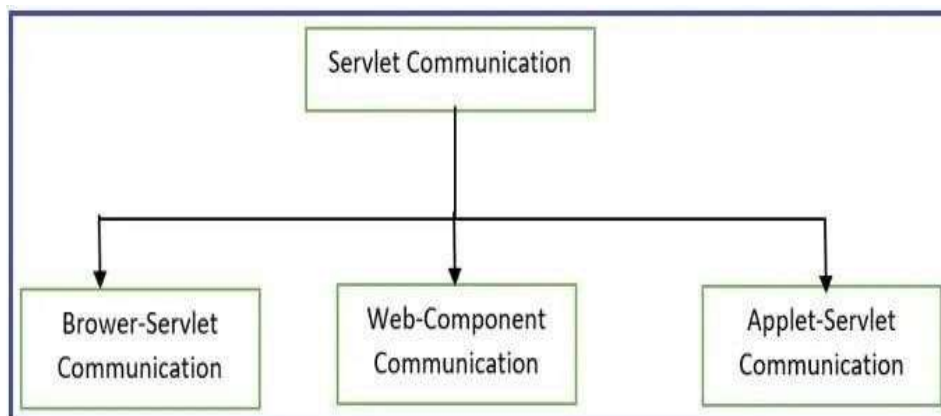


2	It can be used to pass the deployment time parameters to the servlet using during the servlet initialization like database name, file name, etc	It can be used to access the Web application parameters configured in the deployment descriptor file (WEB.xml)
3	It can be used to access the ServletContext object.	It can be used for the inter application communication between servlets, JSPs and other components of a web application.
4	Can access the initialization parameters for a servlet instance.	Can be used to access the server information about the container and the version of the API it supports.

Servlet Communication

In general, web application deployment is not at all suggestible to provide the complete application logic within a single web resource, it is suggestible to distribute the complete application logic over multiple web resources.

In the above context, to execute the application we must require communication between all the web resources, for this, we have to use Servlet Communication. In the web application, we are able to provide servlet communication in the following three ways:



Browser-Servlet Communication

In web applications, we will use a browser as a client at the client machine, from the browser we will send a request to a servlet available at the server, where the servlet will be executed and generate some response to the client browser.

In the above process, we have provided communication between the client browser and servlet. So that sending a normal request from the client to the server and getting a normal response from the server to the client is an example of Browser-Servlet Communication.

3.5 WEB COMPONENT COMMUNICATION

The process of providing communication between more than one web component available at the server machine is called Web Component Communication.

In general, web-component communication is available in between ServletServlet, Servlet-Jsp, Servlet-HTML, Jsp-Jsp, Jsp-Servlet, Jsp-HTML, and so on. In web applications, we are able to achieve web-component communication in the following 2 ways: i. Include Mechanism
ii. Forward Mechanism

Applet Servlet Communication

HTML exhibits high performance by taking less time to load in the browser. However, when we use HTML page for important user details, by default, all the parameters that are passed appended in the URL. This compromises with the security. On the other hand, applet takes more time to load but there is no problem with Java security. This is an advantage of this technique.

Applet is a compiled Java class that can be sent over the network. Applets are an alternative to HTML form page for developing websites. HTML form gives good performance, takes less time to load but has poor security. Whereas, Applets give poor performance, take time to load but have good security.

Session simply means a particular interval of time.

Session Tracking is a way to maintain state (data) of an user. It is also known as session management in servlet.

Http protocol is a stateless so we need to maintain state using session tracking techniques. Each time user requests to the server, server treats the request as the new request. So we need to maintain the state of an user to recognize to particular user.

HTTP is stateless that means each request is considered as the new request. It is shown in the figure given below:



Why use Session Tracking?

To recognize the user, it is used to recognize the particular user.

Your session is set to 20 minutes in web.xml.

Session Tracking employs Four Different techniques

A. Cookies

B. Hidden Form Field

C. URL Rewriting

D. HttpSession

A. Cookies : Cookies are little pieces of data delivered by the web server in the response header and kept by the browser. Each web client can be assigned a unique session ID by a web server. Cookies are used to keep the session going. Cookies can be turned off by the client.

B. Hidden Form Field : The information is inserted into the web pages via the hidden form field, which is then transferred to the server. These fields are hidden from the user's view.

Illustration:

```
< input type= 'hidden' name='session' value='12345'>
```

C. URL Rewriting : With each request and return, append some more data via URL as request parameters. URL rewriting is a better technique to keep session management and browser operations in sync.

D. HttpSession : A user session is represented by the HttpSession object. A session is established between an HTTP client and an HTTP

server using the HttpSession interface. A user session is a collection of data about a user that spans many HTTP requests.

3.6 INTRODUCTION TO JSP

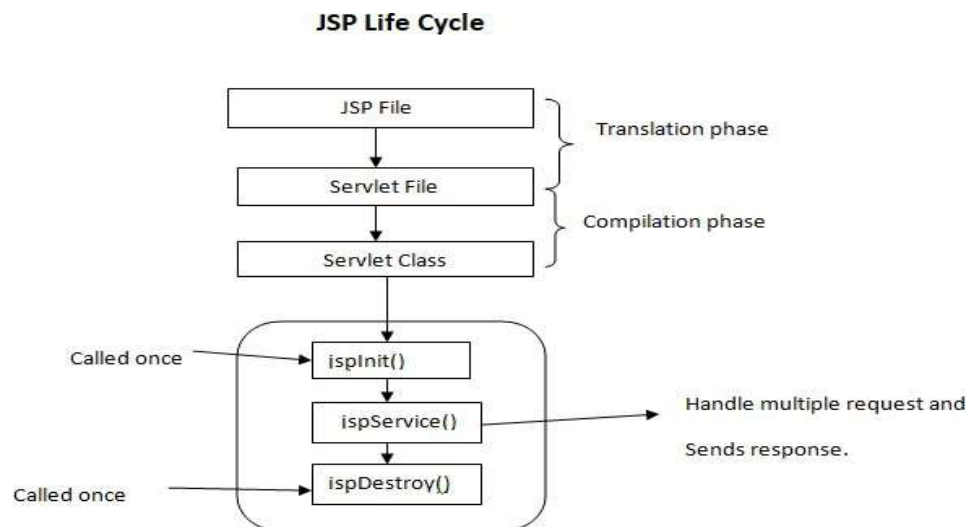
JSP (JavaServer Pages) is a technology that enables developers to create dynamic, data-driven web pages using Java. JSP is a server-side technology that generates HTML, XML, or other types of content dynamically based on data that is retrieved from a database or other sources.

JSP pages are a combination of HTML or XML markup and Java code that is embedded in special tags. The Java code is executed on the server-side and generates dynamic content that is sent to the client browser as HTML or other types of content.

JSP pages can be used to implement a wide range of web applications, from simple forms and pages to complex enterprise applications that require integration with databases, web services, and other technologies.

Life cycle of JSP

A Java Server Page life cycle is defined as the process that started with its creation which later translated to a servlet and afterward servlet lifecycle comes into play. This is how the process goes on until its destruction.



Translation of JSP page to Servlet

Compilation of JSP page(Compilation of JSP into test.java)

Classloading (test.java to test.class)

Instantiation(Object of the generated Servlet is created)

Initialization(jspInit() method is invoked by the container)
Request processing(_jspService() is invoked by the container) JSP
Cleanup (jspDestroy() method is invoked by the container)

Translation of JSP page to Servlet :

This is the first step of the JSP life cycle. This translation phase deals with the Syntactic correctness of JSP. Here test.jsp file is translated to test.java.

Compilation of JSP page :

Here the generated java servlet file (test.java) is compiled to a class file (test.class).

Classloading :

Servlet class which has been loaded from the JSP source is now loaded into the container.

Instantiation :

Here an instance of the class is generated. The container manages one or more instances by providing responses to requests.

Initialization :

jspInit() method is called only once during the life cycle immediately after the generation of Servlet instance from JSP.

Request processing :

_jspService() method is used to serve the raised requests by JSP. It takes request and response objects as parameters. This method cannot be overridden.

JSP Cleanup :

In order to remove the JSP from the use by the container or to destroy the method for servlets jspDestroy() method is used. This method is called once, if you need to perform any cleanup task like closing open files, releasing database connections jspDestroy() can be overridden.

Jsp implicit object scope

Implicit object request has the 'request' scope. 'session' scope means, the JSP object is accessible from pages that belong to the same session from where it was created. The JSP object that is created using the session scope is bound to the session object. Implicit object session has the 'session' scope.

JSP directives

In JSP (JavaServer Pages), directives are special instructions that provide information to the JSP container or web server about how to process the JSP file during translation and execution. Directives are not executed as

part of the response to the client's request, but they are used to control the translation and execution of the JSP page.

There are three types of JSP directives:

Page Directive: The page directive is used to define the attributes of the JSP page, such as error handling, scripting language, session management, and buffer size. It is declared at the beginning of the JSP file using the `<%@ %>` syntax.

Include Directive: The include directive is used to include a file in the current JSP page. It is declared using the `<%@ include %>` syntax.

