml`, you must specify that a `FacesServlet` is responsible for handling JSF applications. The central controller of JSF application is `FacesServlet` and it receives all requests for the JSF application. In addition, `FacesServlet` initializes the JSF components before the JSP is displayed.

Then, explain the code snippet showing the configuration of `FacesServlet` as shown on slide 19.

**In-Class Question:**

After you finish explaining web.xml file, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.

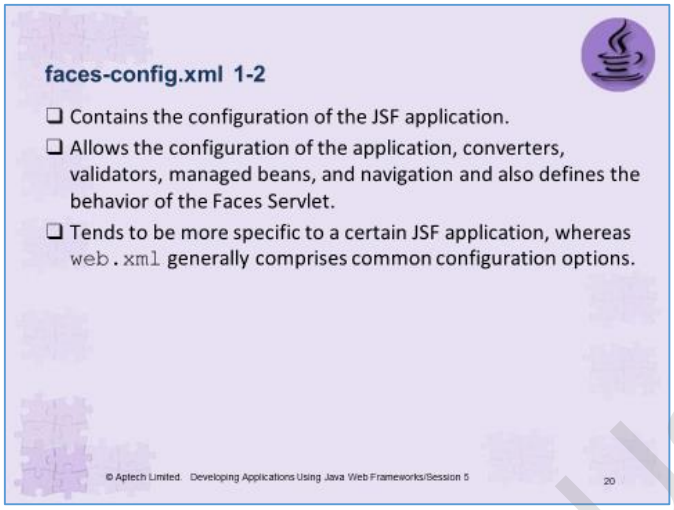
Which feature receives all requests for the JSF application and initializes the JSF components before the JSP is displayed?

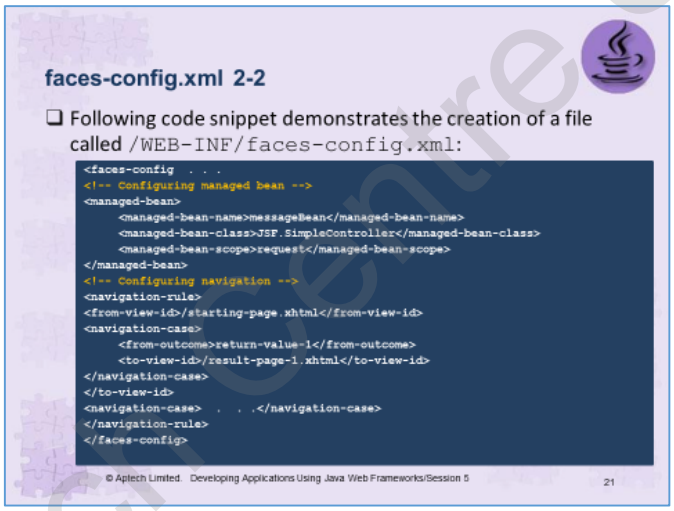
**Answer:**

`FacesServlet`

## Slides 20 and 21

Let us understand `faces-config.xml`.




Use slides 20 and 21 to explain `faces-config.xml`.

Tell the students that `faces-config.xml` allows the user to configure the JSF application components such as managed beans, convertors, validators, and navigation.

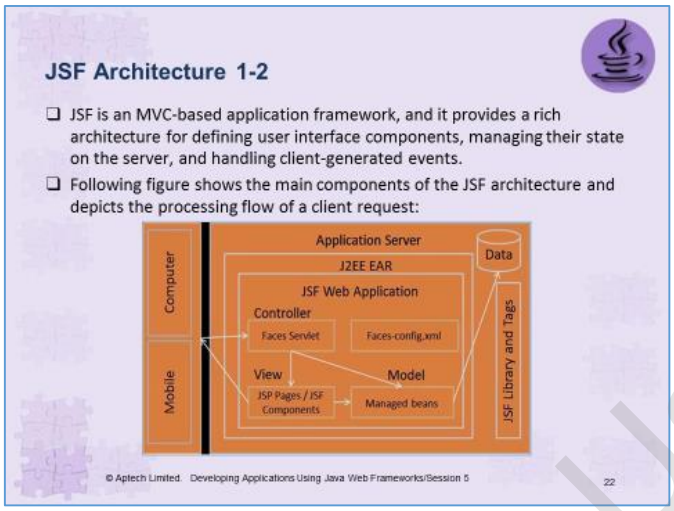
An application architect creates one or more files, called application configuration resource files that use the format to register and configure objects and to define navigation rules. An application configuration resource file is usually called `faces-config.xml`.

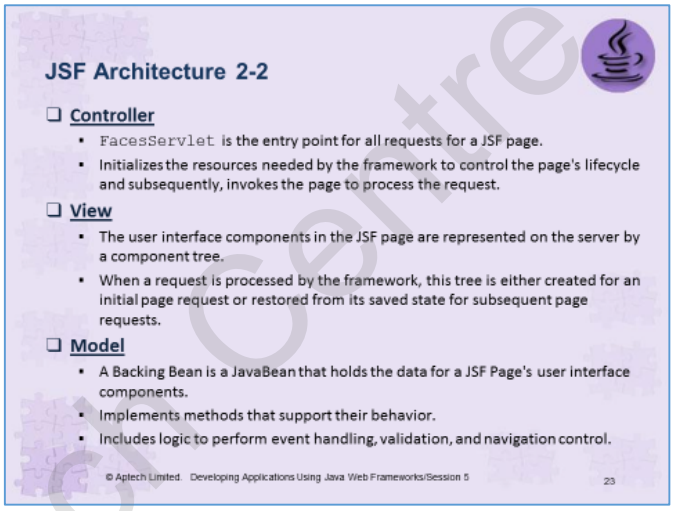
The `faces-config.xml` lists bean resources and navigation rules that are more specific to a certain application.

Explain the code snippet as shown on slide 21 to configure components in `faces-config.xml` file. However, from JSF 2.0, this file is optional, as all the information about the elements can be configured through annotations.

## Slides 22 and 23

Let us understand JSF architecture.




Use slides 22 and 23 to explain JSF Architecture.

Mention that JSF is an MVC-based application framework. It provides a rich architecture for defining user interface components, managing their state on the server, and handling client-generated events. It also provides support for validating user input and controlling page navigation.

All user interactions with the application are handled by a front-end 'Faces' servlet **Controller**.

**Controller**

Explaining the figure as shown on slide 22, tell them the `FacesServlet` is the entry point for all requests for a JSF page. Then, using slide 23, explain that **Controller** initializes the resources needed by the framework to control the page's life cycle and subsequently, invokes the page to process the request.

**View**

The user interface components in the JSF page are represented on the server by a component tree. This is also referred to as a **View**. When a request is processed by the framework, this tree is either created for an initial page request or restored from its saved state (for subsequent page requests).

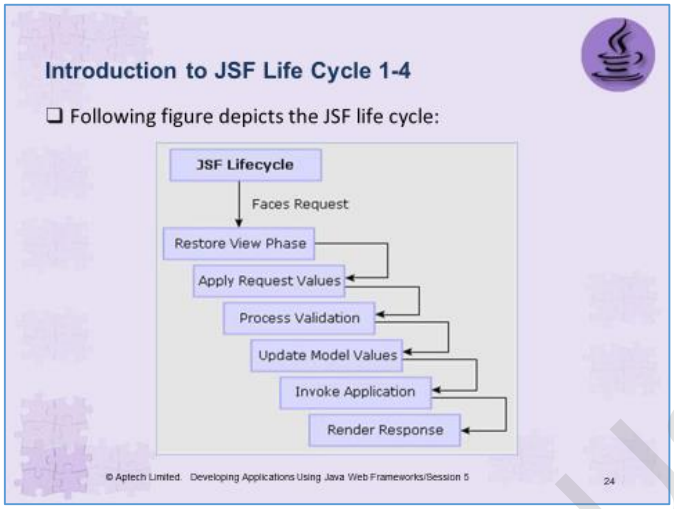
**Model**

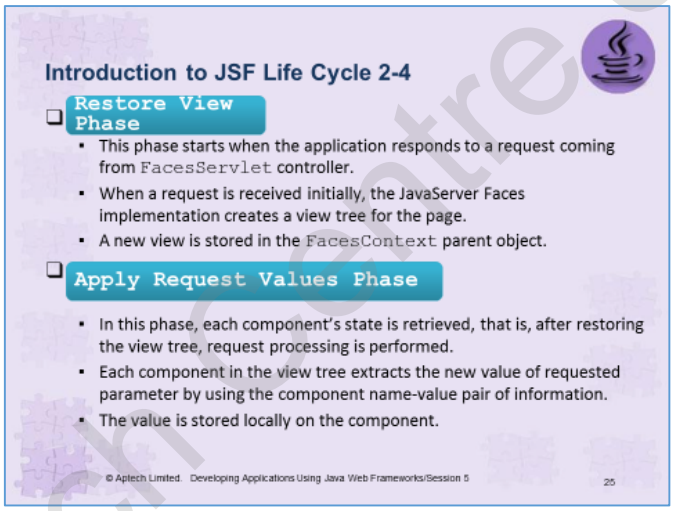
A Backing Bean is a JavaBean that holds the data for the user interface components on the JSF page. It implements methods that support their behavior. It includes logic to perform event handling, validation, and navigation control.

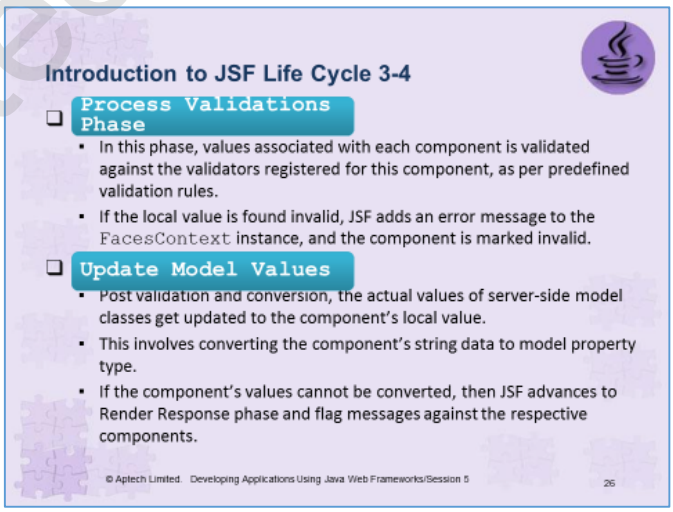
A Backing Bean usually invokes methods from a model object to perform business logic. JSF allows you to declare all the Backing Beans used by the page in the faces configuration file named as face-config.xml, so that they will be automatically instantiated by the Web container at application startup time. These beans are called **Managed beans**.

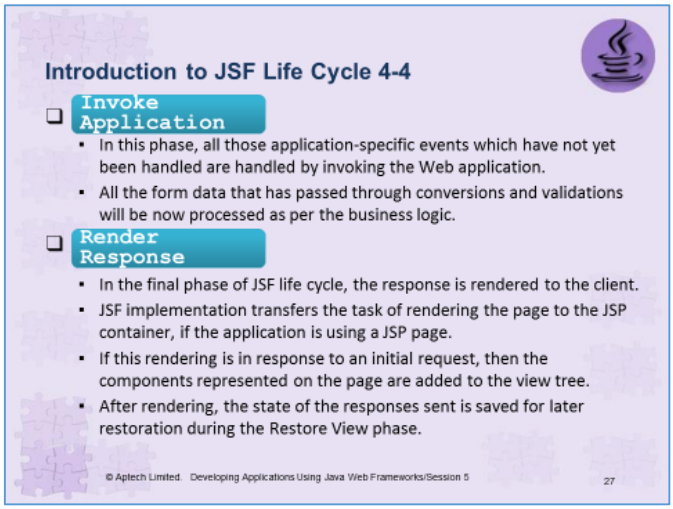
## Slides 24 to 27

Let us understand JSF life cycle.






Use slides 24 to 27 to explain JSF life cycle.

Tell them that **JSF Lifecycle** which dictates the entire flow of events between user requests. For example, upon an initial Web request to access a JSF application, the Faces Controller servlet handles the request by first preparing the JSF context, which is a Java object that holds all application data. The Controller then routes the user to the requested page. The page usually renders application data from the JSF context using a simple Expression Language (EL). Upon subsequent requests, the Controller updates any Model data, providing any new input has been entered. JSF developers have programmatic access to the entire JSF lifecycle at any time during its execution thus affording a high degree of control over the application's behavior at all times.

Tell the students that when a request for a JSF page is made, the JSF implementation starts the **Restore View** phase. In this phase, the JSF implementation builds the view of the page, wires event handlers, and validators to components in the view. It also saves the view in the `FacesContext` instance. The `FacesContext` instance contains all the information needed to process a single request. All the application's component tags, event handlers, converters, and validators have access to the `FacesContext` instance.

In **Apply Request Values** phase, the components are set to their new values, and messages and events is queued.

**Process Validations** phase processes all validators registered on the components in the tree. It examines the component attributes that defines the rules for the validation and compares those rules to the local value stored for the component.

Then, the **Update Model Values** phase determines whether the data is valid or not.

**Invoke Application** phase handles application-level events, such as submitting a form or linking to another page. If the application needs to redirect to a different Web application resource or generate a response that does not contain any JSF components, it can call `FacesContext.responseComplete`.

In the final phase of JSF life cycle, that is, **Render Response**, the response is rendered to the client. JSF implementation transfers the task of rendering the page to the JSP container if the application is using a JSP page. If this rendering is in response to an initial request, then the components represented on the page are added to the view tree. After rendering, the state of the responses sent is saved for later restoration during the Restore View phase.

### In-Class Question:

After you finish explaining JSF life cycle, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.

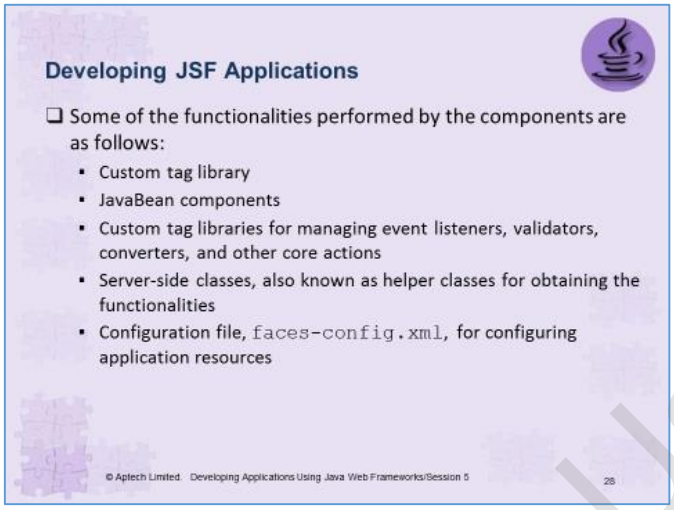
Which phase is started when the application responds to a request coming from FacesServlet controller?

### Answer:

Restore View Phase

## Slide 28

Let us understand developing JSF applications.



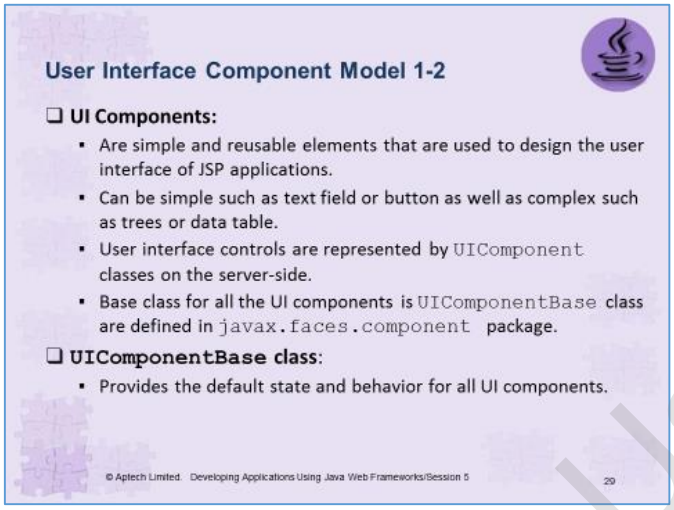

Use slide 28 to explain the steps for developing JSF applications.

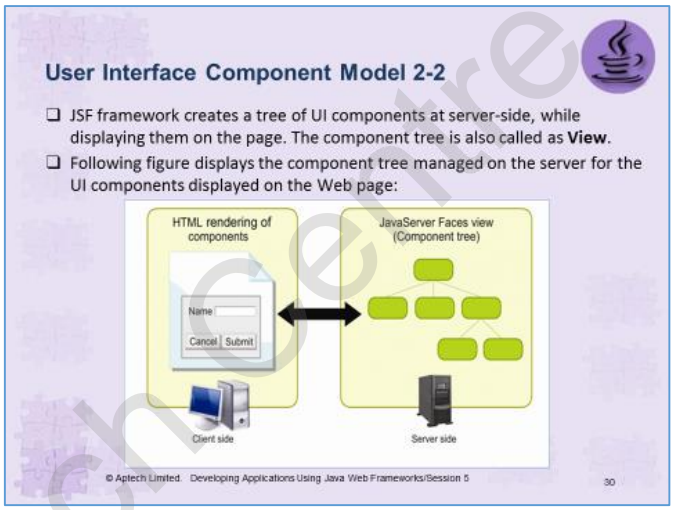
Some of the functionalities performed by the components in a JSF application are as follows:

- Custom tag library used for embedding and rendering UI components within the View page.
- JavaBean components, also known as Backing beans which define the properties and functionalities of UI components.
- Custom tag libraries for managing event listeners, validators, converters, and other core actions.
- Server-side classes, also known as helper classes for obtaining the functionalities such as database access.
- A configuration file, faces-config.xml, for configuring application resources such as JavaBean, page navigation rules, and other custom objects used in the JSF application or applying annotations to the components.