Taglib Directive: The taglib directive is used to specify the location of custom tag libraries that are used in the JSP page. It is declared using the `<%@ taglib %>` syntax.

Directives provide a powerful mechanism to control the behavior of the JSP container and the JSP page during translation and execution. They enable JSP developers to specify various settings and configurations that can affect the performance and behavior of the JSP application.

JSP Scripting elements

JSP (JavaServer Pages) scripting elements are used to include Java code in a JSP page. There are three types of JSP scripting elements:

Expression: The expression element is used to include a Java expression in the JSP page that is evaluated at runtime and the result is included in the response. It is declared using the `<%= %>` syntax. For example: `<p>The current time is <%= new java.util.Date() %></p>`

In this example, the expression `<%= new java.util.Date() %>` returns the current time and is included in the response.

Declaration: The declaration element is used to declare variables and methods that can be used in the JSP page. It is declared using the `<%! %>` syntax. For example:

```
<%! int count = 0; %>
```

In this example, a variable named count is declared and initialized to 0.

Scriptlet: The scriptlet element is used to include Java code in the JSP page that is executed at runtime. It is declared using the `<% %>` syntax. For example:

```
<% if (count > 0) { %> <p>The count is <%= count %></p> <% } %>
```

In this example, the if statement is used to conditionally include a paragraph tag with the value of the count variable in the response.

JSP scripting elements provide a flexible and powerful mechanism to include Java code in JSP pages, which enables developers to dynamically generate content and interact with databases and other resources. However, it is important to use scripting elements judiciously and maintain the separation of concerns between the presentation and business logic layers of the application.

JSP Action Tags

JSP (JavaServer Pages) action tags are special tags that are used to perform specific actions in a JSP page, such as forwarding to another resource, including content from another resource, setting request parameters, and defining custom tag libraries.

There are several JSP action tags available, including:

<jsp:forward>: The `<jsp:forward>` tag is used to forward the request from the current JSP page to another resource, such as another JSP page or a servlet. It is typically used when a request needs to be processed by multiple resources.

<jsp:include>: The `<jsp:include>` tag is used to include content from another resource, such as another JSP page or a static HTML file, in the current JSP page. It is typically used to reuse common content across multiple pages.

<jsp:param>: The `<jsp:param>` tag is used to set request parameters that are passed to the included or forwarded resource. It is typically used to pass data between different components of the application.

<jsp:useBean>: The `<jsp:useBean>` tag is used to instantiate and initialize a JavaBean component that can be used in the JSP page. It is typically used to encapsulate complex business logic in a separate component.

<jsp:setProperty> and <jsp:getProperty>: The `<jsp:setProperty>` and `<jsp:getProperty>` tags are used to set and get properties of a JavaBean component that is used in the JSP page. They are typically used to access and modify data stored in the JavaBean.

Custom Tag Libraries: JSP also allows you to define custom tag libraries, which are sets of custom tags that can be used in JSP pages. Custom tag libraries are typically used to encapsulate complex functionality and provide a higher-level abstraction for common tasks.

JSP action tags provide a powerful mechanism for controlling the behavior of JSP pages and interacting with other components of the application.

They enable developers to create reusable components, encapsulate business logic, and manage the flow of requests and responses between different resources.

Custom Tags in JSP

Custom tags are a powerful feature of JSP (JavaServer Pages) that enable developers to define their own tags that can be used in JSP pages. Custom tags can encapsulate complex functionality and provide a higher-level abstraction for common tasks, which can improve the clarity, maintainability, and reusability of JSP pages.

Custom tags can be defined using the JSP Standard Tag Library (JSTL) or by creating a custom tag library. To create a custom tag library, you define a set of tag handlers that implement the behavior of the custom tags. A tag handler is a Java class that extends a JSP tag handler base class and overrides its methods to implement the behavior of the custom tag. The JSP container invokes the methods of the tag handler during the processing of the JSP page.

There are two types of custom tags:

Simple Custom Tags: Simple custom tags are implemented using a single tag handler class that implements the behavior of the custom tag. Simple custom tags are used to encapsulate simple functionality, such as formatting or validation.

Composite Custom Tags: Composite custom tags are implemented using multiple tag handler classes that work together to implement the behavior of the custom tag. Composite custom tags are used to encapsulate complex functionality, such as a data grid or a chart.

To use a custom tag in a JSP page, you first declare the custom tag library using the `<%@taglib%>` directive, which specifies the URI of the custom tag library and an alias for the library. You can then use the custom tags in the JSP page using the tag name and any attributes that are defined for the tag.

Custom tags provide a powerful mechanism for creating reusable components in JSP pages, which can improve the maintainability, scalability, and performance of web applications. However, it is important to use custom tags judiciously and maintain the separation of concerns between the presentation and business logic layers of the application.

UNIT END EXERCISES

1. What are servlets? Explain the request/response paradigm of servlet
2. Explain different phases of Servlet Life Cycle.
3. Write a short note on GenericServlet Class.
4. Differentiate between ServletConfig and ServletContext.
5. Differentiate between JSP and Servlet
6. Differentiate between JSP Include directives and JSP Include action.
7. What is page directives? Explain page directives and its attributes.

4 JAVA BEANS

Unit Structure:

- 4.1. Objective
- 4.2. Introduction
- 4.3. JavaBeans Properties
- 4.4. Summary
- 4.5. Reference for further reading
- 4.6. Unit End Exercises

4.1. OBJECTIVE

- To understand the concept of java bean.
- To understand the properties of java bean with example.

4.2. INTRODUCTION

Java Bean:-

- A Java Bean is a software component that has been designed to be reusable in a variety of different environments.
- There is no restriction on the capability of a Bean. It may perform a simple function, such as checking the spelling of a document, or a complex function, such as forecasting the performance of a stock portfolio.
- A Bean may be visible to an end user. One example of this is a button on a graphical user interface. A Bean may also be invisible to a user. Software to decode a stream of multimedia information in real time.

Advantages of Java Beans:-

- Since java bean is a reusable component, it implies Java's "write-once, run-anywhere" paradigm. It is used in distributed environments.
- The properties, events, and methods of a Bean that are exposed to an application builder tool can be controlled.
- A Bean may be designed to operate correctly in different environments.

- Auxiliary software can be provided to help a person configure a Bean. This software is only needed when the design-time parameters for that component are being set. It does not need to be included in the runtime environment.
- The configuration settings of a Bean can be saved in persistent storage and restored at a later time.
- A Bean may register to receive events from other objects and can generate events that are sent to other objects.

JAR Files:-

- A JAR file is a Java archive file, which allows you to efficiently deploy a set of classes and their associated resources.
- For example, A multimedia application may use various sound and image files. A set of Beans can control how and when this information is presented. All of these pieces can be placed into one JAR file.
- The elements in a JAR file are compressed, which makes downloading a JAR file much faster than separately downloading several uncompressed files.
- Digital signatures may also be associated with the individual elements in a JAR file, which allows a consumer to be sure that these elements were produced by a specific organization.
- To generate jar file, the syntax is:- `jar options files`

Creating a JAR File

- The following command creates a JAR file named `Xyz.jar` that contains all of the `.class` and `.gif` files in the current directory:

```
jar cf Xyz.jar *.class *.gif
```

Jar Option are:- **c:-** A new archive is to be created.

C:- Change directories during command execution.

f:- The first element in the file list is the name of the archive that is to be created or accessed.

m:- The second element in the file list is the name of the external manifest file.

M:- Manifest file not created.

t:- The archive contents should be tabulated. **u:-** Update existing

JAR file. **v:-** Verbose output should be provided by the utility as it

executes. **x:-** Files are to be extracted from the archive.

0:-Do not use compression.

- If a manifest file such as **Yxz.mf** is available, it can be used with the following command:

jar cfm Xyz.jar Yxz.mf *.class *.gif

- The following command lists the contents of **Xyz.jar**:

jar tf Xyz.jar

- The following command extracts the contents of **Xyz.jar** and places those files in the current directory: **jar xf Xyz.jar**

- The following command adds the file **file1.class** to **Xyz.jar**:

jar -uf Xyz.jar file1.class

- The following command adds all files below **directoryX** to **Xyz.jar**:

jar -uf Xyz.jar -C directoryX *

Manifest Files:-

- A manifest file indicates which of the components in a JAR file are Java Beans.
- An example of a manifest file is provided in the following listing. It defines a JAR file that contains four .gif files and one .class file. The last entry is a Bean.

Name: sunw/demo/slides/slide0.gif

Name: sunw/demo/slides/slide1.gif

Name: sunw/demo/slides/slide2.gif

Name: sunw/demo/slides/slide3.gif

Name: sunw/demo/slides/Slides.class

Java-Bean: True

- A manifest file may reference several .class files. If a .class file is a Java Bean, its entry must be immediately followed by the line "**JavaBean: True**".

Introspection:-

- Introspection is the process of analyzing a Bean to determine its capabilities.
- It allows an application builder tool to present information about a component to a software designer.
- There are two ways in which the developer of a Bean can indicate which of its properties, events, and methods should be exposed by an application builder tool.
 - In the first method, simple naming conventions are used. These allow the introspection mechanisms to infer information about a Bean.
 - In the second way, an additional class is provided that explicitly supplies this information.

- The following sections indicate the design patterns for properties and events that enable the functionality of a Bean to be determined.

4.3 JAVABEANS PROPERTIES

Design Patterns for Properties:-

- Java Bean consists of properties, methods and events, which is called as design patterns.
- A property is a subset of a Bean's state. The values assigned to the properties determine the behavior and appearance of that component.
- There are three types of properties: **simple, Boolean, and indexed.**

1. Simple Properties

- A simple property has a single value. • Design pattern for simple property is: `public T getN();` `public void setN(T arg);`
- where N is the name of the property and T is its type.
- A read/write property has both of these methods to access its values.
- A read-only property has only a get method. A write-only property has only a set method.

The following listing shows a class that has two read/write simple properties:

```

public class Student
{
    private int
rollno,marks;
    public int getRollno( )
    {
        return rollno;
    }
    public void setRollno(int d)
    {
        rollno = d;
    }
    public int getMarks( )
    {
        return marks;
    }
    public void setMarks(int h)
    {
        marks = h;
    }
}

```

2. Boolean Properties:-

- A Boolean property has a value of **true** or **false**.

Design patterns are:-

```

public boolean isN( );
public boolean getN( );
public void setN(boolean
value);

```

Where N is the name of the property:

- Either the first or second pattern can be used to retrieve the value of a Boolean property.

The following listing shows a class that has one Boolean property:

```

public class Line
{
    private boolean dotted =
false;
    public boolean isDotted( )
    {
        return dotted;
    }
    public void setDotted(boolean dotted)
    {
        this.dotted = dotted;
    }
}

```

3. Indexed Properties:-

- An indexed property consists of multiple values.
- Design patterns are:-

```

public T getN(int index);    public
void setN(int index, T value);    public T[ ] getN( );    public
void setN(T values[ ]);

```
- Where N is the name of the property and T is its type.

The following listing shows a class that has one read/write indexed property:

```

public class PieChart
{
    private double data[ ];    public double getData(int index)
    {
        return data[index];
    }
    public void setData(int index, double value)
    {
        data[index] = value;
    }
    public double[ ] getData( )
    {
        return data;
    }
    public void setData(double[ ] values)
    {

```

```

        data = new double[values.length];
        System.arraycopy(values, 0, data, 0, values.length);
    }
}

```

4. Design Patterns for Events:-

- Beans use the delegation event model.
- Beans can generate events and send them to other objects.
- Design patterns are:-

```

        public void addTListener(TListener eventListener);

        public void addTListener(TListener eventListener) throws
        TooManyListeners;    public void removeTListener(TListener
        eventListener); •    Where T is the type of the event.

```

- These methods are used by event listeners to register an interest in events of a specific type.
- The first pattern indicates that a Bean can multicast an event to multiple listeners.
- The second pattern indicates that a Bean can unicast an event to only one listener.
- The third pattern is used by a listener when it no longer wishes to receive a specific type of event notification from a Bean.

The following listing outlines a class that notifies other objects when a temperature value moves outside a specific range.

```

public class Thermometer
{
    public void addTemperatureListener(TemperatureListener tl)
    {
        ...
    }
    public void removeTemperatureListener(TemperatureListener tl)
    {
        ...
    }
}

```


5. Bound Properties:-

- A Bean that has a bound property generates an event when the property is changed.
- The event is of type `PropertyChangeEvent` and is sent to objects that previously registered an interest in receiving such notifications.
- Sometimes when a Bean property changes, another object might need to be notified of the change, and react to the change. Whenever a bound property changes, notification of the change is sent to interested listeners.
- The accessor methods for a bound property are defined in the same way as those for simple properties. However, you also need to provide the event listener registration methods for `PropertyChangeListener` classes and fire a `PropertyChangeEvent` event to the `PropertyChangeListener` objects by calling their `propertyChange` methods.

- **Class:-**

- `PropertyChangeEvent` class:- The `PropertyChangeEvent` class encapsulates property change information, and is sent from the property change event source to each object in the property change listener list with the `propertyChange` method. ●

Methods:-

- `public void addPropertyChangeListener(PropertyChangeListener listener):-`
 - Add a `PropertyChangeListener` to the listener list. The listener is registered for all properties.
 - `public void firePropertyChange(String propertyName, Object oldValue, Object newValue):-` Report a bound property update to any registered listeners. No event is fired if old and new are equal and non-null.
 - `public void removePropertyChangeListener(PropertyChangeListener listener):-` Remove a

PropertyChangeListener from the listener list. This removes a PropertyChangeListener that was registered for all properties.

6. Constrained Properties:-

- The java bean component allows for the possibility that one or more listener objects might not allow certain changes to the value of a property. This is known as Constrained properties.
- A Bean that has a constrained property generates an event when an attempt is made to change its value. The event is of type **PropertyChangeEvent**. It is sent to objects that previously registered an interest in receiving such notifications. Those other objects have the ability to veto the proposed change.
- Constrained properties are more complicated than bound properties because they also support property change listeners which happen to be vetoers.
- The following operations in the setXXX method for the constrained property must be implemented in this order:
 - Save the old value in case the change is vetoed.
 - Notify listeners of the new proposed value, allowing them to veto the change.
 - If no listener vetoes the change (no exception is thrown), set the property to the new value.
- **Class:-**
 - **VetoableChangeSupport**:-This class can be used by beans that support constrained properties. You can use an instance of this class as a member field of your bean and delegate various work to it.
- **Methods:-**
 - public void **addVetoableChangeListener**(VetoableChangeListener listener):-Add a VetoableListener to the listener list. The listener is registered for all properties.
 - public void **fireVetoableChange**(String propertyName, Object oldValue, Object newValue) throws PropertyVetoException:- Report a vetoable property update to any registered listeners. If anyone vetos the change, then fire a new event reverting everyone to the

old value and then rethrow the `PropertyVetoException`. No event is fired if old and new are equal and non-null.

- `public void removeVetoableChangeListener(VetoableChangeListener listener)`:-Remove a `VetoableChangeListener` from the listener list. This removes a `VetoableChangeListener` that was registered for all properties.

7. BeanInfo Interface:-

- A bean implementor who wishes to provide explicit information about their bean may provide a `BeanInfo` class that implements this `BeanInfo` interface and provides explicit information about the methods, properties, events, etc, of their bean.
- This interface is used to determine design patterns.
- This interface defines several methods, including these:
 - **`PropertyDescriptor[] getPropertyDescriptors()`**:- returns an array of `PropertyDescriptors` describing the editable properties supported by this bean.
 - **`EventSetDescriptor[] getEventSetDescriptors()`**:- returns an array of `EventSetDescriptors` describing the kinds of events fired by this bean.
 - **`MethodDescriptor[] getMethodDescriptors()`**:- returns an array of `MethodDescriptors` describing the externally visible methods supported by this bean.
- By implementing these methods, a developer can designate exactly what is presented to a user.
- **`SimpleBeanInfo`** is a class that provides default implementations of the **`BeanInfo`** interface, including the three methods just shown.
 - You may extend this class and ○ override one or more of them.
- **Example:-**The following example shows how this is done for the **`Colors`**
- **Bean.** This bean displays a colored ellipse which changes its color randomly and shape to rectangle. **`ColorsBeanInfo`** is a subclass of **`SimpleBeanInfo`**.

- It overrides **getPropertyDescriptors()** in order to designate which properties are presented to a Bean user. This method creates a **PropertyDescriptor** object for the **rectangular** property. The **PropertyDescriptor** constructor that is used is shown here:
- **PropertyDescriptor(String property, Class beanCls)** throws **IntrospectionException**.
- Here, the first argument is the name of the property, and the second argument is the class of the Bean.

```
// A Bean information class.

package sunw.demo.colors; import
java.beans.*;

public class ColorsBeanInfo extends SimpleBeanInfo
{
    public PropertyDescriptor[] getPropertyDescriptors( )
    {
        try
        {
            PropertyDescriptor rectangular = new
            PropertyDescriptor("rectangular",
            Colors.class);
            PropertyDescriptor pd[] = {rectangular};
            return pd;
        }
        catch(Exception e)
        {
            return null;
        }
    }
}
```

- You must compile this file from the BDK\\demo directory or set CLASSPATH so that it includes c:\\bdk\\demo. If you don't, the compiler won't find the Colors.class file properly.
- After this file is successfully compiled, the colors.mft file can be updated, as shown here:

Name: sunw/demo/colors/ColorsBeanInfo.class

Name: sunw/demo/colors/Colors.class

Java-Bean: True

8. Persistence:-

- Persistence is the ability to save a Bean to nonvolatile storage and retrieve it at a later time.
- In the first case, we can manually save the bean by selecting File/Save menu from the menu bar of the bean box. This allows you to specify the name of a file to which the beans and their configuration parameters should be saved. To restore, go to the menu bar and select File/Load.
- **Example:-**
 - Let us first see how the BDK allows you to save a set of Beans that have been configured and connected together to form an application. Suppose we have two beans colors and TickTock. The rectangular property of the Colors Bean was changed to true, and the interval property of the TickTock Bean was changed to one second.
 - To save the application, go to the menu bar of the BeanBox and select File Save. A dialog box should appear, allowing you to specify the name of a file to which the Beans and their configuration parameters should be saved. Supply a filename and click the OK button on that dialog box. Exit from the BDK.
 - Start the BDK again. To restore the application, go to the menu bar of the BeanBox and select File | Load. A dialog box should appear, allowing you to specify the name of the file from which an application should be restored. Supply the name of the file in which the application was saved, and click the OK button.
 - Your application should now be functioning.
- In the second case, object serialization is used. To do this, the bean class must implement a `java.io.Serializable` interface.
- The object serialization capabilities provided by the Java class libraries are used to provide persistence for Beans.
- If a Bean inherits directly or indirectly from **java.awt.Component**, it is automatically serializable, because that class implements the **java.io.Serializable** interface. If a Bean does not inherit an implementation of the **Serializable** interface, you must provide this yourself. Otherwise, containers cannot save the configuration of your component.
- The **transient** keyword can be used to designate data members of a Bean that should not be serialized.