## Slides 29 and 30

Let us understand user interface component model.




Use slides 29 and 30 to explain user interface component model.

Using slide 25, explain that the user interface components are simple and reusable elements that are used to design the user interface of JSP applications. These components can be simple such as text field or button as well as complex such as trees or data table. In JSF, user interface controls are represented by `UIComponent` classes on the server-side. The class defines the behavior of the components in the JSF application environment regardless of how it will be presented to the client. It not only includes the `UIComponent` class, but also the other helper components such as Renderer for rendering the component, Validator for registering validators onto a component, Converter for registering data converters on to a component, and so on.

The base class for all the UI components is `UIComponentBase` class defined in `javax.faces. component` package. The `UIComponentBase` class provides the default state and behavior for all UI components. There is a set of HTML component classes defined in `javax.faces. component.html` package. These components are derived from the `UIComponentBase` class.

The UI components are associated with UI tags used in designing the Web page of the application. The JSF framework creates a tree of UI components at server-side while displaying them on the page. This helps the components to remember the values between the requests forwarded to the server. The component tree is also called as View.

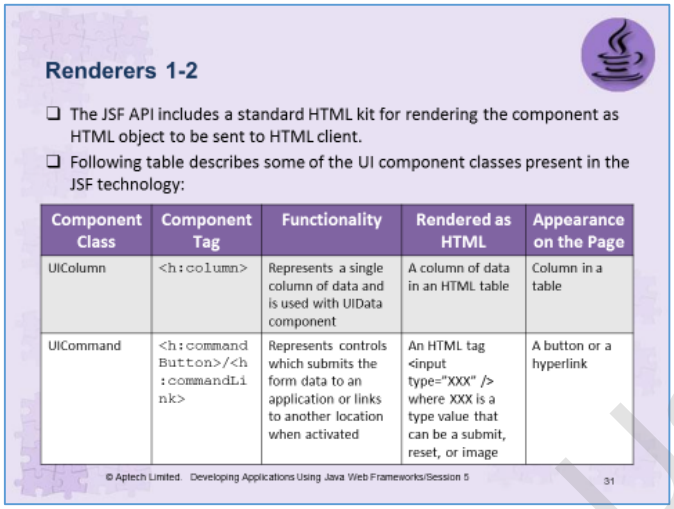
Explain that the component tree managed on the server for the UI components displayed on the Web page as shown on slide 30.

Some of the user interface component class included in JSF are as follows:

- `UICommand`: Represents a control that fires actions when activated.
- `UIData`: Represents a data binding to a collection of data represented by a `javax.faces.model.DataModel` instance.
- `UIGraphic`: Displays an image.
- `UIInput`: Takes data input from a user. This class is a subclass of UIOutput.
- `UIMessage`: Displays a localized error message.
- `UIMessages`: Displays a set of localized error messages.
- `UIOutcomeTarget`: Displays a hyperlink in the form of a link or a button.
- `UIOutput`: Displays data output on a page.
- `UIPanel`: Manages the layout of its child components.
- `UIParameter`: Represents substitution parameters.
- `UISelectBoolean`: Allows a user to set a boolean value on a control by selecting or deselecting it. This class is a subclass of the UIInput class.
- `UISelectItem`: Represents a single item in a set of items.
- `UISelectItems`: Represents an entire set of items.
- `UISelectMany`: Allows a user to select multiple items from a group of items.
- `UISelectOne`: Allows a user to select one item from a group of items.
- `UIViewParameter`: Represents the query parameters in a request.
- `UIViewRoot`: Represents the root of the component tree.

## Slides 31 and 32

Let us understand renderers.

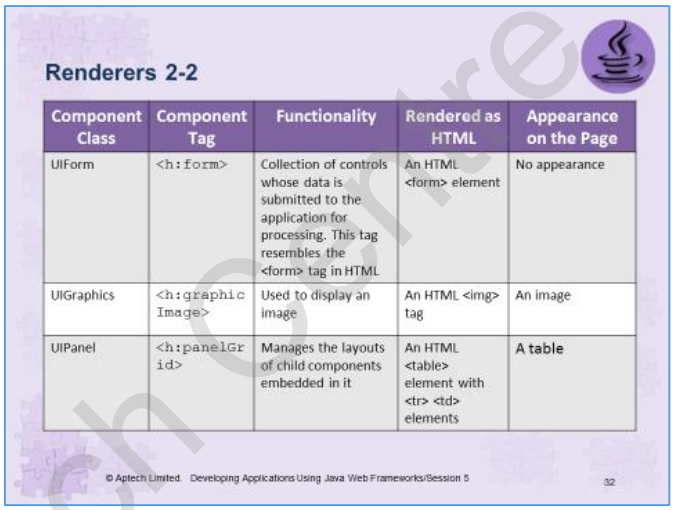


**Renderers 1-2**

- The JSF API includes a standard HTML kit for rendering the component as HTML object to be sent to HTML client.
- Following table describes some of the UI component classes present in the JSF technology:

| Component Class | Component Tag | Functionality | Rendered as HTML | Appearance on the Page |
|---|---|---|---|---|
| UIColumn | `<h:column>` | Represents a single column of data and is used with UIData component | A column of data in an HTML table | Column in a table |
| UICommand | `<h:commandButton>/<h:commandLink>` | Represents controls which submits the form data to an application or links to another location when activated | An HTML tag <input type="XXX" /> where XXX is a type value that can be a submit, reset, or image | A button or a hyperlink |

 31



**Renderers 2-2**

| Component Class | Component Tag | Functionality | Rendered as HTML | Appearance on the Page |
|---|---|---|---|---|
| UIForm | `<h:form>` | Collection of controls whose data is submitted to the application for processing. This tag resembles the <form> tag in HTML | An HTML <form> element | No appearance |
| UIGraphics | `<h:graphicImage>` | Used to display an image | An HTML <img> tag | An image |
| UIPanel | `<h:panelGrid>` | Manages the layouts of child components embedded in it | An HTML <table> element with <tr> <td> elements | A table |

 32

Use slides 31 and 32 to explain renderers.

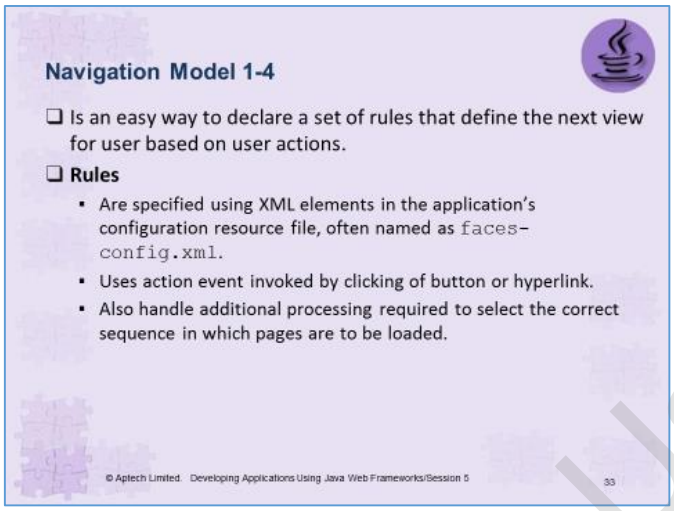
The components in JSF are not responsible for how they will be displayed to the Web client. They are designed to define the functionality of a component whereas the rendering is performed by separate renderer. The rendering model defines the way the components are displayed to the clients. For example, a `UISelectOne` component can be rendered as a set of radio buttons, as a combo box, or as a list box.

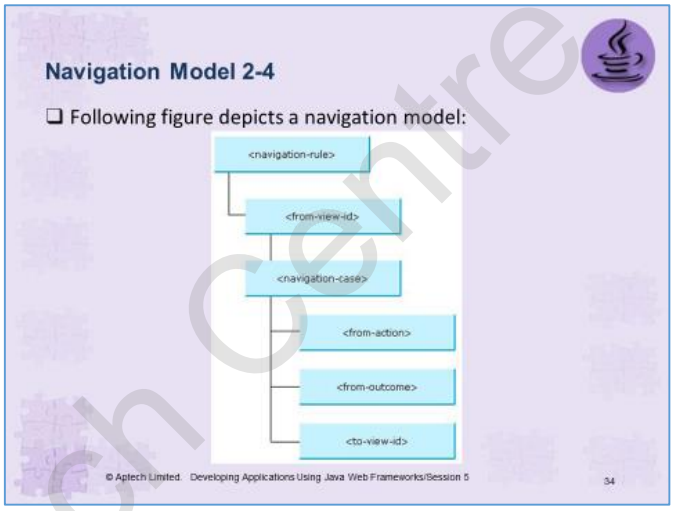
The JSF API includes a standard HTML kit for rendering the component as HTML object to be sent to HTML client.

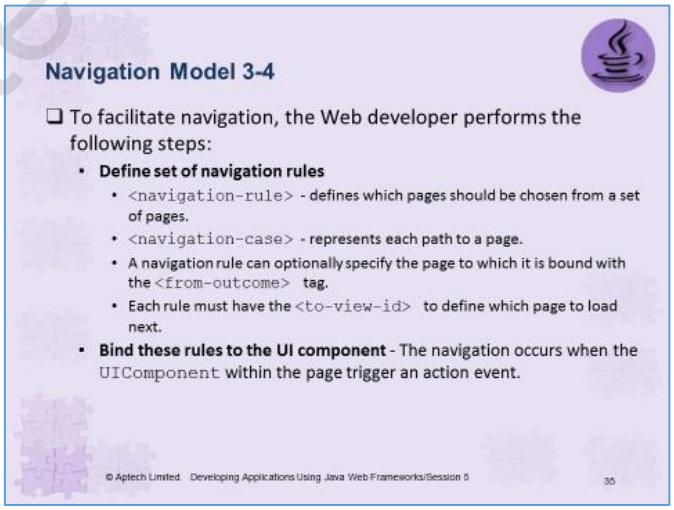
Then, using slides 31 and 32, explain the UI component class with their rendered HTML element and its appearance on the Web page.

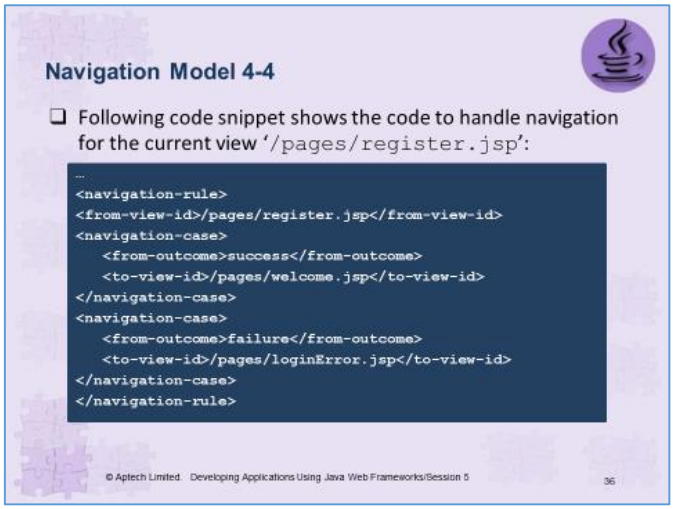
## Slides 33 to 36

Let us understand Navigation Model.






Use slide 33 to 36 to explain Navigation Model.

Tell them that any commercial Web application requires multiple Web pages that together serve the application functionality. A Web developer has to define a correct sequence of Web pages to be loaded based on user's action. This is a daunting task since the functionality is distributed across multiple pages.

Using slide 33, explain that the JSF Navigation model is an easy way to declare a set of rules that define the next view based on the actions. These rules are specified using XML elements in the application's configuration resource file, often named as `faces-config.xml`. The rules uses action event invoked by clicking of button or hyperlink. They can also handle additional processing required to select the correct sequence in which pages are to be loaded.

Explain the figure as given on slide 34 that shows the elements of a Navigational model. Tell them that the `<navigation-rule>` element defines which pages should be chosen from a set of pages. Then, within the `<navigation-rule>` element, the `<navigation-case>` element is given which represents each path to a page. A navigation rule can optionally specify the page to which it is bound with the `<from-outcome>` tag. Each rule must have the `<to-view-id>` to define which page to load next. The navigation case is selected based on logical outcome and/or the expression for action method.

JSF supports two kinds of Navigation models are namely, static and dynamic. Static navigation is useful when the response of the current page is known to the user. Dynamic Navigation is useful when the output of the current page is highly unpredictable and the output depends mainly on the execution of some Business logic.

For example, consider a Login application with `login.jsp` page displaying the input fields. On page submission, the control is transferred either to `success.jsp` or `failure.jsp` page in the application.

The following code snippet shows the configuration of Navigational model in the `faces-config.xml` file:

```
. . .
<navigation-rule>
                        <description></description>
     <from-view-id>/login.jsp</from-view-id>

     <navigation-case>
         <from-outcome>loginSuccess</from-outcome>
         <to-view-id>/loginSuccess.jsp</to-view-id>
     </navigation-case>

      <navigation-case>
         <from-outcome>loginFailure</from-outcome>
         <to-view-id>/loginFailure.jsp</to-view-id>
     </navigation-case>
 </navigation-rule>
. . .
```

Here, we have added two navigation cases, one for success and the other for failure.

Two of the most important changes in JSF 2.0 are the introduction of annotations and the new convention used for navigation. These new features essentially make the `faces-config.xml` file optional. With JSF 2.0, you can use annotations in managed beans, registering listeners, resource rendering, and so on. Now, any annotated POJO can be used as a managed bean and navigation rules can be defined in managed beans.

The runtime will consider annotations only if the `faces-config.xml` is not there or the metadata-complete attribute of the `faces-config.xml file` is not set to **true**.

**Implicit Navigation**

Tell them that JSF also supports implicit navigation. Implicit navigation does not need the explicit navigation rules to be defined in the `faces-config.xml`. Implicit navigation allows the default navigation handler to choose the XHTML page with the matching name as specified in the `action` attribute of the `UIComponent`.

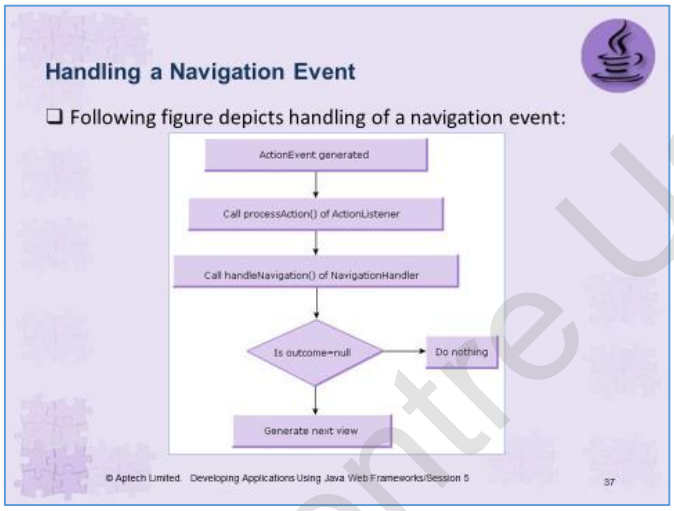
For example, consider that you are using Facelete technology, then to move the navigation control from `welcome.xhtml` to `home.xhtml` is as follows:

```
<h:commandLink value="Next" action="home"/>
```

Thus, the `action` attribute is treated as the `to-view-id` element in the `faces-config.xml`. The default navigation handler appends a '`/`' and `.xhtml` extension. This simplifies the Navigation model in JSF 2.0.

## Slide 37

Let us understand handling a navigation event.