9. The Java Beans API:-

- The Java Beans functionality is provided by a set of classes and interfaces in the **java.beans** package.

The Interfaces Defined in java.beans:

- **AppletInitializer:-** Methods in this interface are used to initialize Beans that are also applets.
- **BeanInfo:-** This interface allows a designer to specify information about the properties, events, and methods of a Bean.
- **Customizer:-** This interface allows a designer to provide a graphical user interface through which a Bean may be configured.
- **DesignMode:-** Methods in this interface determine if a Bean is executing in design mode.
- **PropertyChangeListener:-** A method in this interface is invoked when a bound property is changed.
- **PropertyEditor:-** Objects that implement this interface allow designers to change and display property values.
- **VetoableChangeListener:-** A method in this interface is invoked when a constrained property is changed.
- **Visibility:-** Methods in this interface allow a Bean to execute in environments where a graphical user interface is not available.

10. The Classes Defined in java.beans:-

- **BeanDescriptor:-** This class provides information about a Bean. It also allows you to associate a customizer with a Bean.
- **Beans:-** This class is used to obtain information about a Bean.
- **EventSetDescriptor:-** Instances of this class describe an event that can be generated by a Bean.
- **IndexedPropertyDescriptor:-** Instances of this class describe an indexed property of a Bean.
- **FeatureDescriptor:-** This is the superclass of the `PropertyDescriptor`, `EventSetDescriptor`, and `MethodDescriptor` classes.

- **IntrospectionException**:-An exception of this type is generated if a problem occurs when analyzing a Bean.
- **Introspector**:- This class analyzes a Bean and constructs a BeanInfo object that describes the component.
- **MethodDescriptor**:-Instances of this class describe a method of a Bean.
- **ParameterDescriptor**:-Instances of this class describe a method parameter.
- **PropertyChangeEvent**:-This event is generated when bound or constrained properties are changed. It is sent to objects that registered an interest in these events and implement either the PropertyChangeListener or VetoableChangeListener interfaces.
- Example:- The following program illustrates the **Introspector**, **PropertyDescriptor**, and **EventSetDescriptor** classes and the **BeanInfo** interface. It Lists the properties and events of the **Colors** Bean.

```

//Show properties and
events. package
sunw.demo.colors; import
java.awt.*; import
java.beans.*;

    public class IntrospectorDemo
    {
        public static void main(String args[ ])
        {
            try {
Class c = Class.forName("sunw.demo.colors.Colors");
BeanInfo beanInfo = Introspector.getBeanInfo(c);
PropertyDescriptor beanDescriptor =beanInfo.getBeanDescriptor( );
System.out.println("Bean name = " +beanDescriptor.getName( ));
System.out.println("Properties:");
PropertyDescriptor          propertyDescriptor[          ]=
            beanInfo.getPropertyDescriptors();

            for(int i = 0; i < propertyDescriptor.length; i++)
            {
                System.out.println("\t" +propertyDescriptor[i].getName( ));
            }

```



```

        System.out.println("Events:");

        EventSetDescriptor eventSetDescriptor[] =
        beanInfo.getEventSetDescriptors();

        for(int i = 0; i < eventSetDescriptor.length; i++)
        {
            System.out.println("\t" +eventSetDescriptor[i].getName());
        }

        }

        catch(Exception e) {
            System.out.println("Exception caught. " + e);
        }
    }
}

```

The output from this program is the following:

Bean name =

Colors Properties:

rectangular Events:

propertyChange mouseMotion focus , mouse,
inputMethod, key, component.

4.4. SUMMARY

- JavaBeans is a portable, platform-independent model written in Java Programming Language. Its components are referred to as beans.
- A JavaBean property can be accessed by the user of the object.

- JavaBeans components are built purely in Java, hence are fully portable to any platform
- JavaBeans is pretty compatible, there isn't any new complicated mechanism for registering components with the run-time system.

4.5. REFERENCE FOR FURTHER READING

1. Herbert Schildt, Java2: The Complete Reference, Tata McGrawHill, 5th Edition
2. Joe Wigglesworth and Paula McMillan, Java Programming: Advanced Topics, Thomson Course Technology (SPD), 3rd Edition

4.6. UNIT END EXERCISES

1. What are Java Beans? What are the advantages of using Java Beans?
2. What are JAR files? What are the options available with the JAR files? List any five options.
3. What is meant by the Design Pattern of the Beans? What are the property types that support design patterns?
4. Explain BeanInfo Interface briefly. And what are the different Descriptors that contain in BeanInfo classes
5. Define manifest file of Beans & explain the concept of Introspection of the Bean.

5 STRUTS 2 PART-I

Unit Structure:

- 5.1 Objective
- 5.2 Introduction
- 5.3 Basic MVC Architecture,
- 5.4 Struts 2 framework features,

- 5.5 Struts 2 MVC pattern,
- 5.6 Request life cycle, Examples,
- 5.7 Configuration Files
- 5.8 Actions
- 5.9 Summary

5.10 Reference for further reading

5.11 Unit End Exercises

5.1 OBJECTIVE

1. Building applications on the web using struts 2.
2. Using web frameworks.
3. To understand and explore The Struts 2 framework.

5.2 INTRODUCTION

- Struts is an application development framework that is designed for and used with the popular J2EE platform.
- It cuts time out of the development process and makes developers more productive by providing them a series of tools and components to build applications with.
- Struts falls under the Jakarta subproject of the Apache Software Foundation and comes with an Open Source license.
- The struts framework is an open source framework for creating wellstructured web based applications.
- The struts framework is based on the **Model View Controller (MVC)** paradigm which distinctly separates all the three layers - Model (state of the application), View (presentation) and Controller (controlling the application flow).
- This makes struts different from conventional JSP applications where sometimes logic, flow and UI are mingled in a Java Server Page.

- The struts framework is a complete **web framework** as it provides complete web form components, validators, error handling, internationalization, tiles and more. Struts framework provides its own Controller component.
- Struts can integrate well with Java Server Pages (JSP), Java Server Faces (JSF), JSTL, Velocity templates and many other presentation technologies for View.
- For Model, Struts works great with data access technologies like JDBC, Hibernate, EJB.
- Struts: a collection of Java code designed to help you build solid applications while saving time. It provides the basic skeleton and plumbing;
- Second, the benefit of using a framework is that it allows your code to be highly platform independent. For example, the same Struts code should work under Tomcat on an old Windows machine as runs using Weblogic on Linux or Solaris in production.

5.3 BASIC MVC ARCHITECTURE,

- Struts is based on the time-proven Model-View-Controller (MVC) design pattern.
- The MVC pattern is widely recognized as being among the most welldeveloped and mature design patterns in use.
- By using the MVC design pattern, processing is broken into three distinct sections aptly named the Model, the View, and the Controller.

These are described in the following subsections:

A) Model Components

- Model components provide a "model" of the business logic or data behind a Struts program. For example, in a Struts application that manages customer information, it may be appropriate to have a "Customer" Model component that provides program access to information about customers.
- It's very common for Model components to provide interfaces to databases or back-end systems. For example, if a Struts application needs to access employee information that is kept in an enterprise HR information system, it might be appropriate to design an "Employee" Model component that acts as an interface between the Struts application and the HR information system.
- Model components are generally standard Java classes. There is no specifically required format for a Model component, so it may be possible to reuse Java code written for other projects.

B) View Components

- View components are those pieces of an application that present information to users and accept input. In Struts applications, these correspond to Web pages.
- View components are used to display the information provided by Model components. For example, the "Customer" Model component needs a View component to display its information. Usually, there will be one or more View components for each Web page in a Struts application.
- View components are generally built using JavaServer Page files. Struts provides a large number of "**JSP Custom Tags**" which extend the normal capabilities of JSP and simplify the development of View components.