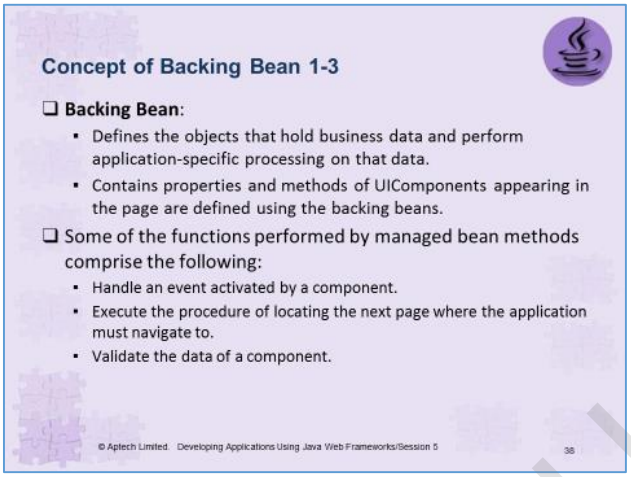


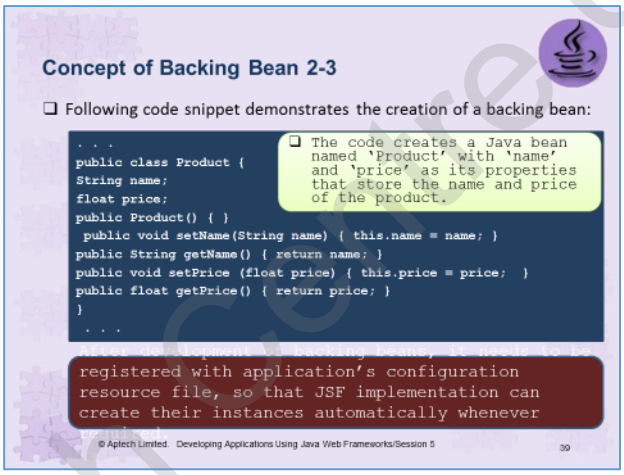

Use slide 37 to explain handling a navigation event.

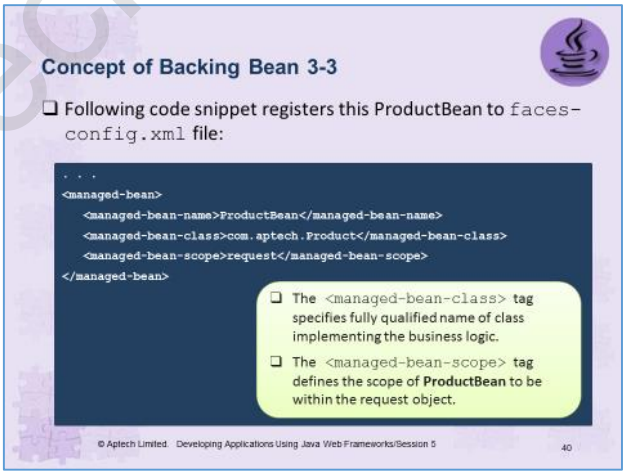
Explain the figure as shown on slide 37 that depicts the flowchart for handling the navigational events.

## Slides 38 to 40

Let us understand the concept of backing bean.




Use slides 38 to 40 to explain the concept of backing bean.

Tell the students that a typical JSF application includes one or more backing beans, each of which is a JSF managed bean that is associated with the UI components used in a particular page.

Backing bean class defines a set of UI component properties and a set of methods that perform functions for a component. Component properties are as follows:

- A component's value
- A component instance
- A converter instance
- A listener instance
- A validator instance

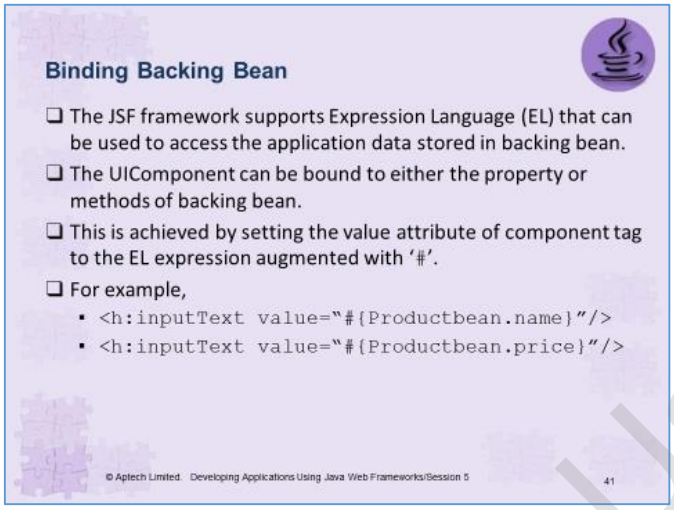
Configuring a Bean consists of the following steps:

1. Configure simple beans and more complex trees of beans.
2. Initialize bean properties with values.
3. Place beans in a particular scope.
4. Expose the beans to the unified EL so that page authors can access them.

Slide 39 shows the code snippet that demonstrates the creation of a backing bean. Then, using slide 40, explain how to register backing bean in the `faces-config.xml`. Tell them that the code is to be written in applications configuration resource file. The <`managed-bean-name`> tag sets the name of backing bean to '**ProductBean**'. The <`managed-bean-class`> tag specifies fully qualified name of class implementing the business logic. The <`managed-bean-scope`> tag defines the scope of **ProductBean** to be within the request object.

## Slide 41

Let us understand binding backing bean.




Use slide 41 to explain binding backing bean.

The JSF framework supports Expression Language (EL) that can be used to access the application data stored in backing bean. Each UI component in the Web page is coupled with the property of associated backing bean. The UIComponent can be bound to either the property or methods of backing bean. This is achieved by setting the value attribute of component tag to the EL expression augmented with '#'.

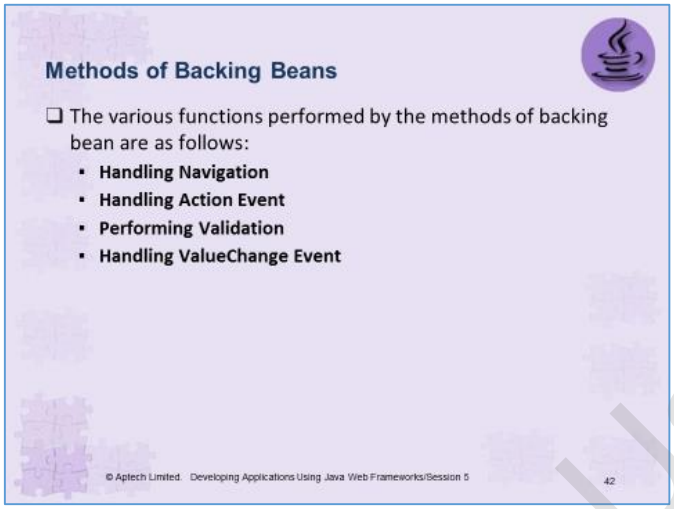
**Tips:**

Each of the managed bean properties can be bound to one of the following:

- A component value
- A component instance
- A converter instance
- A listener instance
- A validator instance

## Slide 42

Let us understand methods of backing beans.



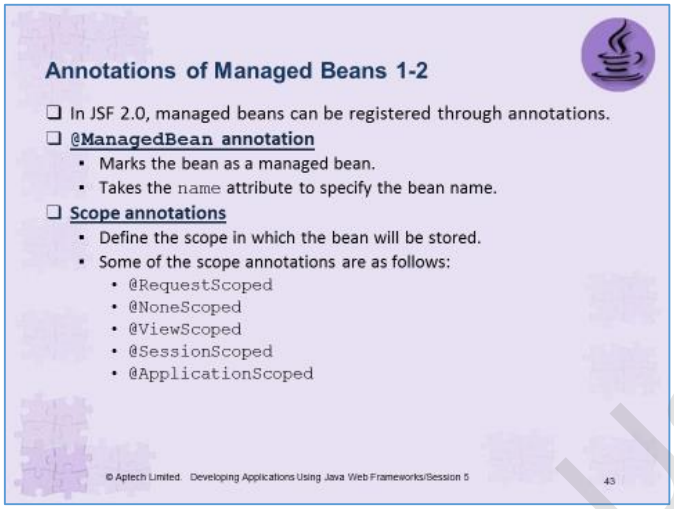

Use slide 42 to explain the methods of Backing Beans.

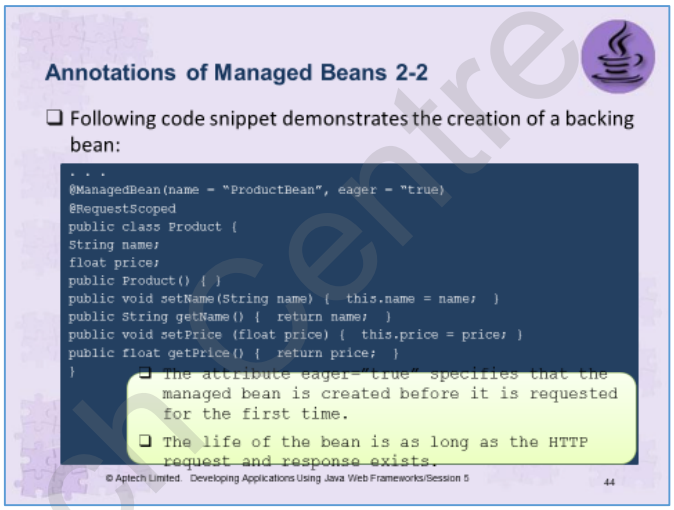
Some other functions that backing bean methods performs are as follows:

- Validating a component's data.
- Handling an event fired by a component.
- Performing processing to determine the next page to which the application must navigate.

## Slides 43 and 44

Let us understand the annotations of managed beans.




Use slides 43 and 44 to explain the annotations of managed beans.

**Scope annotation** - Scope annotation set the scope into which the managed bean is stored. If scope is not given then bean will default to request scope.

Some of the scope annotations are as follows:

- `@RequestScoped` - gets created upon a HTTP request and get destroyed when the HTTP response associated with the HTTP request is finished.
- `@NoneScoped` - gets created upon an EL evaluation and get destroyed immediately after the EL evaluation.
- `@ViewScoped` - gets created upon a HTTP request and get destroyed once user postback to a different view.
- `@SessionScoped` - gets created upon the first HTTP request involving this bean in the session and get destroyed when the HTTP session is invalidated.

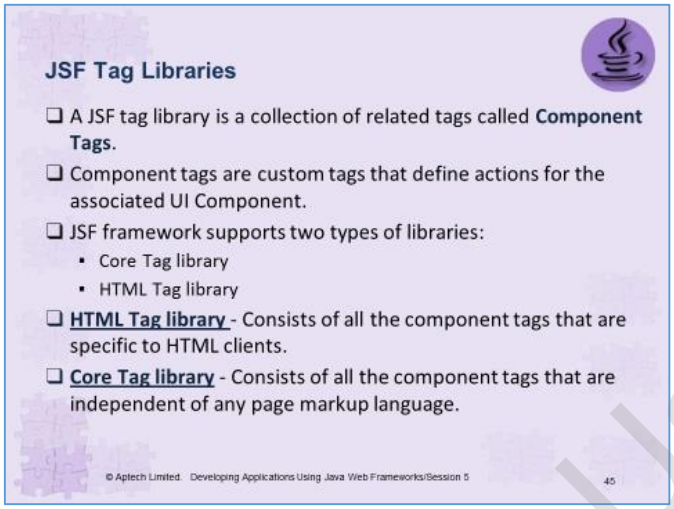
- `@ApplicationScoped` - gets created upon the first HTTP request involving this bean in the application and get destroyed when the Web application shuts down.
- `@CustomScoped` - Bean will be active as long as the bean's entry in the custom Map which is created for this scope is active.

`@ManagedProperty` annotation – With the help of `@ManagedProperty` annotation a managed bean's property can be injected in another managed bean.

Then, using slide 44, explain the code snippet to create a backing bean with annotations. Tell them that the attribute `eager="true"` specifies that the managed bean is created before it is requested for the first time. The life of the bean is as long as the HTTP request and response exists. This means that the bean will be created when the HTTP request is sent and destroyed when the response is received.

## Slide 45

Let us understand JSF tag libraries.




Use slide 45 to explain JSF tag libraries.

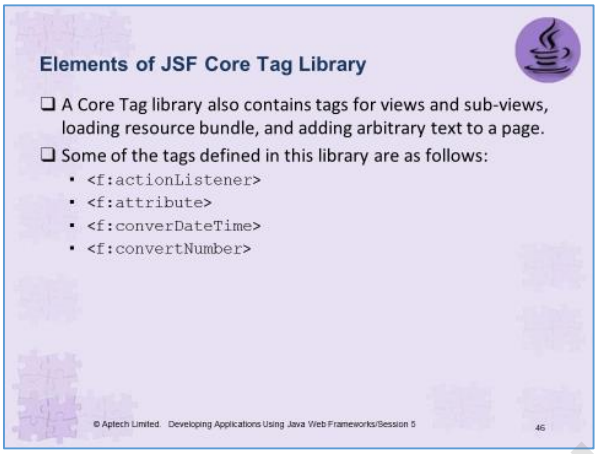
The component tags are custom tags that define actions for the associated UI Component. A JSF-based Web application uses these component tags to interact with JSP pages in the application.

JSF framework supports two types of libraries named Core Tag library and HTML Tag library. The HTML Tag library consists of all the component tags that are specific to HTML clients.

The Core Tag library consists of all the component tags that are independent of any page markup language. These tags allow to take advantages of features of JSF framework, such as validation, conversion, and event handling.

## Slide 46

Let us understand the elements of JSF core tag library.




Use slide 46 to explain the elements of JSF core tag library. Core tag library also contains tags for views and sub-views, loading resource bundle, and adding arbitrary text to a page. All the action elements in the core tag library are commonly prefixed with '`f`'.

A JSF-based Web application must import the core tag library before it can use its custom tags by specifying a `taglib` directive at the top of JSP file. The value of `uri` attribute of `taglib` directive denotes the reference path to Sun Website from where the JSF Core library can be accessed. The value of prefix attribute indicates that prefix '`f`' will be added to refer to each component tag under this library.

Then, explain some of the tags defined in the core tag library as shown on slide 46.

### Tips:

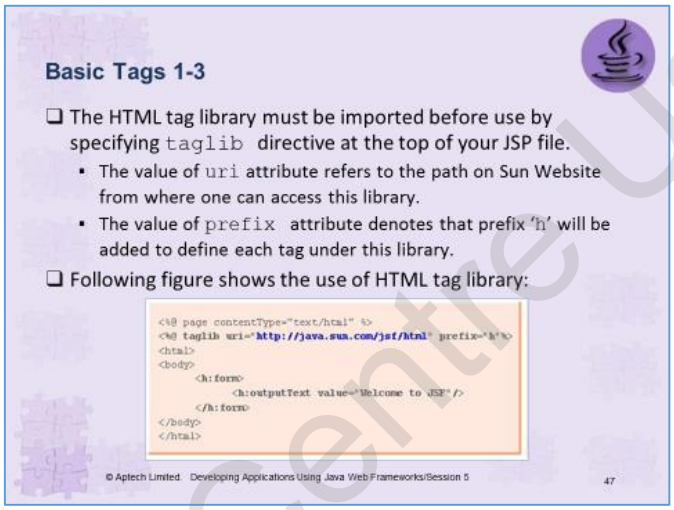
Some of the most commonly used JSF core tags are as follows:

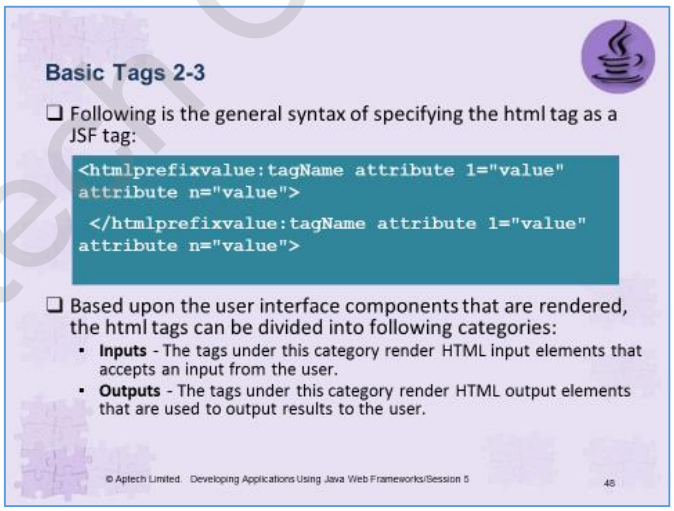
- `valueChangeListener` - Registers a value-change listener on a parent component.
- `converter` - Registers an arbitrary converter on the parent component.
- `facet` - Signifies a nested component that has a special relationship to its enclosing tag.
- `loadBundle` - Specifies a ResourceBundle that is exposed as a Map.
- `param` - Substitutes parameters into a `MessageFormat` instance and adds query string name-value pairs to a URL.
- `selectItem` - Represents one item in a list of items in a `UISelectOne` or `UISelectMany` component.
- `selectItems` - Represents a set of items in a `UISelectOne` or `UISelectMany` component.
- `subview` - Contains all JSF tags in a page that is included in another JSP page containing JavaServer Faces tags.

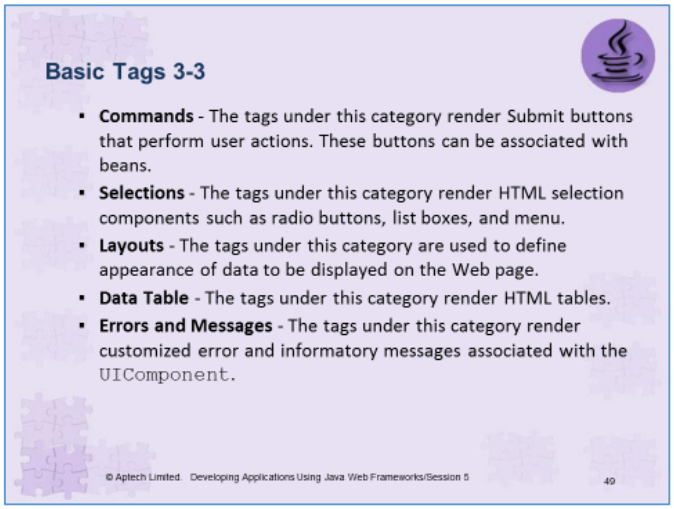
- `validateDoubleRange` - Registers a `DoubleRangeValidator` on a component.
- `validateLength` - Registers a `LengthValidator` on a component.
- `validateLongRange` - Registers a `LongRangeValidator` on a component.
- `validator` - Registers a custom validator on a component.

### Slides 47 to 49

Let us understand basic tags.






Use slides 47 to 49 to explain basic tags.

The HTML tag library must be imported before use by specifying `taglib` directive at the top of your JSP file. The value of `uri` attribute refers to the path on Sun Web site from where one can access this library. The value of `prefix` attribute denotes that prefix 'h' will be added to define each tag under this library.

Some important HTML tags are as follows:

- `Column` - used as a child of the dataTable tag, to represent a column of data.
- `inputHidden` - displays as an HTML input tag with its type set to 'hidden'.
- `commandButton` - displays as an HTML input element.
- `commandLink` - displays as an HTML an element.
- `dataTable` - displays as an HTML table element. It has as its children `h:column` entities, which describe the columns of the table.
- `form` - displays an HTML form element.
- `graphicImage` - displays an HTML img element.
- `inputSecret` - displays as an HTML input tag with its type set to 'password'.
- `inputText` - displays an HTML input element.
- `inputTextarea` - displays an HTML textarea element.
- `message` - displays the first FacesMessage that is assigned to the component referenced by the 'for' attribute.
- `messages` - provides all or some FacesMessages depending on the 'for' and 'globalOnly' attributes.

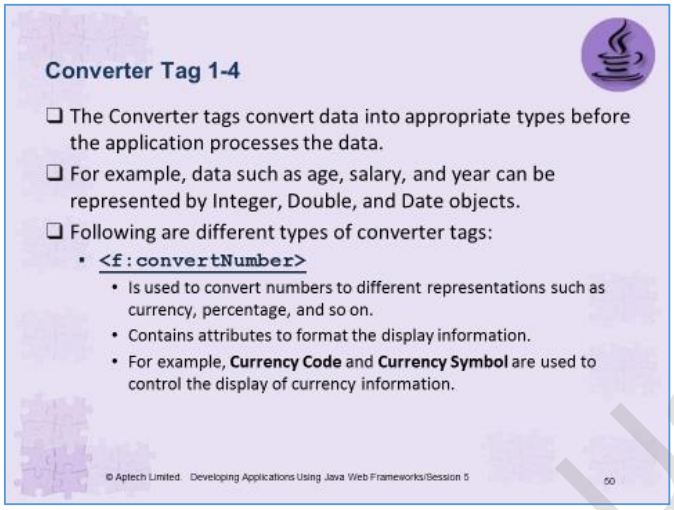
**Tips:**

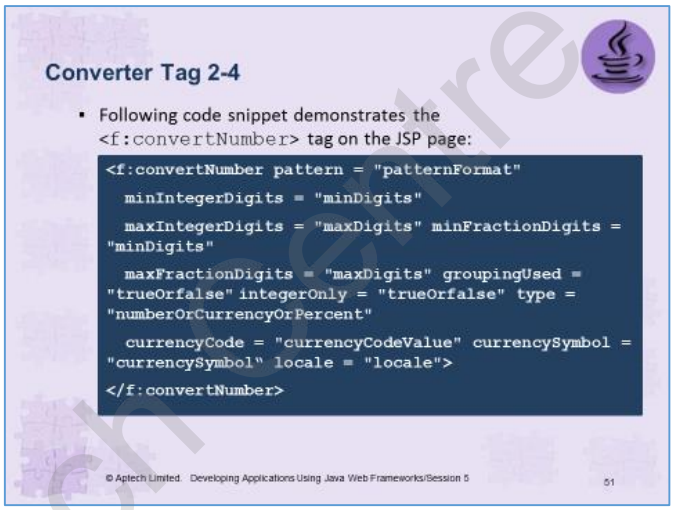
Refer to the following link for more information on core and HTML tags:

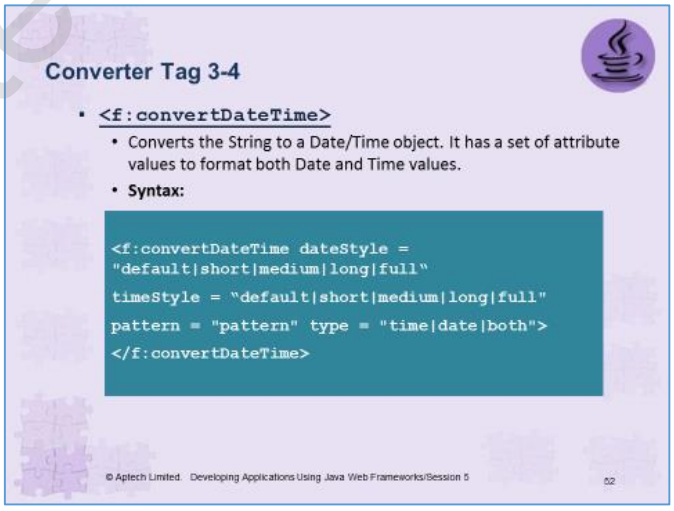
http://www.horstmann.com/corejsf/jsf-tags.html

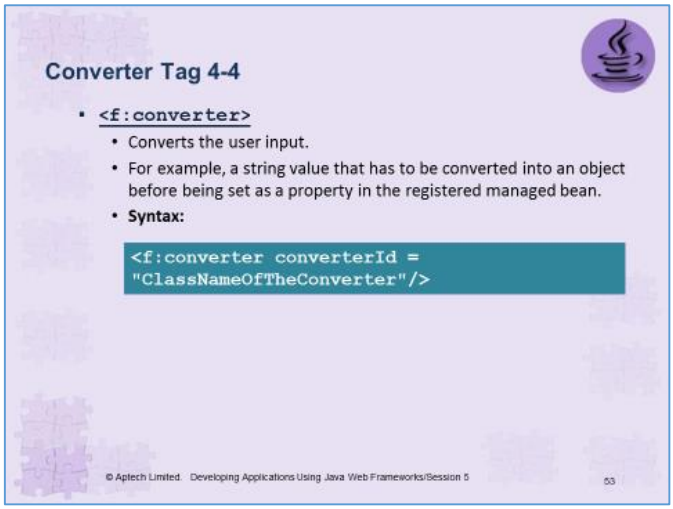
## Slides 50 to 53

Let us understand converter tag.






Use slides 50 to 53 to explain converter tag.

JSF tag library has a collection of pre-built converters to convert string to int, float, Boolean, and so on. Converter tags converts data into appropriate types before the application processes the data.

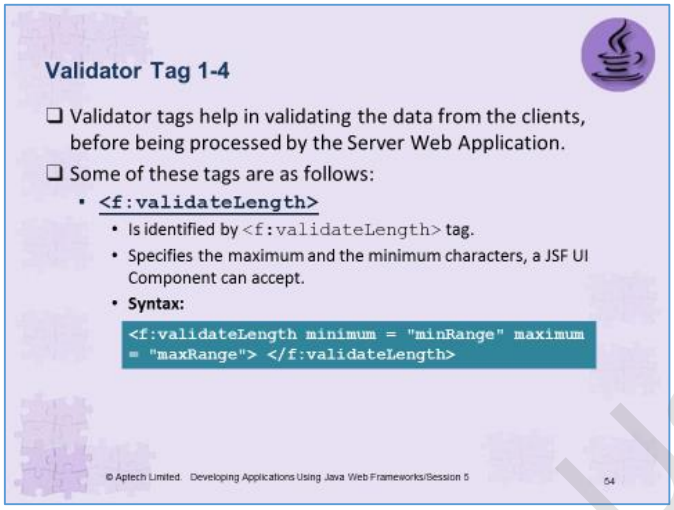
Explain the converter tags with the examples as mentioned on slides 50 to 53.

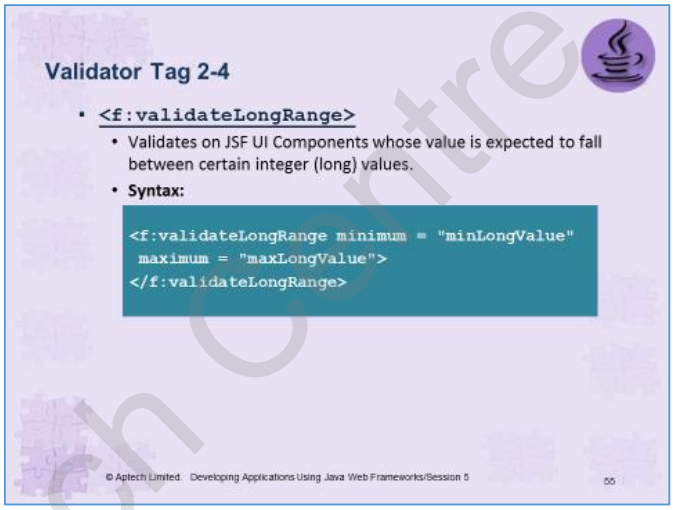
**Tips:**

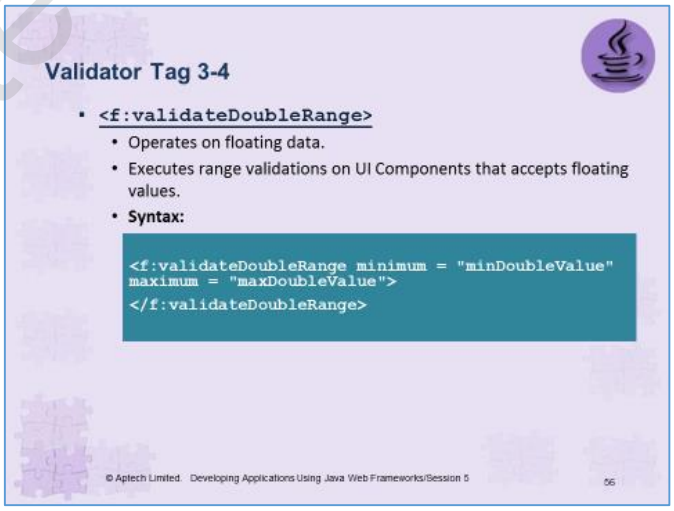
Refer to the following link for more information on converter tags:
http://www.javabeat.net/converter-tags-in-jsf/

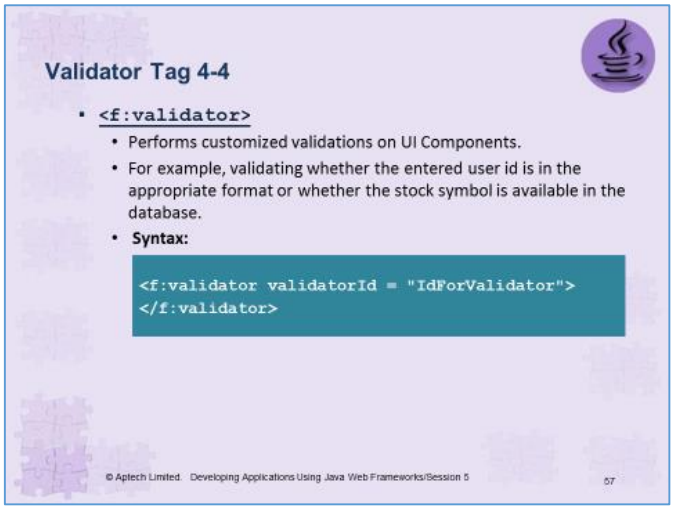
## Slides 54 to 57

Let us understand validator tag.






Use slides 54 to 57 to explain validator tags.

In JSF framework, binding between JSF UI components and validation logic is done with the help of Validator tags. Validator tags helps in validating the data from the clients, before being processed by the server Web application.

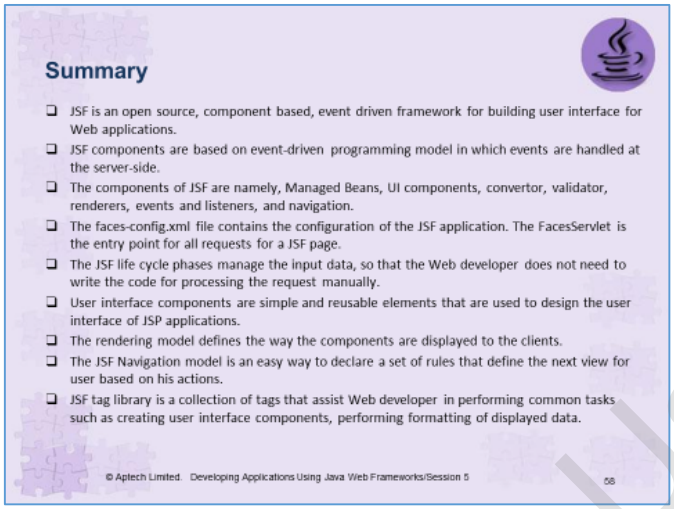
Then, explain the validator tags as mentioned on slides 54 to 57.

**Tips:**

Refer to the following link to get more information on validator tags:
http://java.dzone.com/articles/jsf-validation-tutorial-error

## Slide 58

Let us summarize the session.




In slide 58, you will summarize the session. You will end the session, with a brief summary of what has been taught in the session.

### 5.3 Post Class Activities for Faculty

You should familiarize yourself with the topics of the next session. You should also explore the JSF Expression Language and Faceletes that are offered with the next session.

**Tips:**

You can also check the Articles/Blogs/Expert Videos uploaded on the OnlineVarsity site to gain additional information related to the topics covered in the next session. You can also connect to online tutors on the OnlineVarsity site to ask queries related to the sessions.

# Session 6 – Expression Language, Facelets, and Data Table

## 6.1 Pre-Class Activities

Before you commence the session, you should familiarize yourself with the topics of this session in-depth. You should revisit topics of the previous session for a brief review.

Here, you can ask students the key topics they can recall from previous session. Prepare a question or two which will be a key point to relate the current session objectives.

### 6.1.1 Objectives

By the end of this session, the learners will be able to:

- Describe the features of JSF 1.2 and JSF 2.0
- Explain Expression language and its types in JSF
- Explain the use of facelets View in JSF 2.0
- Explain JSF 2.0 DataTables
- Explain JSF 2.0 Event Handling
- Explain the code to integrate JSF 2.0 with JDBC API

### 6.1.2 Teaching Skills

To teach this session successfully, you should be aware of the features of JSF 1.2 and JSF 2.0.

Familiarize yourself with the Expression language and its types in JSF. You should also know about the use of facelets View in JSF 2.0 and also about JSF 2.0 DataTables.

Aware yourself with JSF 2.0 Event Handling and the code to integrate JSF 2.0 with JDBC API.

For teaching in the class, you are expected to use slides and LCD projectors.

**Tips:**

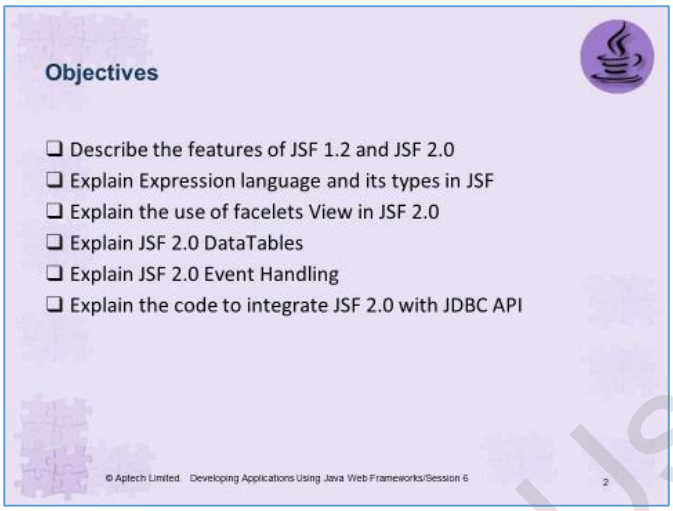
It is recommended that you test the understanding of the students by asking questions in between the class.

**In-Class Activities:**

Follow the order given here during In-Class activities.

### Overview of the Session:

Give the students a brief overview of the current session in the form of session objectives. Show the students slide 2 of the presentation.