C) Controller Components

- Controller components coordinate activities in the application.
- This may mean taking data from the user and updating a database through a Model component, or it may mean detecting an error condition with a back-end system and directing the user through special error processing.
- Controller components accept data from the users, decide which Model components need to be updated, and then decide which View component needs to be called to display the results.
- One of the major contributions of Controller components is that they allow the developer to remove much of the error handling logic from the JSP pages in their application.
- This can significantly simplify the logic in the pages and make them easier to develop and maintain.
- Controller components in Struts are Java classes and must be built using specific rules. They are usually referred to as "Action classes."

Architecture Overview

- All incoming requests are intercepted by the Struts servlet controller.
- The Struts Configuration file `struts-config.xml` is used by the controller to determine the routing of the flow.
- This flows consists of an alternation between two transitions:

From View to Action	A user clicks on a link or submits a form on an HTML or JSP page. The controller receives the request, looks up the mapping for this request, and forwards it to an action. The action in turn calls a Model layer (Business layer) service or function.
From Action to View	After the call to an underlying function or service returns to the action class, the action forwards to a resource in the View layer and a page is displayed in a web browser.

5.4 STRUTS 2 FRAMEWORK FEATURES

1. Simple and Easy
2. Simplified Design
3. Tag support
4. Easy to modify tags
5. Easy Plugins
6. Stateful Checkboxes
7. Configurable MVC components
8. POJO based actions
9. AJAX support
10. Integration support
11. Various Result Types
12. Various Tag support
13. Theme and Template support

Simplified Design

- Programming abstract classes instead of interfaces is one of the design problems of struts1 framework that has been resolved in the struts 2 framework.
- Most of the Struts 2 classes are based on interfaces and most of its core interfaces are HTTP independent.
- Struts 2 Action classes are framework independent and are simplified to look as simple POJOs.

- Framework components are tried to keep loosely coupled.

Tag support

- Struts2 has improved the form tags and the new tags allow the developers to write less code.

Easy to modify tags

- Tag markups in Struts2 can be tweaked using Freemarker templates. This does not require JSP or java knowledge. Basic HTML, XML and CSS knowledge is enough to modify the tags.

Easy Plugins

- Struts 2 extensions can be added by dropping in a JAR. No manual configuration is required

AJAX Support

- The AJAX theme gives interactive applications a significant boost. The framework provides a set of tags to help you ajaxify your applications, even on Dojo.

Stateful Checkboxes

- Struts 2 checkboxes do not require special handling for false values.

QuickStart

- Many changes can be made on the fly without restarting a web container.

Customizing Controller

- Struts 1 lets you customize the request processor per module, Struts 2 lets you customize the request handling per action, if desired.

Easy Spring Integration

- Struts 2 Actions are Spring-aware. Just need to add Spring beans.

5.5 STRUTS 2 MVC PATTERN

- Struts2 is a pull-MVC framework.
- The Model-View-Controller pattern in Struts2 is implemented with the following five core components
 - Actions

- Interceptors
 - Value Stack / OGNL
 - Results / Result types
 - View technologies
 - Struts 2 is slightly different from a traditional MVC framework, where the action takes the role of the model rather than the controller, although there is some overlap.
 - Struts 2 follows the Model-View-Controller (MVC) design patterns. The figure shown below shows how Struts 2 framework implements MVC components.
 - Action - model
 - Result - view
 - FilterDispatcher - controller
 - The role each module plays Controller's job is to map incoming HTTP requests to actions. Those mapping are defined by using XML-based configuration(struts.xml) or Java annotations.
 - The Model in Struts 2 is actions. So each action is defined and implemented by following the framework defined contract, an example consisting of an execute() method.
 - Model components consist of the data storage and business logic. Each action is an encapsulation of requests and is placed in ValueStack.
 - View is the presentation component of MVC pattern.
-

5.1 REQUEST LIFE CYCLE

● In accordance with Fig. 1, We need to understand the workflow by way of user's request life cycle in Struts 2 as shown below:

1. User sends a request to the server requesting for some resources /pages.
2. The Filter Dispatcher looks at the request and then regulates the appropriate Action.
3. Configured interceptor functionalities apply such as validation, file upload etc.
4. Selected action is performed based on the requested operation.
5. Once more, configured interceptors are applied to do any post processing if required.

Example:

Creating a Dynamic Web Project

Start your Eclipse and then go with File > New > Dynamic Web Project and enter project name as HelloWorldStruts2 and set rest of the options as given in the following screen

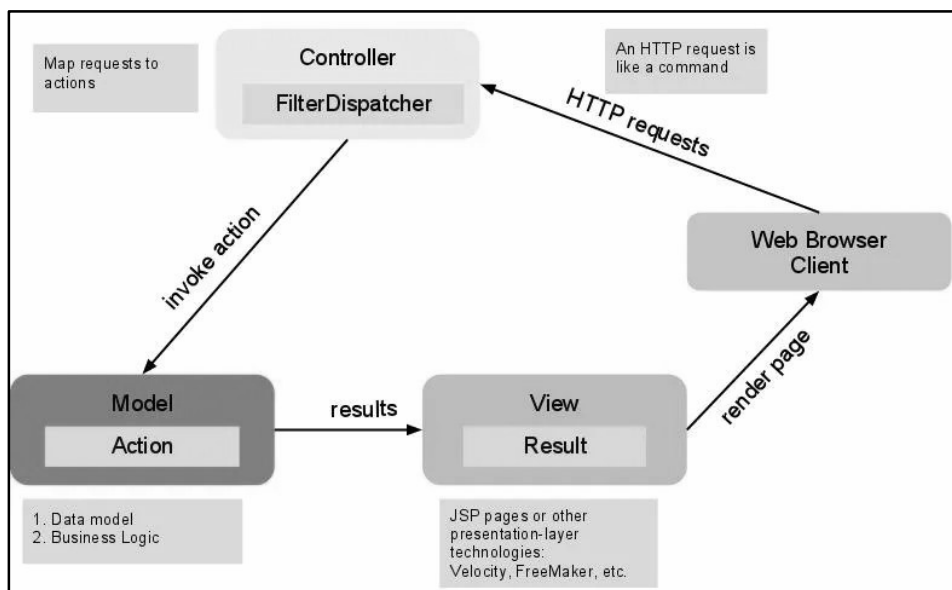
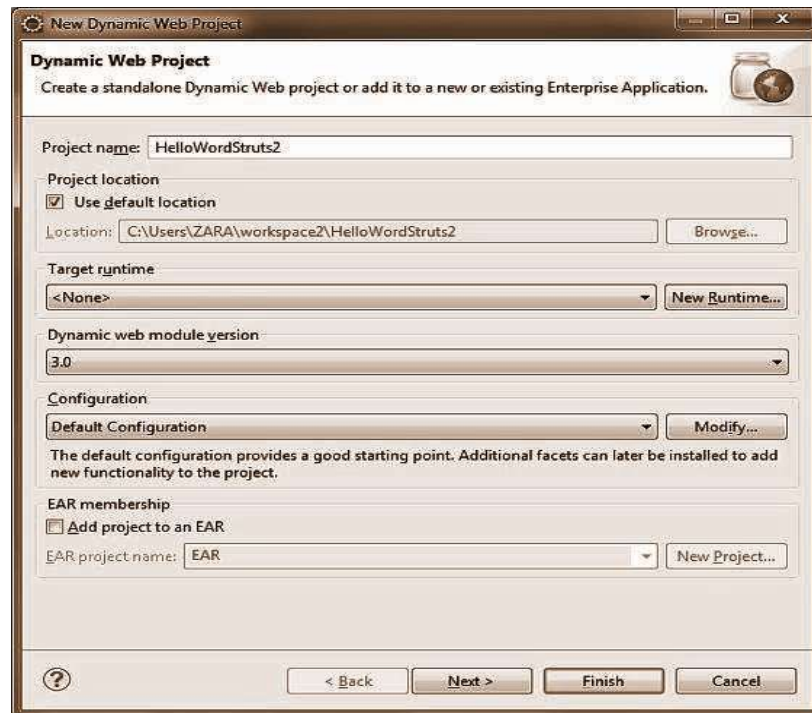
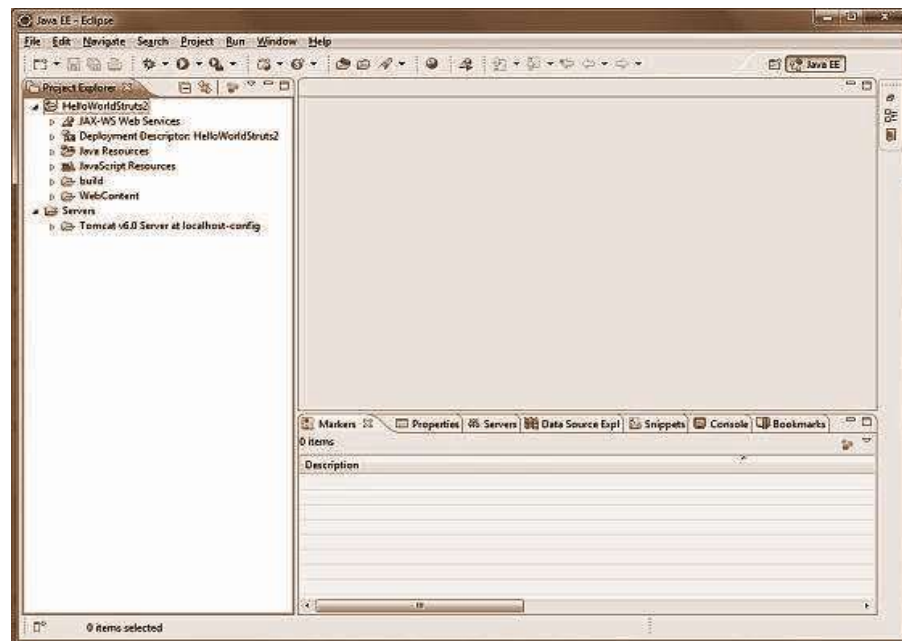