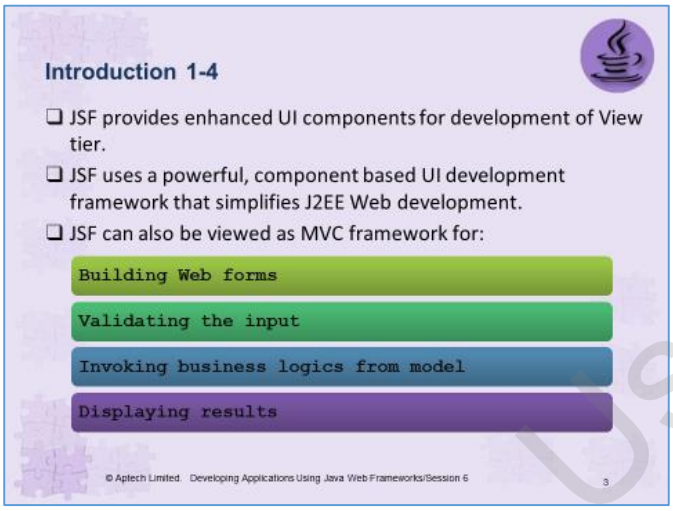

Tell the students that they will be introduced to the features of JSF 1.2 and JSF 2.0. The session explains Expression language (EL) and its types in JSF.

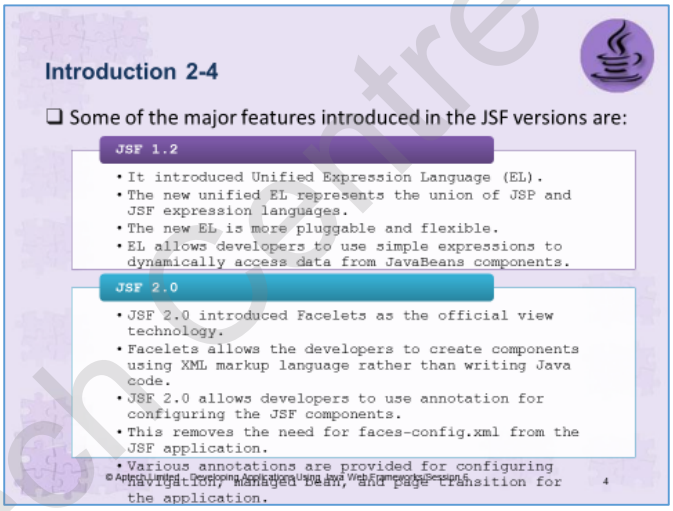
They will further learn how to use Facelets, DataTables, and event handling in JSF. Explain JSF 2.0 Event Handling. Finally, they will learn how to integrate JSF 2.0 code with JDBC API.

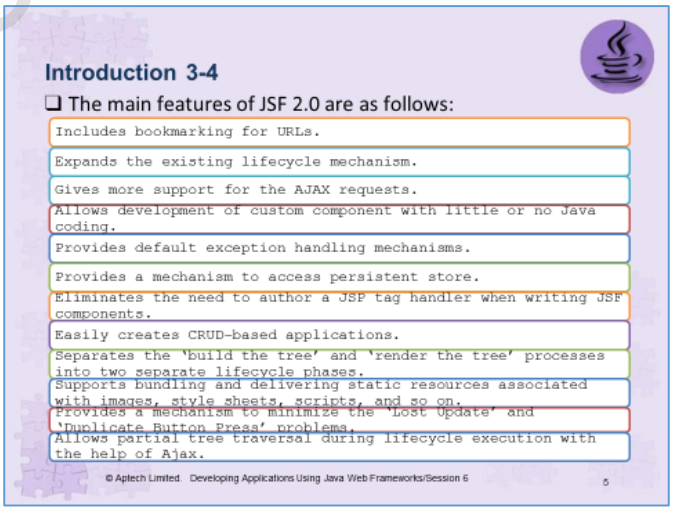
## 6.2 In-Class Explanations

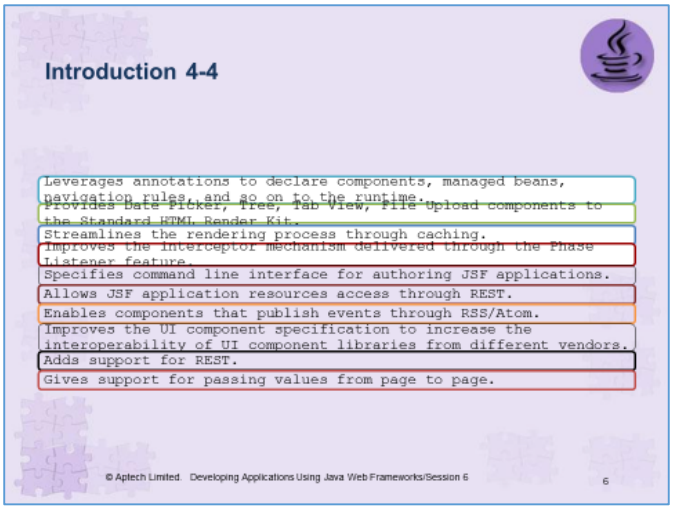
### Slides 3 to 6

Let us understand the major features introduced in the JSF versions.






Use slides 3 to 6 to explain the major features introduced in the JSF versions.

Summarize JSF in two major revisions that are as follows:

**JSF 1.2**

JSF 1.2 introduced Unified Expression Language (EL). The new unified EL introduced in JSF 1.2 represents the union of JSP and JSF expression languages. This new EL is more pluggable and flexible. EL allows developers to use simple expressions to dynamically access data from JavaBeans components.

**JSF 2.0**

JSF 2.0 introduced Facelets as the official view technology. Facelets allow the developers to create components using XML markup language rather than writing Java code. JSF 2.0 allows developers to use annotation for configuring the JSF components. This removes the need for faces-confi.xml from the JSF application. Various annotations are provided for configuring navigation, managed bean, and page transition for the application.

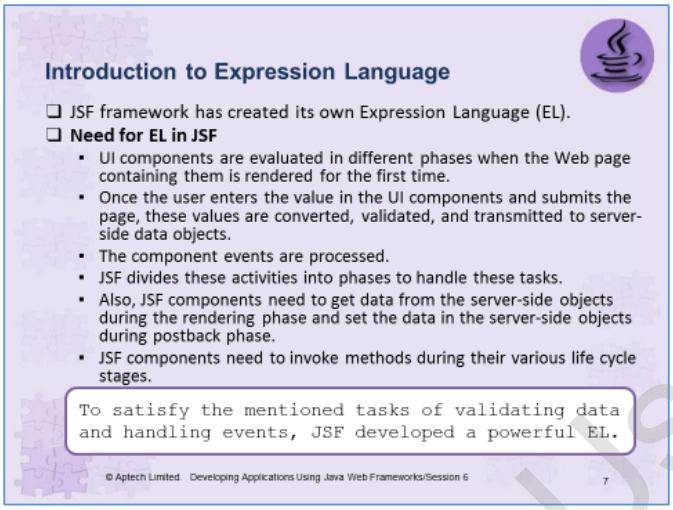
The main features of JSF 2.0 are as follows:

- Includes bookmarking for URLs.
- Expands the existing life cycle mechanism.
- Gives more support for the AJAX requests.
- Use annotations instead of faces-config xml.
- Allows development of custom component with little or no Java coding.
- Provides default exception handling mechanisms. With this, all runtime errors will be forwarded to an error page.
- Provides a mechanism to access persistent store.
- Eliminates the need to author a JSP tag handler when writing JSF components.
- Easily creates CRUD-based applications.
- Separates the ‘build the tree’ and ‘render the tree’ processes into two separate life cycle phases.

- Supports bundling and delivering static resources associated with images, stylesheets, scripts, and so on.
- Provides a mechanism to minimize the 'Lost Update' and 'Duplicate Button Press' problems.
- Allows partial tree traversal during life cycle execution with the help of Ajax.
  Leverages annotations to declare components, managed beans, navigation rules, and so on to the runtime.
- Provides Date Picker, Tree, Tab View, and File Upload components to the Standard HTML Render Kit.
- Streamlines the rendering process through caching.
- Improves the interceptor mechanism delivered through the Phase Listener feature.
- This helps the developer to control the exact requests that are allowed to be processed by each Phase Listener instance.
- Specifies command line interface for authoring JSF applications.
- Allows JSF application resources access through REST.
- Enables components that publish events through RSSAtom.
- Improves the UI Component specification to increase the interoperability of UI component libraries from different vendors.
- Adds support for REST.
- Gives support for passing values from page to page.

**Slide 7**

Let us understand Expression Language.




Use slide 7 to explain Expression Language. Tell the students that EL allows the page authors to use simple expressions for dynamically accessing data from JavaBeans components.

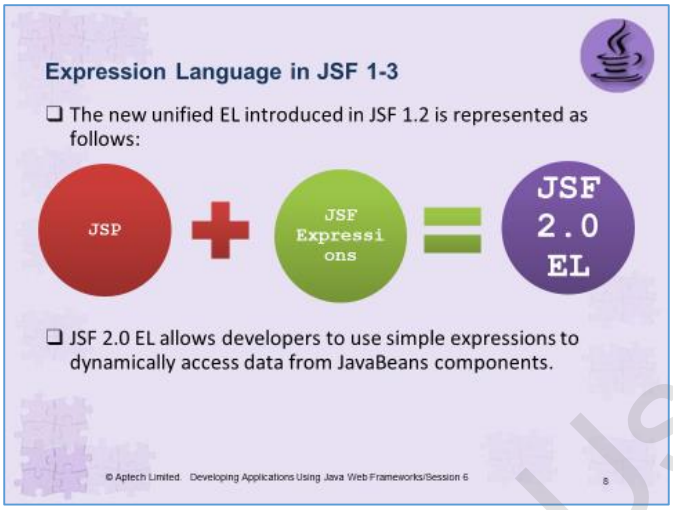
JSF, being a UI framework, needed its own expression language. This expression language when used with JSP tags had its own limitations. Hence, the specification writers and experts have developed a new unified Expression Language in JSF 1.1, which aligned these technologies by adopting the features offered by JSF. The use of this new EL increased the productivity and made the maintenance easier.

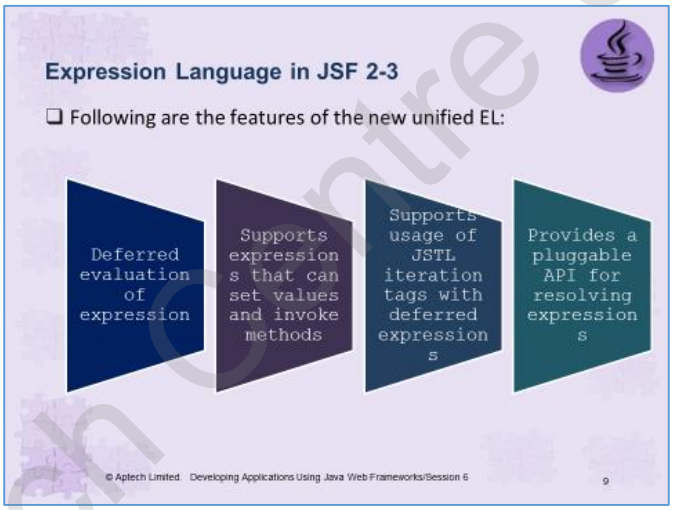
Then, discuss the need for using EL in JSF as mentioned on slide 7.

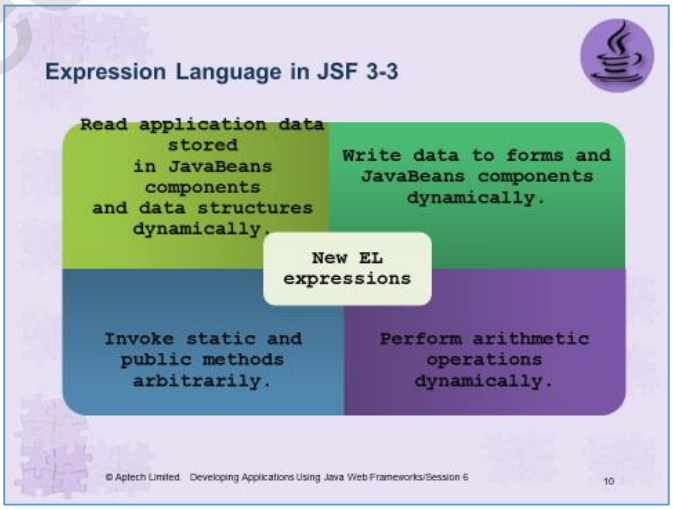
The new unified EL introduced in JSF 1.2, which represents the union of JSP and JSF expression languages. The same EL is also used in JSF 2.0.

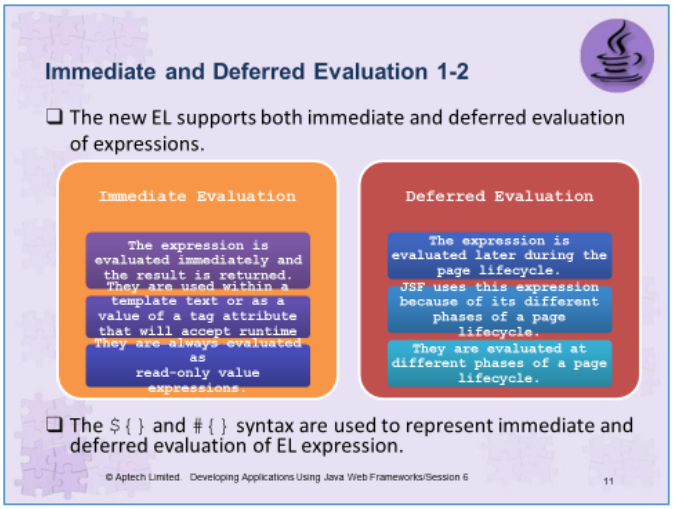
**Slides 8 to 12**

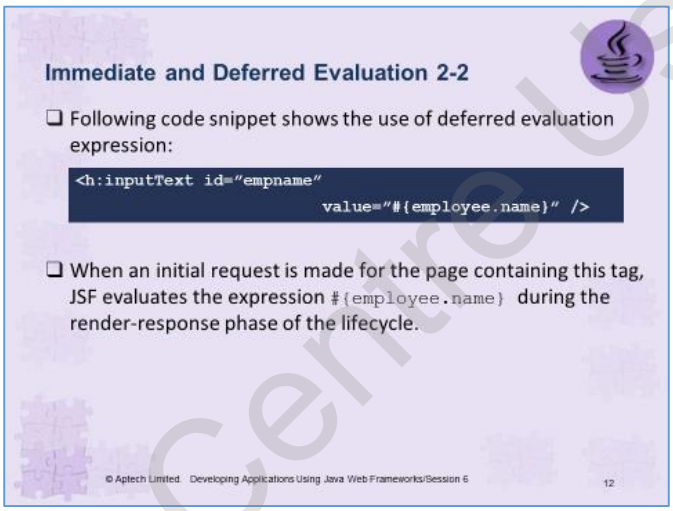
Let us understand Expression Language in JSF.






Use slides 8 to 12 to explain Expression Language in JSF.

Tell the students that EL is used for the following functions in JSF:

- Deferred and immediate evaluation of expressions.
- The ability to set as well as get data.
- The ability to invoke methods.
- Dynamically read application data stored in JavaBeans components, various data structures, and implicit objects.
- Dynamically write data, such as user input into forms, to JavaBeans components.
- Invoke arbitrary static and public methods.
- Dynamically perform arithmetic operations.

Use slides 11 and 12 to explain immediate and deferred evaluation. Tell the students that JSF mostly uses deferred evaluation expressions because of its multiphase lifecycle. During the lifecycle, component events are handled, data is validated, and other tasks are performed in a particular order. So JSF implementation

should defer evaluation of expressions until the appropriate point in the lifecycle. Deferred evaluation expressions can be either value expressions or method expressions. Further, tell them that the new EL supports both immediate and deferred evaluation of expressions.

**Immediate Evaluation**

In immediate evaluation, the expression is evaluated immediately and the result is returned.

The `${}` syntax is used to represent immediate evaluation of EL expression.

Immediate evaluation expressions are used within a template text or as a value of a tag attribute that will accept runtime expressions. They are always evaluated as read-only value expressions.

For example, the tag whose value attribute references an immediate evaluation expression that gets the total price from the session-scoped bean named cart is as follows:

```
<fmt:formatNumber value="${sessionScope.cart.total}"/>
```

The JSF implementation evaluates the expression `${sessionScope.cart.total}`, converts it, and passes the returned value to the tag handler.

Immediate evaluation expressions are always read-only value expressions. This means in the given example expression cannot set the total price, but instead can only get the total price from the cart bean.

**Deferred Evaluation**

In deferred evaluation, the expression is evaluated later during the page life cycle. JSF uses deferred evaluation expressions because of its different phases of a page life cycle.

The `#{}` syntax is used to represent immediate evaluation of EL expression.

Deferred evaluation expressions are evaluated at different phases of a page life cycle. In JSF, the controller evaluates the expression at different phases of the page life cycle, depending on how the expression is used in the page.