Fig. 1 Interactions among each MVC module



Select all the default options in the next screens and finally check Generate Web.xml deployment descriptor option. This will create a dynamic web project for you in Eclipse. Now go with Windows > Show View > Project Explorer, and you will see your project window something as below:



Now copy the following files from struts 2 lib folder C:\struts-2.2.3\lib to our project's WEB-INF\lib folder. To do this, you can simply drag and drop all the following files into the WEB-INF\lib folder.

- commons-fileupload-x.y.z.jar

- commons-io-x.y.z.jar
- commons-lang-x.y.jar
- commons-logging-x.y.z.jar
- commons-logging-api-x.y.jar
- freemarker-x.y.z.jar
- javassist-.xy.z.GA
- ognl-x.y.z.jar
- struts2-core-x.y.z.jar
- xwork-core.x.y.z.jar

Create Action Class

Create a java file HelloWorldAction.java under Java Resources > src with a package name com.idoluniversity.struts2 with the contents given below.

The Action class responds to a user action when the user clicks a URL. One or more of the Action class's methods are executed and a String result is returned. Based on the value of the result, a specific JSP page is rendered.

Example:

```
package com.idoluniversity.struts2; public class HelloWorldAction {  
private String name;  public String execute() throws Exception {  
    return "success";  
}  
  
public String getName() {  
    return name;  
}  
public void setName(String name) {  
    this.name = name;  
}  
}
```

Create a View

- Create the below jsp file HelloWorld.jsp in the WebContent folder in your eclipse project. To do this, right click on the WebContent folder in the project explorer and select New >JSP File.

```
<%@ page contentType = "text/html; charset = UTF-8" %>
<%@ taglib prefix = "s" uri = "/struts-tags" %>
<html>
  <head>
    <title>Hello World</title>
  </head>

  <body>
    Hello World, <s:property value = "name"/>
  </body>
</html>
```

- The taglib directive tells the Servlet container that this page will be using the Struts 2 tags and that these tags will be preceded by s.
- The s:property tag displays the value of action class property "name" which is returned by the method getName() of the HelloWAction class.

Create Main Page

- Create index.jsp in the WebContent folder. This file will serve as the initial action URL where a user can click to tell the Struts 2 framework to call a defined method of the HelloWAction class and render the HelloWorld.jsp view.

```
<%@ page language = "java" contentType = "text/html; charset =
ISO-8859-1"
    pageEncoding = "ISO-8859-1"%>
<%@ taglib prefix = "s" uri = "/struts-tags"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
  <head>
    <title>Hello World</title>
  </head>
  <body>
    <h1>Hello World From Struts2</h1>
```

```

<form action = "hello">
  <label for = "name">Please enter your name</label><br/>
  <input type = "text" name = "name"/>
  <input type = "submit" value = "Say Hello"/>
</form>
</body>
</html>

```

- Create a file called struts.xml. Since Struts 2 requires struts.xml to be present in the classes folder. Hence, create struts.xml file under the WebContent/WEB-INF/classes folder. Eclipse does not create the "classes" folder by default. Right click on the WEB-INF folder in the project explorer and select New > Folder. Your struts.xml should look like.

```

<?xml version = "1.0" Encoding = "UTF-8"?>
<!DOCTYPE struts PUBLIC
  "-//Apache Software Foundation//DTD Struts Configuration
  2.0//EN"
  "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
  <constant name = "struts.devMode" value = "true" />

  <package name = "helloworld" extends = "struts-default">
    <action name = "hello"
      class = "com.idoluniversity.struts2.HelloWAction"
      method = "execute">
      <result name = "success">/HelloWorld.jsp</result>
    </action>
  </package>
</struts>

```

- Creating a package is useful when you want to group your actions together. In our example, we named our action as "hello" which is corresponding to the URL /hello.action and is backed up by theHelloWAction.class. The execute method of HelloWAction.class is the method that is run when the URL /hello.action is invoked. If the outcome of the execute method returns "success", then we take the user to HelloWorld.jsp.

- Then create a web.xml file which is an entry point for any request to Struts 2. The entry point of Struts2 application will be a filter defined in deployment descriptor (web.xml).
- Hence, we will define an entry of org.apache.struts2.dispatcher.FilterDispatcher class in web.xml. The web.xml file needs to be created under the WEB-INF folder under WebContent. Eclipse had already created a skeleton web.xml file for you when you created the project. So, let's just modify it as follows – Example:

```
<?xml version = "1.0" Encoding = "UTF-8"?>
<web-app xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
xmlns = "http://java.sun.com/xml/ns/javaee"          xmlns:web =
"http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
xsi:schemaLocation = "http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"      id =
"WebApp_ID" version = "3.0">

    <display-name>Struts 2</display-name>

    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>

    <filter>
        <filter-name>struts2</filter-name>
        <filter-class>
org.apache.struts2.dispatcher.FilterDispatcher
        </filter-class>
    </filter>
    <filter-mapping>
        <filter-name>struts2</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
</web-app>
```

To Enable Detailed Log

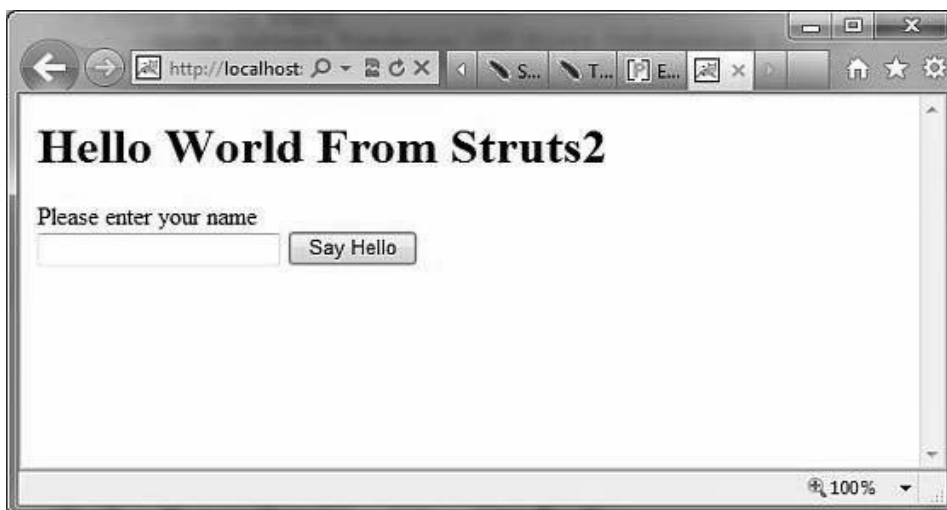
- Creating logging.properties file under WEB-INF/classes folder. Keep the following two lines in your property file

```
org.apache.catalina.core.ContainerBase.[Catalina].level = INFO
org.apache.catalina.core.ContainerBase.[Catalina].handlers = \
java.util.logging.ConsoleHandler
```

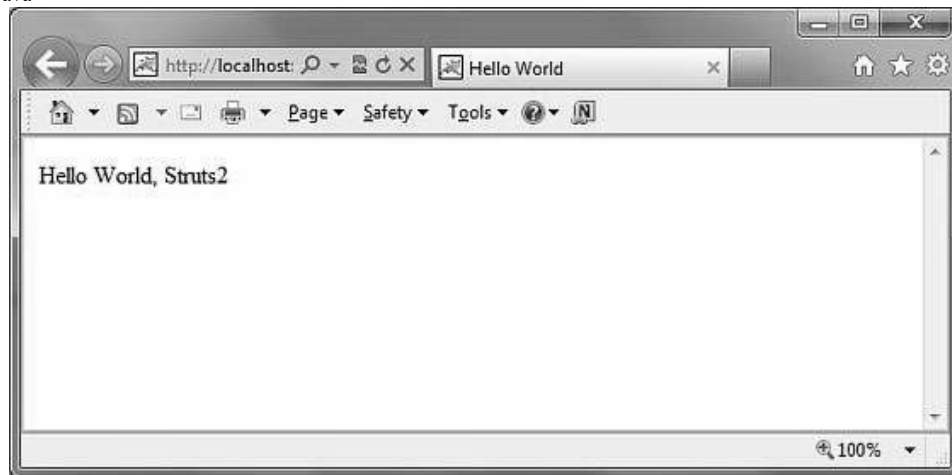
- The default logging.properties specifies a ConsoleHandler for routing logging to stdout and also a FileHandler. A handler's log level threshold can be set using SEVERE, WARNING, INFO, CONFIG, FINE, FINER, FINEST or ALL.