For example, the following example shows a JSF `h:inputText` tag, which represents a text field component into which a user enters a value.

```
<h:inputText id="name" value="#{customer.name}" />
```

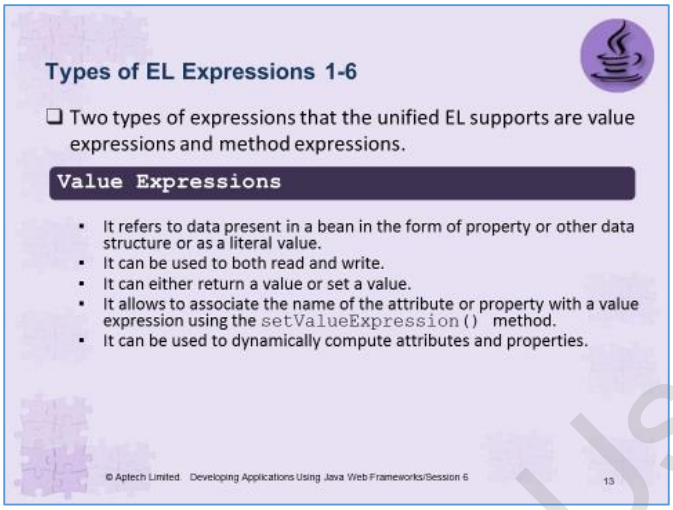
The `h:inputText` tag's value attribute references a deferred evaluation expression that points to the `name` property of the customer bean.

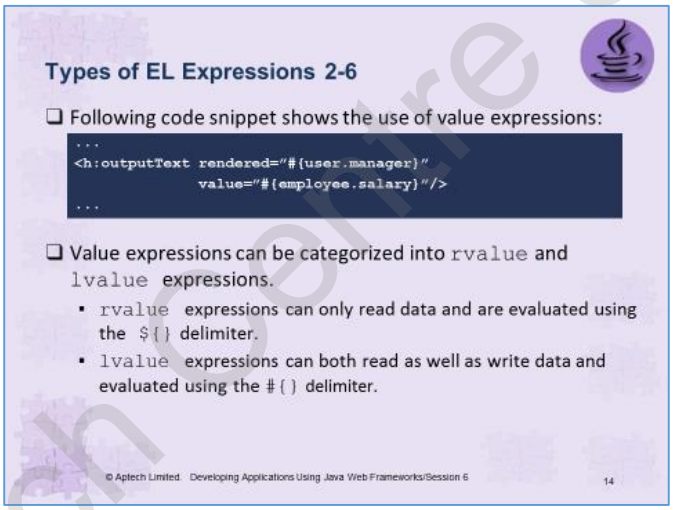
For an initial request of the page containing this tag, the JSF implementation evaluates the `#{customer.name}` expression during the render-response phase of the life cycle. During this phase, the expression merely accesses the value of name from the customer bean, as is done in immediate evaluation.

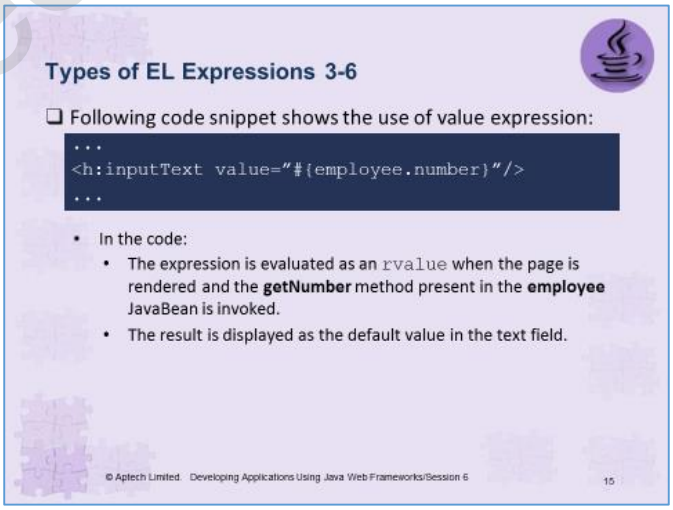
A deferred expression can be a value expression and method expression.

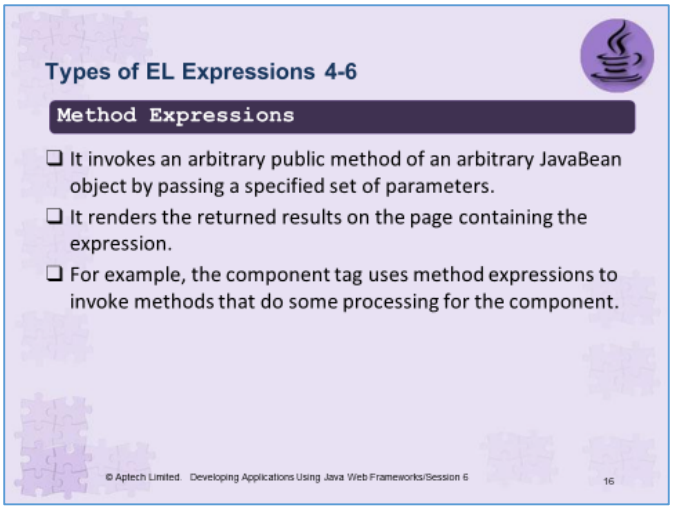
**Slides 13 to 18**

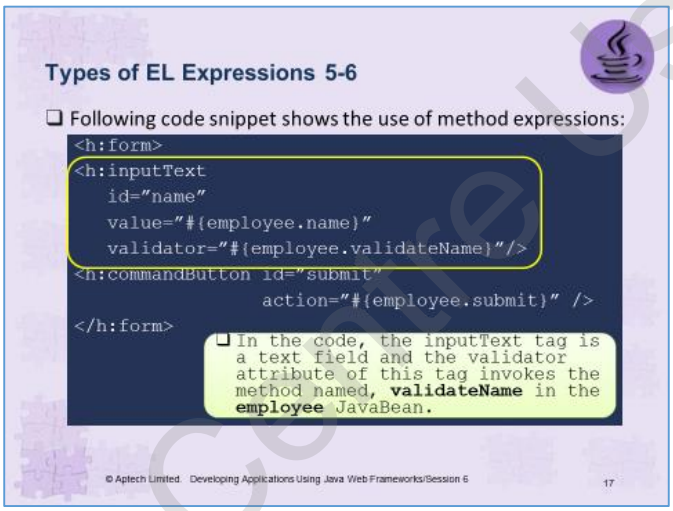
Let us understand the types of EL expressions.

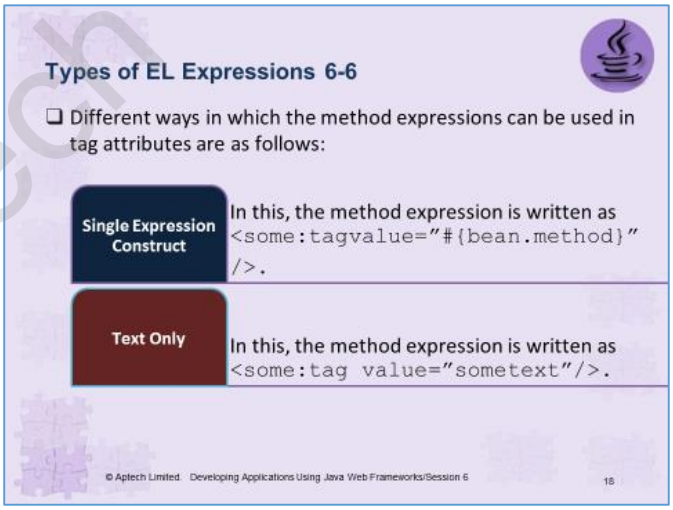





Use slides 13 to 18 to explain the different types of EL Expressions.

- **Value expressions** – It can either yield a value or set a value. Value expressions are divided into rvalue and lvalue expressions. Rvalue expressions can read data but cannot write it. Lvalue

expressions can both read and write data. Both rvalue and lvalue expressions can refer to the following objects and their properties or attributes:

- JavaBeans components
- Collections
- Java SE enumerated types
- Implicit objects

All expressions that are evaluated immediately use the `${}` delimiters and are always rvalue expressions. Expressions whose evaluation can be deferred use the `#{}` delimiters and can act as both rvalue and lvalue expressions.

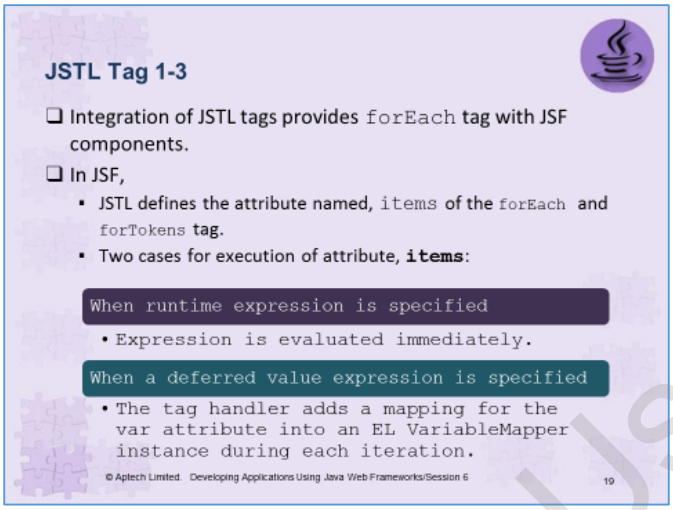
Using slides 15 and 16, explain the code snippets to work with the value expression.

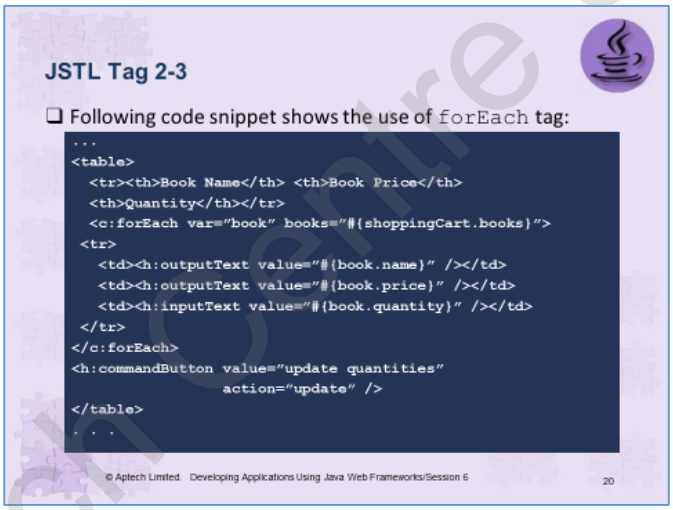
- **Method expressions** - It references methods that can be invoked and can return a value. They can be used only in tag attributes.

Using slides 17 and 18, explain the method expressions and the code snippet to use the method expressions.

**Slides 19 to 21**

Let us understand JSTL tag.



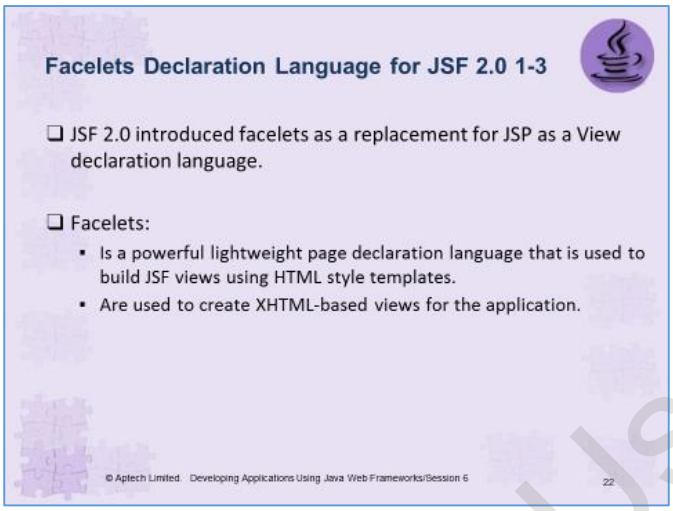



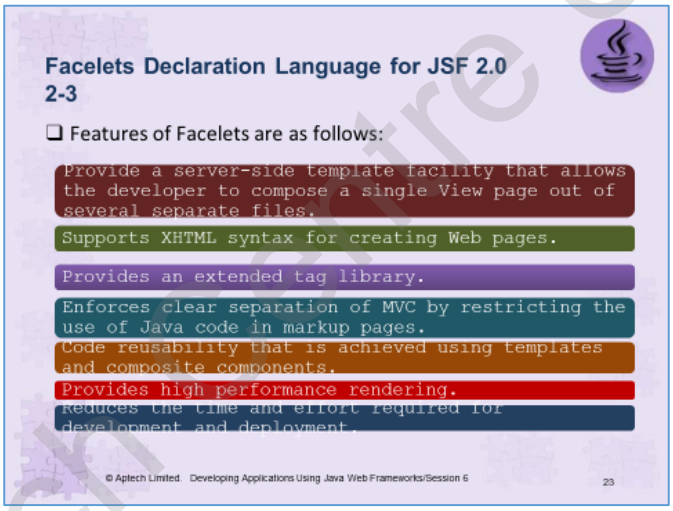


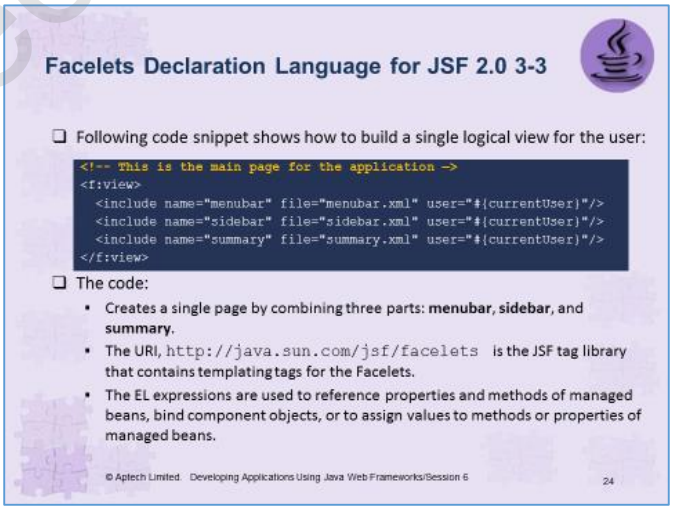
Use slides 19 to 21 to explain JSTL tag.

**Slides 22 to 24**

Let us understand facelets declaration language for JSF 2.0.




Use slides 22 to 24 to explain facelets declaration language for JSF 2.0.

JSF 2.0 introduced facelets as a replacement for JSP as a View declaration language. Facelet is a powerful lightweight page declaration language that is used to build JSF views using HTML style templates. In other words, Facelets are used to create XHTML-based views for the application.

The features of Facelets are as follows:

- Provides a server-side template facility that allows the developer to compose a single view page out of several separate files.
- Supports XHTML syntax for creating Web pages.
- Provides an extended tag library.
- Enforces clear separation of MVC by restricting the use of Java code in markup pages.
- Code reusability that is achieved using templates and composite components.
- Provides high performance rendering.
- Reduces the time and effort required for development and deployment.
- Code reusability that is achieved using templates and composite components.
- Provides high performance rendering.
- Reduces the time and effort required for development and deployment.

**Tips:**

Some of the advantages of using Facelets are as follows:

- Facelets allow the developer to create templates and composite components. This in turn, allows code reuse. Templating is the process where the developer defines the look and feel of the Web application. This skeletal definition of the Web page is known as a template. Developers can reuse these templates and composite components for developing other Web pages.

- A composite component also comprises other components and tags which can later be reused in the Web pages.

- JSF allows for content reuse through referencing a file or through defining custom tags.

- The compilation tie for Facelets is less and it allows compile tie validation of EL expressions.

- Facelets are independently rendered and are compatible with any render kit.

Then, discuss the code snippet as shown on slide 24 to demonstrate how a single page development can be done by designing single logical view. Tell them this is just a pseudo-code for a markup language.

Tell them that the code snippet create a single page by combining three parts: menubar, sidebar, and summary. This can be reused in the multiple pages and can be further customized by using the EL. The similar composition of pages can be performed using Facelets in JSF.

**In-Class Question:**

After you finish explaining facelets declaration language for JSF 2.0., you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.

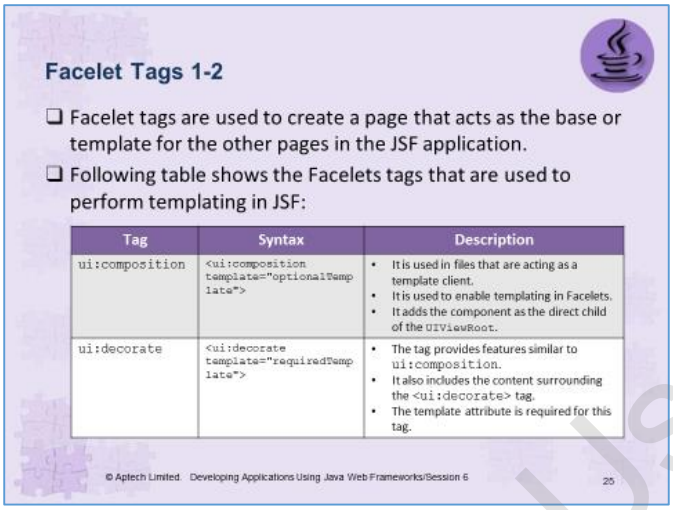
Which page declaration language is used to build JSF views using HTML style templates and to build component trees?

**Answer:**

Facelets

**Slides 25 and 26**

Let us understand Facelet tags.

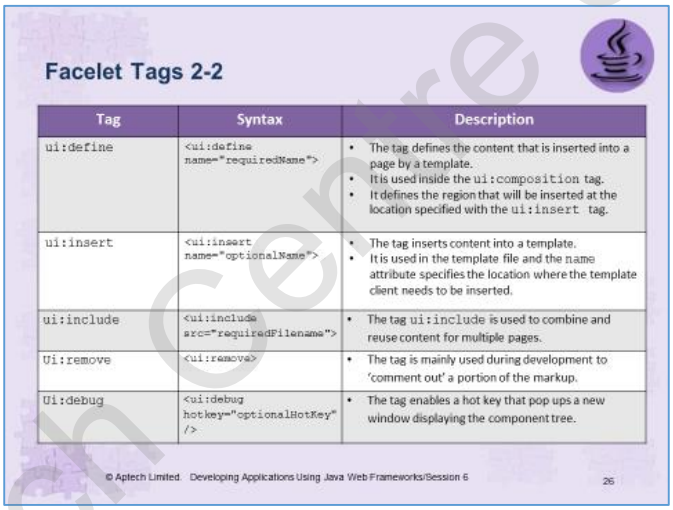


**Facelet Tags 1-2**

- Facelet tags are used to create a page that acts as the base or template for the other pages in the JSF application.
- Following table shows the Facelets tags that are used to perform templating in JSF:

| Tag | Syntax | Description |
|---|---|---|
| ui:composition | <ui:composition template="optionalTemplate"> | • It is used in files that are acting as a template client.<br>• It is used to enable templating in Facelets.<br>• It adds the component as the direct child of the UIViewRoot. |
| ui:decorate | <ui:decorate template="requiredTemplate"> | • The tag provides features similar to ui:composition.<br>• It also includes the content surrounding the <ui:decorate> tag.<br>• The template attribute is required for this tag. |

 25



**Facelet Tags 2-2**

| Tag | Syntax | Description |
|---|---|---|
| ui:define | <ui:define name="requiredName"> | • The tag defines the content that is inserted into a page by a template.<br>• It is used inside the ui:composition tag.<br>• It defines the region that will be inserted at the location specified with the ui:insert tag. |
| ui:insert | <ui:insert name="optionalName"> | • The tag inserts content into a template.<br>• It is used in the template file and the name attribute specifies the location where the template client needs to be inserted. |
| ui:include | <ui:include src="requiredFilename"> | • The tag ui:include is used to combine and reuse content for multiple pages. |
| Ui:remove | <ui:remove> | • The tag is mainly used during development to 'comment out' a portion of the markup. |
| Ui:debug | <ui:debug hotkey="optionalHotKey" /> | • The tag enables a hot key that pop ups a new window displaying the component tree. |

 26

Use slides 25 and 26 to explain facelet tags.

The URI, `http://java.sun.com/jsf/facelets` is the JSF tag library that contains templating tags for the Facelets. To use these tags, the developer must prefix `ui:` on the page.

For example, the following code snippet shows the use of the namespace for using Faceletes:

```
<html
   xmlns="http://www.w3.org/1999/xhtml"
   xmlns:ui="http://java.sun.com/jsf/facelets"
>
```

For example, `ui:insert` or `ui:component`. The facelets library also support tags for composite components.

Facelets support EL expressions to reference properties and methods of managed beans, bind component objects, or to assign values to methods or properties of managed beans.

Then, discuss the Faceletes tags that are used to perform templating as listed on slides 25 and 26.

Tell them to create a template, you can use these tags:

- `<ui:insert>` - Used in template file. It defines contents to be placed in a template. `ui:define` tag can be used to replace its contents.
- `<ui:define>` - Defines the contents to be inserted in a template.
- `<ui:include>` - Includes contents of one xhtml page into another xhtml page.
- `<ui:composition>` - Loads a template using **template** attribute. It can also define a group of components to be inserted in xhtml page.

Some of the other tags used in Facelet tags are as follows:

- `ui:decorate` – This tag is like ui:composition except that text not within the `ui:decorate` tags is included in the Facelets page.
- `ui:component` - This tag is like `ui:composition` except adds a new `UIComponent` as the root component in the `UIComponents` structure.
- `ui:param` - Specifies a variable when a facelets page is included within a template or a facelets page.

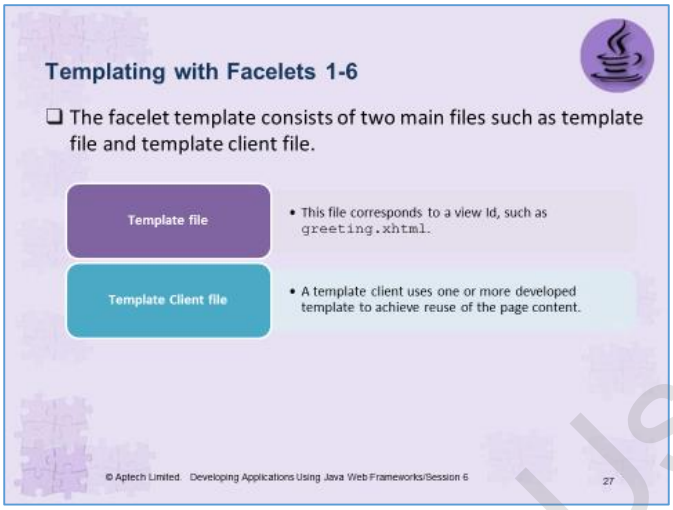
**In-Class Question:**
After you finish explaining facelet tags, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.

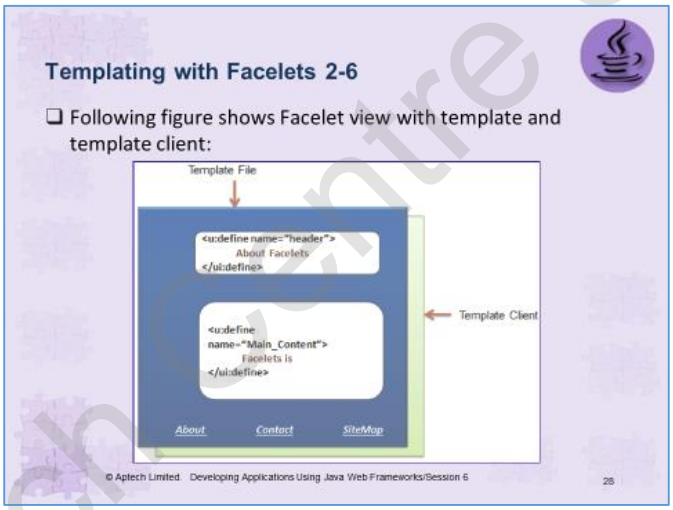
Which tags helps to create common layout for a Web application?

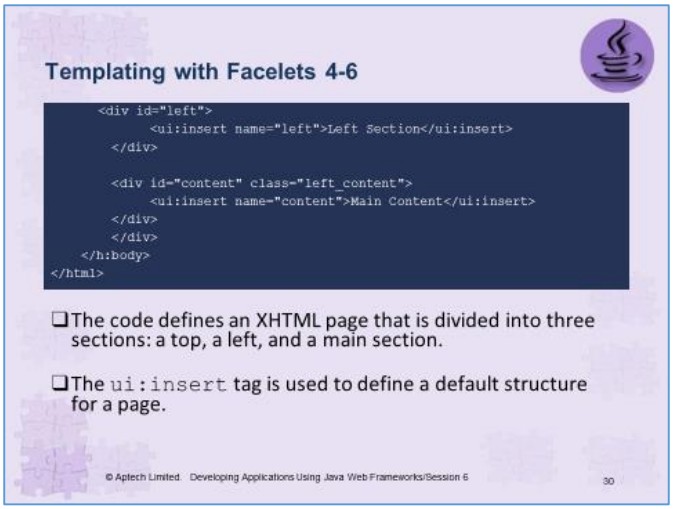
**Answer:**
Facelet tags

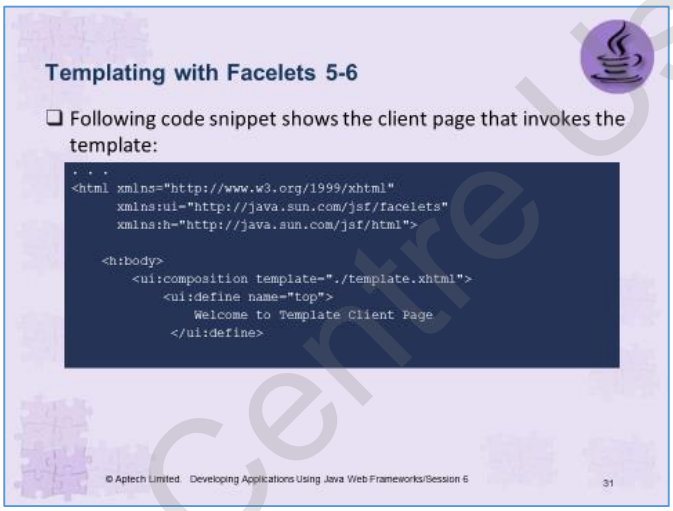
**Slides 27 to 32**

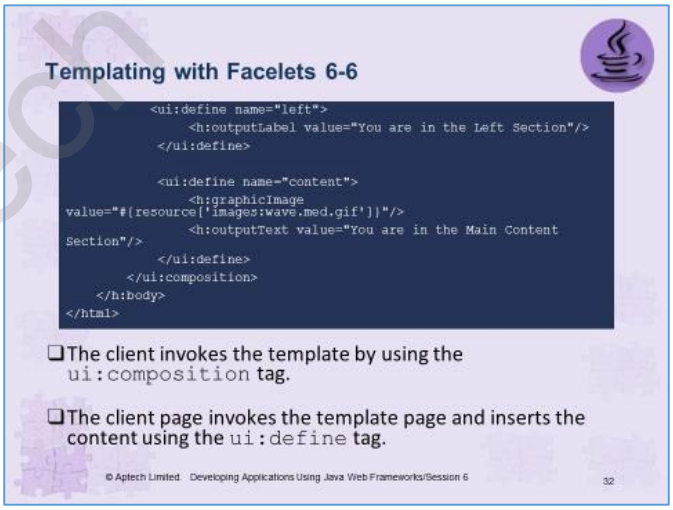
Let us understand templating with facelets.




Use slides 27 to 32 to explain templating with facelets.

Tell the students that facelets pages are XHTML pages by default.

Tell them that there are two main perspectives in templating with Facelets.

a. Template file

b. Template client file

The template client is the file whose name actually corresponds to the views such as greeting.xhtml. The template client employs one or more template files to achieve reuse of page content. Template file specifies the layout in facelets and consists of `<ui:insert/>` tags to define the structure of a facelets composition that uses the template for defining `UIComponents`.

Explain the concept of templating with the help of figure as shown on slide 28.

Then, explain the code snippet given on slides 29 and 30 to create the template file named `template.xhtml`. This page is created with this tag allows to define a default structure for a page. A template page is used as a template for other pages, usually referred to as client pages.

```
<h:head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-
8" />
        <h:outputStylesheet library="css" name="default.css"/>
        <h:outputStylesheet library="css" name="cssLayout.css"/>
        <title>Facelete Template</title>
    </h:head>

 <h:body>
 <div id=”top” class=”top”>
 <ui:insert name=”top”>Top Section</ui:insert>
 </div>

 <div>

 <div id=”left”>
 <ui:insert name=”left”>Left Section</ui:insert>
 </div>

<div id=”content” class=”left_content”>
 <ui:insert name=”content”>Main Content</ui:insert>
 </div>

</div>
</h:body>
```

The corresponding `CSSLayout.css` file is as follows:

```
#top {
    position: relative;
    background-color: #036fab;
    color: white;
    padding: 5px;
    margin: 0px 0px 10px 0px;
}
```

```
#bottom {
    position: relative;
    background-color: #c2dfef;
    padding: 5px;
    margin: 10px 0px 0px 0px;
}

#left {
    float: left;
    background-color: #ece3a5;
    padding: 5px;
    width: 150px;
}

#right {
    float: right;
    background-color: #ece3a5;
    padding: 5px;
    width: 150px;
}

.center_content {
    position: relative;
    background-color: #dddddd;
    padding: 5px;
}

.left_content {
    background-color: #dddddd;
    padding: 5px;
    margin-left: 170px;
}

.right_content {
    background-color: #dddddd;
    padding: 5px;
    margin: 0px 170px 0px 170px;
}

#top a:link, #top a:visited {
  color: white;
  font-weight : bold;
  text-decoration: none;
}

#top a:link:hover, #top a:visited:hover  {
  color: black;
  font-weight : bold;
  text-decoration : underline;
}
```

The structure of default.css is as follows:

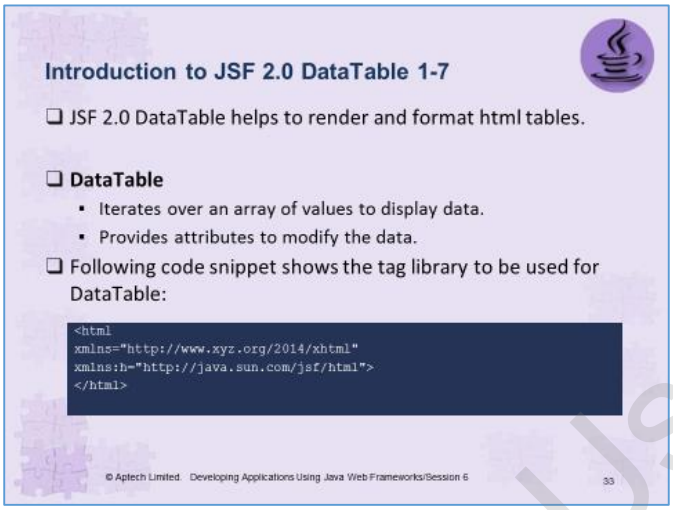
```
body {
    background-color: #ffffff;
    font-size: 12px;
    font-family: Verdana, "Verdana CE",  Arial, "Arial CE", "Lucida Grande CE",
lucida, "Helvetica CE", sans-serif;
    color: #000000;
    margin: 10px;
```

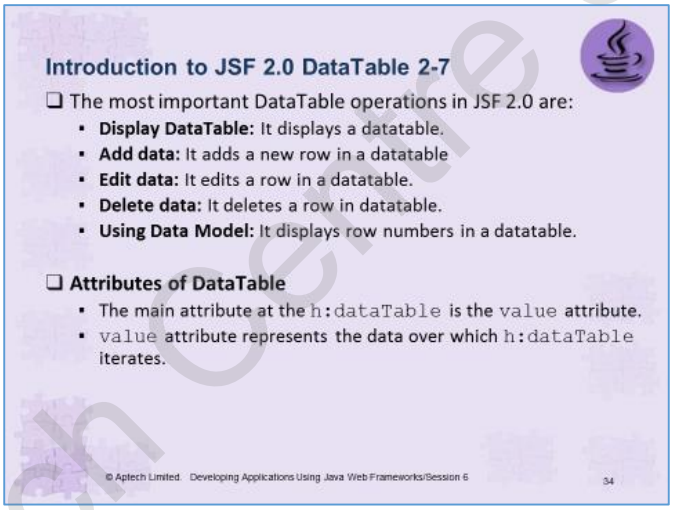
```
}

h1 {
font-family: Arial, "Arial CE", "Lucida Grande CE", lucida, "Helvetica CE",
sans-serif;
    border-bottom: 1px solid #AFAFAF;
    font-size:  16px;
    font-weight: bold;
    margin: 0px;
    padding: 0px;
    color: #D20005;
}

a:link, a:visited {
  color: #045491;
  font-weight : bold;
  text-decoration: none;
}

a:link:hover, a:visited:hover  {
  color: #045491;
  font-weight : bold;
  text-decoration : underline;
}
```

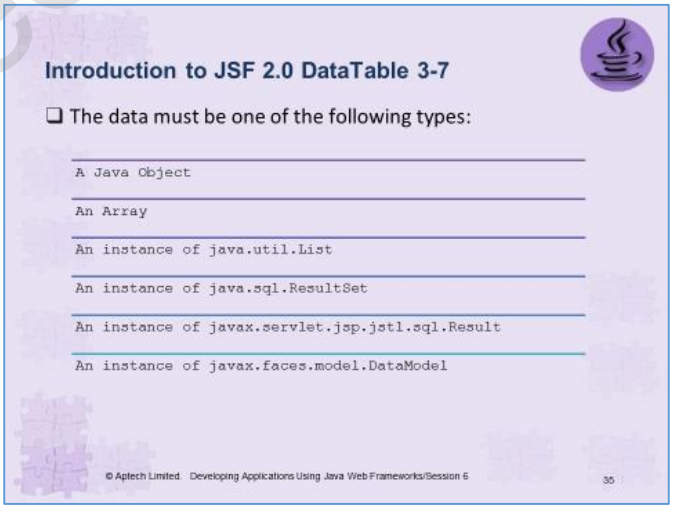
Then, explain the code of the template client as given on slides 31 and 32. Tell them the client page invokes the template by using the `ui:composition` tag. In the code, a client page named `greeting.xhtml` invokes the template page named `template.xhtml`.