Procedure for Executing the Application

- Right click on the project name and click Export > WAR File to create a War file.
- Then deploy this WAR in the Tomcat's webapps directory.
- Finally, start the Tomcat server and try to access the URL <http://localhost:8080/HelloWorldStruts2/index.jsp>.
- This will give you following screen



Enter the value "Struts2" and submit the page.



5.7 CONFIGURATION FILES

- Configuration files like web.xml, struts.xml, strutsconfig.xml and struts.properties

The web.xml File

- The web.xml configuration file is a J2EE configuration file that determines how elements of the HTTP request are processed by the servlet container. It is not strictly a Struts2 configuration file, but it is a file that needs to be configured for Struts2 to work.
- The web.xml file needs to be created under the folder WebContent/WEB-INF.
- This is the first configuration file you will need to configure if you are starting without the aid of a template or tool that generates it.

Following is the content of the web.xml file which we used in our previous example.

```
<?xml version = "1.0" Encoding = "UTF-8"?>
<web-app xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
xmlns = "http://java.sun.com/xml/ns/javaee"          xmlns:web =
"http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
xsi:schemaLocation = "http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"      id =
"WebApp_ID" version = "3.0">
    <display-name>Struts 2</display-name>
    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>
    <filter>
```



```

<filter-name>struts2</filter-name>
<filter-class>
    org.apache.struts2.dispatcher.FilterDispatcher
</filter-class>
</filter>
<filter-mapping>
    <filter-name>struts2</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
</web-app>

```

The Struts.xml File

- The struts.xml file contains the configuration information that you will be modifying as actions are developed. This file can be used to override default settings for an application, for example *struts.devMode = false* and other settings which are defined in the property file. This file can be created under the folder WEBINF/classes.

```

<?xml version = "1.0" Encoding = "UTF-8"?>
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
    <constant name = "struts.devMode" value = "true" />
    <package name = "helloworld" extends = "struts-default">
        <action name = "hello"
            class = "com.idoluniversity.struts2.HelloWAction"
            method = "execute">
            <result name = "success">/HelloWorld.jsp</result>
        </action>
        <!-- more actions can be listed here -->
    </package>
    <!-- more packages can be listed here --> </struts>

```

The Struts-config.xml File

- The struts-config.xml configuration file is a link between the View and Model components in the Web Client but you would not have to touch these settings for 99.99% of your projects.

The configuration file basically contains following main elements –

Sr.No	Interceptor & Description
1	struts-config This is the root node of the configuration file.
2	form-beans This is where you map your ActionForm subclass to a name. You use this name as an alias for your ActionForm throughout the rest of the strutsconfig.xml file, and even on your JSP pages.
3	global forwards This section maps a page on your webapp to a name. You can use this name to refer to the actual page. This avoids hardcoding URLs on your web pages.
4	action-mappings This is where you declare form handlers and they are also known as action mappings.
5	controller This section configures Struts internals and rarely used in practical situations.
6	plug-in This section tells Struts where to find your properties files, which contain prompts and error messages

Following is the sample struts-config.xml file

```
<?xml version = "1.0" Encoding = "ISO-8859-1" ?>
<!DOCTYPE struts-config PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 1.0//EN"
    "http://jakarta.apache.org/struts/dtds/struts-config_1_0.dtd">
<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>

```

The Struts.properties File

- This configuration file provides a mechanism to change the default behavior of the framework.
- The values configured in this file will override the default values configured in default.properties which is contained in the struts2core-x.y.z.jar distribution.

There are a couple of properties that you might consider changing using the struts.properties file

```

#### When set to true, Struts will act much more friendly for developers
struts.devMode = true
#### Enables reloading of internationalization files

```

```

struts.i18n.reload = true
#### Enables reloading of XML configuration files
struts.configuration.xml.reload = true ####
Sets the port that the server is run on
struts.url.http.port = 8080

```

5.8 ACTIONS

- Actions are the core of the Struts2 framework, as they are for any MVC (Model View Controller) framework. Each URL is mapped to a specific action, which provides the processing logic which is necessary to service the request from the user.
- Action also serves in two other important capacities.
 - Firstly, the action plays an important role in the transfer of data from the request through to the view, whether it's a JSP or other type of result.

- Secondly, the action must assist the framework in determining which result should render the view that will be returned in the response to the request.

Create Action

- The only requirement for actions in Struts2 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.
- ActionSupport class which implements six interfaces including Action interface. The Action interface is as follows

```
public interface Action {    public static final
String SUCCESS = "success";    public static
final String NONE = "none";    public static final
String ERROR = "error";    public static final
String INPUT = "input";    public static final
String LOGIN = "login";    public String
execute() throws Exception; }
```

- Action method in the Hello World example

```
package com.idoluniversity.struts2;
public class HelloWAction {
private String name;

    public String execute() throws Exception {
        return "success";
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
}
```

- Following change to the execute method and extend the class ActionSupport as follows

```

package com.idoluniversity.struts2; import
com.opensymphony.xwork2.ActionSupport; public
class HelloWorldAction extends ActionSupport {
private String name;
    public String execute() throws Exception {
if ("SECRET".equals(name)) {      return
SUCCESS;
    } else {
        return ERROR;
    }
}
    public String getName() {
return name;
    }
    public void setName(String name) {
        this.name = name;
    }
}

```

- Then modify our struts.xml file as follows

```

<?xml version = "1.0" Encoding = "UTF-8"?>
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration
2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
    <constant name = "struts.devMode" value = "true" />
    <package name = "helloworld" extends = "struts-default">
        <action name = "hello"
            class = "com.idoluniversity.struts2.HelloWAction"
method = "execute">
            <result name = "success">/HelloWorld.jsp</result>
            <result name = "error">/AccessDenied.jsp</result>
        </action>
    </package>
</struts>

```

Create a View

- Create the below jsp file HelloWorld.jsp in the WebContent folder in your eclipse project. To do this, right click on the WebContent folder in the project explorer and select New >JSP File.

Then defined in Action interface

```
<%@ page contentType = "text/html; charset = UTF-8" %>
<%@ taglib prefix = "s" uri = "/struts-tags" %>
<html>
  <head>
    <title>Hello World</title>
  </head>
  <body>
    Hello World, <s:property value = "name"/>
  </body>
</html>
```

- Following is the code which will be invoked by the framework in case the action result is ERROR which is equal to String constant "error". Following is the content of AccessDenied.jsp

```
<%@ page contentType = "text/html; charset = UTF-8" %>
<%@ taglib prefix = "s" uri = "/struts-tags" %>
<html>
  <head>
    <title>Access Denied</title>
  </head>

  <body>
    You are not authorized to view this page.
  </body>
</html>
```

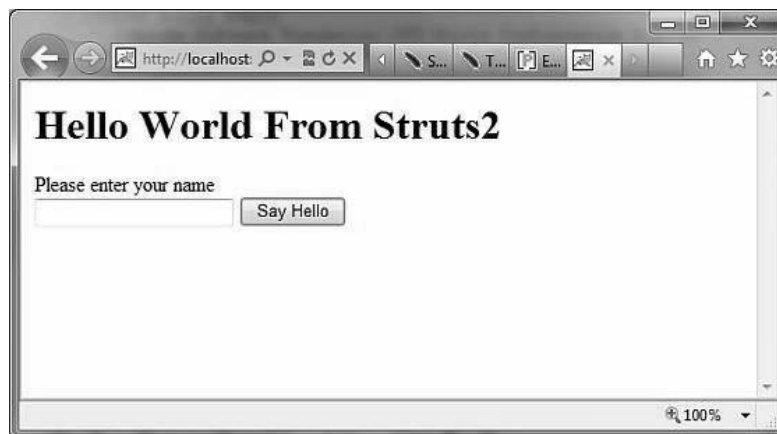
- We also need to create index.jsp in the WebContent folder. This file will serve as the initial action URL where the user can click to tell the Struts 2 framework to call the executemethod of the HelloWAction class and render the HelloWorld.jsp view.

```
<%@ page language = "java" contentType = "text/html; charset =
ISO-8859-1" pageEncoding = "ISO-8859-1"%>
<%@ taglib prefix = "s" uri = "/struts-tags"%>
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01  
Transitional//EN"  
"http://www.w3.org/TR/html4/loose.dtd">  
<html>  
  <head>  
    <title>Hello World</title>  
  </head>  
  
  <body>  
    <h1>Hello World From Struts2</h1>  
    <form action = "hello">  
      <label for = "name">Please enter your name</label><br/>  
      <input type = "text" name = "name"/>  
      <input type = "submit" value = "Say Hello"/>  
    </form>  
  </body>  
</html>
```

Execute the Application

- Right click on the project name and click Export > WAR File to create a War file. Then deploy this WAR in the Tomcat's webapps directory. Finally, start the Tomcat server and try to access the URL <http://localhost:8080/HelloWorldStruts2/index.jsp>.