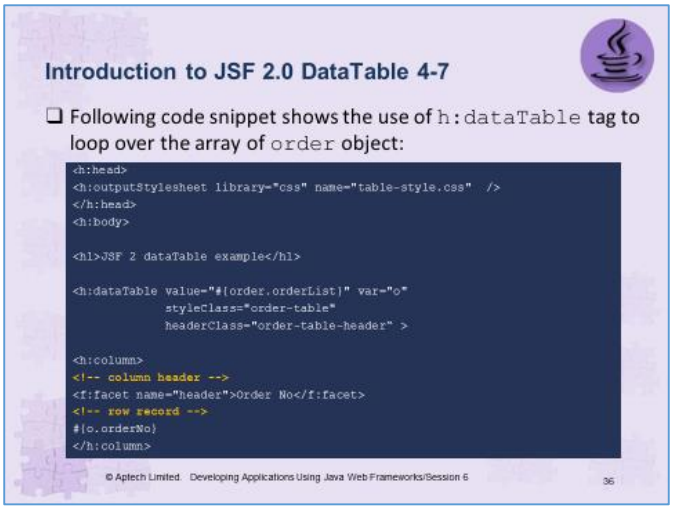
**Slides 33 to 39**

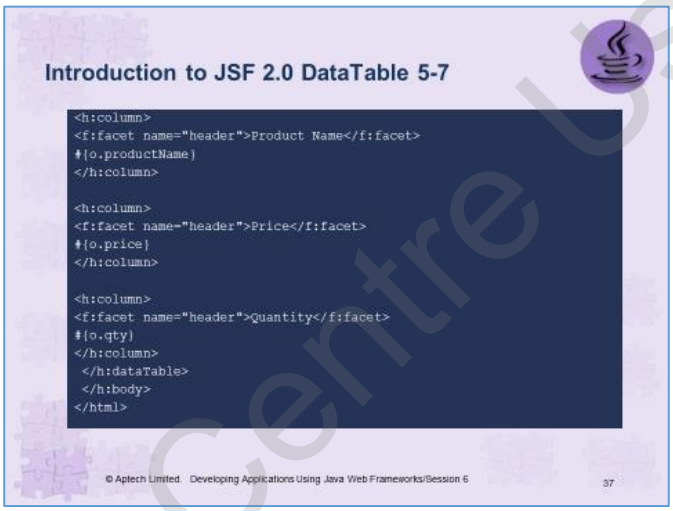
Let us understand JSF 2.0 DataTable.

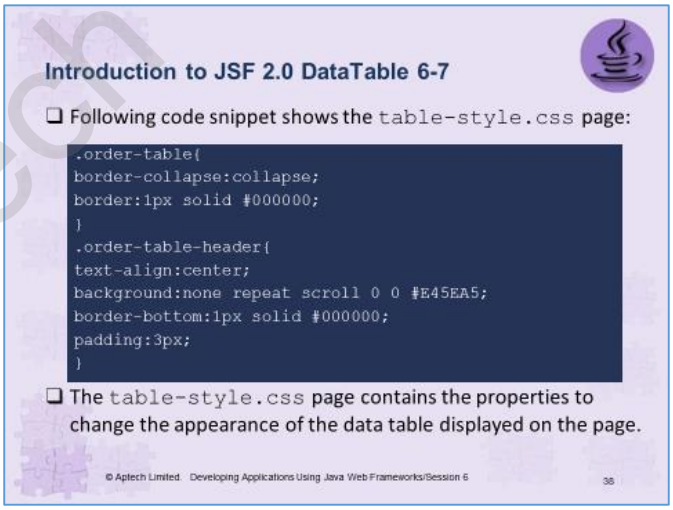


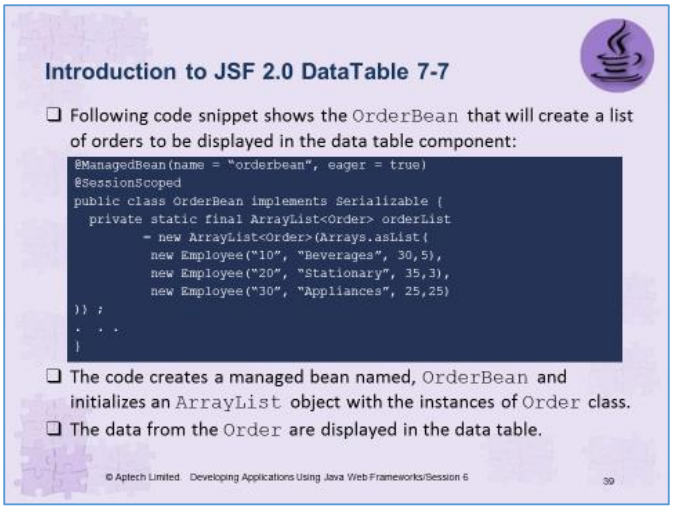








Use slides 33 to 39 to explain JSF 2.0 DataTable. JSF 2.0 DataTable helps to render and format html tables. DataTable iterates over an array of values to display data. It provides attributes to modify the data.

Then, using slides 33 to 35, explain them that JSF 2.0 support the use of h:dataTable by iterating a list of data filled inside a bean's list to create an HTML table. The main attribute at the h:dataTable is the value attribute that represents the data over which h:dataTable iterates, this data must be one of the following types:

- ‡ A Java Object
- ‡ An Array
- ‡ An instance of `java.util.List`
- ‡ An instance of `java.sql.ResultSet`
- ‡ An instance of `javax.Servlet.jsp.jstl.sql.Result`
- ‡ An instance of `javax.faces.model.DataModel`

The `h:dataTable` component will have only h:column and it discards all other component, though the h:column can render an unlimited number of the components. The `h:dataTable` component adds a header and footer for the dataTable that is created and thus, provides the developer capability.

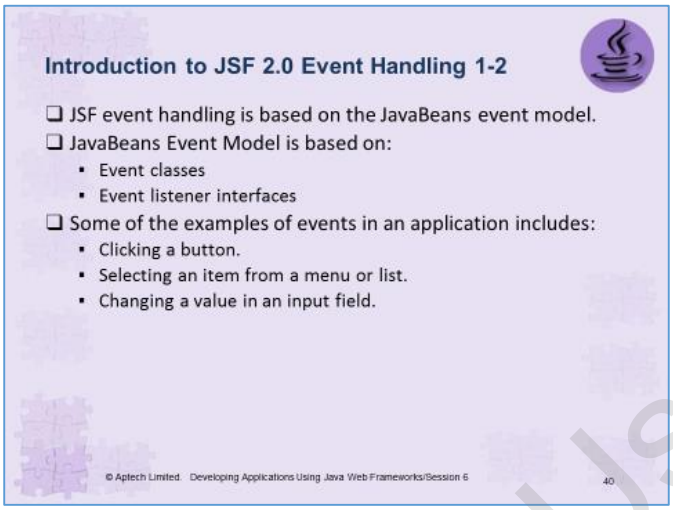
Then, explain the code snippet given on slides 36 to 39 to create a DataTable in JSF.

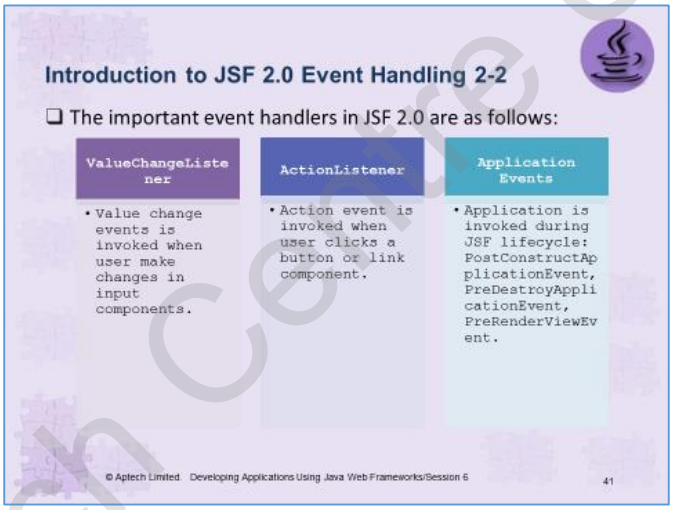
**Tips:**

You can refer to this link for understanding on DataTable:
http://www.mkyong.com/jsf2/jsf-2-datatable-example/

### Slides 40 and 41

Let us understand JSF 2.0 Event Handling.




Use slides 40 and 41 to explain JSF 2.0 Event Handling.

Tell the students that JSF supports event handling throughout the JSF life-cycle.

JSF event handling is based on the JavaBeans event model, where event classes and event listener interfaces are used by the JSF application to handle events generated by components. Some examples of events in an application include clicking a button, selecting an item from a menu or list, and changing a value in an input field. When a user activity occurs, the component creates an event object that stores information about the event and identifies the component that generated the event. The event is added to an event queue.

JSF tells the component to broadcast the event to the corresponding registered listener, which invokes the listener method that processes the event.

Then, discuss the important event handlers in JSF which are as follows:

**ValueChangeListener**

Value-change event - occurs when the user changes the value of a component such as `UIInput`, `UISelectOne`, `UISelectMany`, and `UISelectBoolean` components, that implements `EditableValueHolder`.

The `valueChangeListener` tag registers a `ValueChangeListener` instance on the `UIComponent` associated with the closest parent UIComponent custom action.

For example, the following shows a custom `ValueChangeListener` instance that is associated with an `inputText`.

```
<h:inputText id="inputText1">
 <f:valueChangeListener type="jsf.MyValChangeListener"/>
</h:inputText>
```

Some of the constraints are as follows:

- Must be nested inside a `UIComponent`.
- The corresponding `UIComponent` implementation class must implement `EditableValueHolder` and therefore, define a public `addValueChangeListener( )` method that accepts a `ValueChangeListener` parameter.
- The specified listener class must implement `javax.faces.event.ValueChangeListener`.
- Type and/or binding must be specified.

**Tips:**

To know more about `ValueChangeListener`, refer to this link:

http://www.javatutorials.co.in/jsf-2-2-valuechangelistener-attribute-example/

**ActionListener**

Action event - occurs when the user activates a component like button or hyperlink, that implements `ActionSource`.

In button or link component, add a `f:actionListener` tag inside, and specify an implementation class of `ActionListener` interface, and override its `processAction()`.

For example, to register a button event, the JSF page will invoke the ActionListener on Submit button as shown in the following code snippet:

```
<h:body>


    <h1>JSF 2 actionListener example</h1>
```

```
        <h:form id="form">

          <h:commandButton id="submitButton"
                value="Submit" action="#{normal.outcome}" >
                <f:actionListener type="com.listener.NormalActionListener" />
          </h:commandButton>

        </h:form>

    </h:body>
```

The managed bean is as follows:

```
@ManagedBean(name="normal")
@SessionScoped
public class NormalBean{
      public String outcome(){
            return "result";
        }
}
```

The implementation of the ActionListener is as follows:

```
public class NormalActionListener implements ActionListener{

        @Override
        public void processAction(ActionEvent event)
```

```
                throws AbortProcessingException {


                System.out.println("Any use case here?");


        }
}
```

Tell them that JSF also provides application events that are used to perform application specific tasks during JSF application life cycle.

The following table shows the application events:

| Event | Description |
|---|---|
| `PostConstructApplicationEvent` | Fires when application starts. It can be used to perform initialization tasks after application has started. |
| `PreDestroyApplicationEvent` | Fires when application is about to shut down. It can be used to perform a cleanup tasks before application is about to be shut down. |
| `PreRenderViewEvent` | Fires before a JSF page is to be displayed. It can be used to authenticate user and provide restricted access to JSF View. |

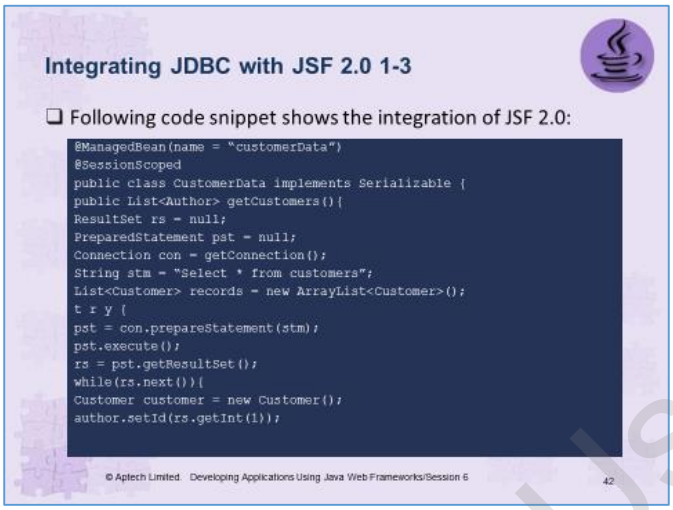
**Tips:**

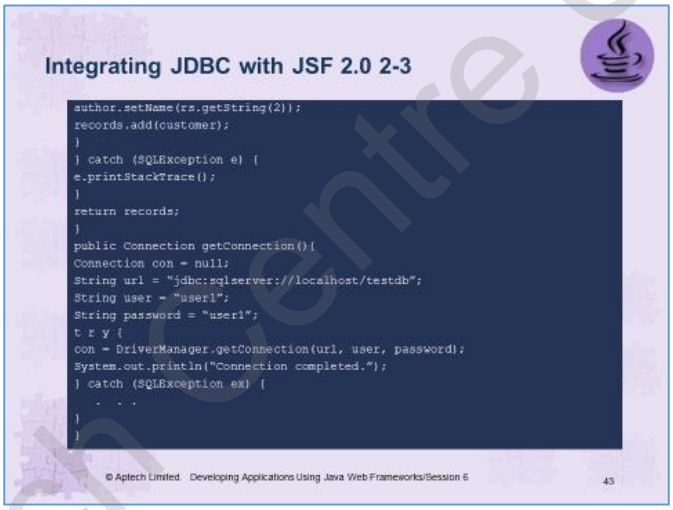
To implement application life cycle events, you can refer to this link:

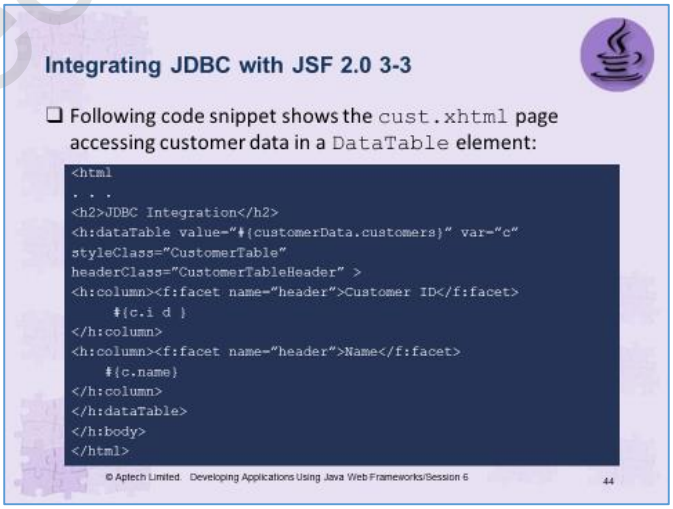
http://www.mkyong.com/jsf2/jsf-2-postconstructapplicationevent-and-predestroyapplicationevent-example/

**Slides 42 to 44**

Let us understand the integration of JDBC with JSF 2.0.



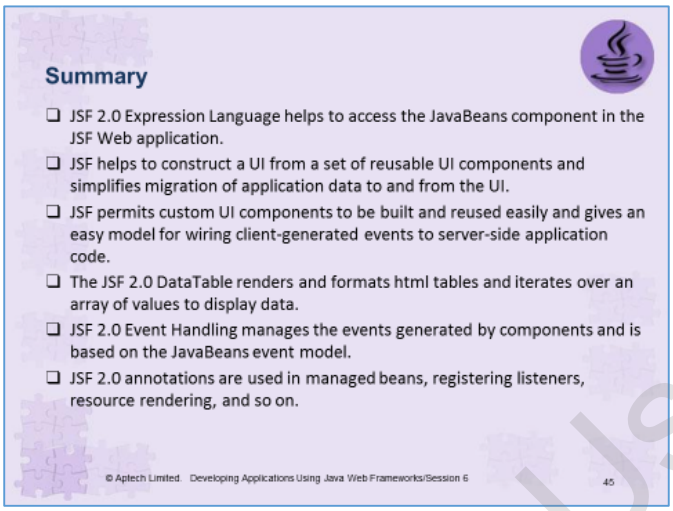




Use slides 42 to 44 to explain the integration of JDBC with JSF 2.0.

**Slide 45**

Let us summarize the session.




In slide 45, you will summarize the session. You will end the session, with a brief summary of what has been taught in the session.

## 6.3 Post Class Activities for Faculty

This session ends the **Developing Applications Using Java Web Frameworks** course. Ask the students some questions related from all the topics which will help you to know the learnings taken by the students. You can solve the queries related to other sessions taught in the course.

**Tips:**

You can also check the Articles/Blogs/Expert Videos uploaded on the OnlineVarsity site to gain additional information related to the topics covered in the course. You can also connect to online tutors on the OnlineVarsity site to ask queries related to the sessions.