- Let us enter a word as "SECRET" and you should see the following page



5.9 SUMMARY

- Apache Struts 2 is an open-source web application framework for developing Java EE web applications.
- It uses and extends the Java Servlet API to encourage developers to adopt a model view controller (MVC) architecture.

5.10. REFERENCE FOR FURTHER READING

1. Herbert Schildt, Java2: The Complete Reference, Tata McGrawHill, 5th Edition
2. Joe Wigglesworth and Paula McMillan, Java Programming: Advanced Topics, Thomson Course Technology (SPD), 3rd Edition

5.11 UNIT END EXERCISES

1. Explain the Struts 2 Basic MVC Architecture?
2. What are the Struts 2 framework features?
3. Explain the Request life cycle with the help of an example?
4. Write a short note on:
 - a. Configuration Files
 - b. Actions
5. Differentiate between struts.xml and web.xml

6 STRUTS 2 PART-II

Unit Structure:

- 6.1 Objective
- 6.2 Introduction
- 6.3 Interceptors,
- 6.4 Results & Result Types
- 6.5 Value Stack/OGNL
- 6.6 Summary
- 6.7 Reference for further reading
- 6.8 Unit End Exercises

6.1 OBJECTIVE

1. Introducing interceptors and the ValueStack.
2. Bundling actions into packages & Implementing actions.
3. To understand the role of a view in the Struts2 MVC framework.
4. To learn Object-Graph Navigation Language.

6.2 INTRODUCTION

- The tasks that are done by the Struts 2 framework before and after an Action is executed are done by Struts 2 interceptors. Interceptors are standard Java classes included in the Struts 2 core jar which are executed in a specific order.
- In our example application there is a package node in struts.xml. The package node has an attribute of extends with a value of “strutsdefault.”
- The value “struts-default” identifies to the framework the specific stack of interceptors that will be executed before and after the Actions in that package.

6.3 INTERCEPTORS

- Struts 2 framework provides a list of out-of-the-box interceptors that come pre-configured and ready to use. Few of the important interceptors are listed below.

Interceptor	Description
alias	Allows parameters to have different name aliases across requests.
checkbox	Assists in managing checkboxes by adding a parameter value of false for check boxes that are not checked.
conversionError	Places error information from converting strings to parameter types into the action's field errors.
createSession	Automatically creates an HTTP session if one does not already exist.
debugging	Provides several different debugging screens to the developer.
execAndWait	Sends the user to an intermediary waiting page while the action executes in the background.
exception	Maps exceptions that are thrown from an action to a result, allowing automatic exception handling via redirection.
fileUpload	Facilitates easy file uploading.
logger	Provides simple logging by outputting the name of the action being executed.
params	Sets the request parameters on the action.
prepare	This is typically used to do pre-processing work, such as setup database connections.
profile	Allows simple profiling information to be logged for actions.
scope	Stores and retrieves the action's state in the session or application scope.
ServletConfig	Provides the action with access to various servletbased information.

timer	Provides simple profiling information in the form of how long the action takes to execute.
token	Checks the action for a valid token to prevent duplicate form submission.
validation	Provides validation support for actions

Use of Interceptors

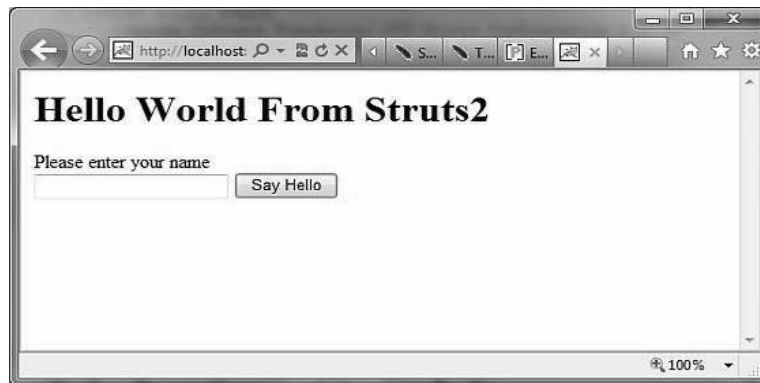
- Let us consider the existing interceptor to our "Hello World" program. ● Let's use the timer interceptor whose purpose is to measure how long it took to execute an action method.
- Then use a params interceptor whose purpose is to send the request parameters to the action.
- Then we find that the name property is not being set because the parameter is not able to reach the action.
- Let us modify the struts.xml file to add an interceptor as follows

```
<?xml version = "1.0" Encoding = "UTF-8"?>
<!DOCTYPE struts PUBLIC
  "-//Apache Software Foundation//DTD Struts Configuration
  2.0//EN"
  "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
  <constant name = "struts.devMode" value = "true" />

  <package name = "helloworld" extends = "struts-default">
    <action name = "hello"
      class = "com.idoluniversity.struts2.HelloWorldAction"
      method = "execute">
      <interceptor-ref name = "params"/>
      <interceptor-ref name = "timer" />
      <result name = "success">/HelloWorld.jsp</result>
    </action>
  </package>
</struts>
```

- Right click on the project name and click Export > WAR File to create a War file. Then deploy this WAR in the Tomcat's webapps directory.

Finally, start the Tomcat server and try to access the URL `http://localhost:8080/HelloWorldStruts2/index.jsp`. This will produce the following screen –



- Now enter any word in the given text box and click the Say Hello button to execute the defined action. Now if you will check the log generated.

```
INFO: Server startup in 3539 ms 27/08/2022 8:40:53 PM
com.opensymphony.xwork2.util.logging.commons.CommonsLog
ger info
INFO: Executed action [//hello!execute] took 109 ms.
```

Create Custom Interceptors

- Using custom interceptors the application is an elegant way to provide crosscutting application features. Creating a custom interceptor is easy, the interface that needs to be extended is the following
Interceptor interface

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

- As the names suggest, the `init()` method provides a way to initialize the interceptor, and the `destroy()` method provides a facility for interceptor cleanup.
- Unlike actions, interceptors are reused across requests and need to be thread safe, especially the `intercept()` method.

Create Interceptor Class

- Let us create the following MyInterceptor.java in Java Resources > src folder

```
package com.idoluniversity.struts2;
import java.util.*; import
com.opensymphony.xwork2.ActionInvocation; import
com.opensymphony.xwork2.interceptor.AbstractInterceptor;
public class MyInterceptor extends AbstractInterceptor {
    public String intercept(ActionInvocation invocation)throws
Exception {

        /* let us do some pre-processing */
        String output = "Pre-Processing";
        System.out.println(output);
        /* let us call action or next interceptor */
        String result = invocation.invoke();
        /* let us do some post-processing */
        output = "Post-Processing";
        System.out.println(output);
        return result;
    }
}
```

- The framework itself starts the process by making the first call to the ActionInvocation object's invoke(). Each time invoke() is called, ActionInvocation consults its state and executes whichever interceptor comes next.
- When all of the configured interceptors have been invoked, the invoke() method will cause the action itself to be executed.

The following figure shows the same concept through a request flow

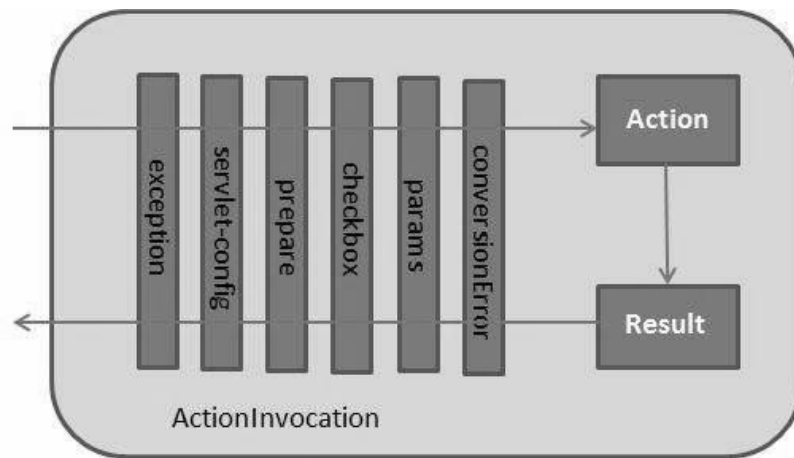


Fig.1. Request Flow

Create Action Class

- Create a java file HelloWorldAction.java under Java Resources > src with a package name com.idoluniversity.struts2 with the contents given below.

```

package com.idoluniversity.struts2; import
com.opensymphony.xwork2.ActionSupport; public
class HelloWorldAction extends ActionSupport {
private String name;

    public String execute() throws Exception {
System.out.println("Inside action....");    return "success";
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
}

```

Create a View

- Create the below jsp file HelloWorld.jsp in the WebContent folder in your eclipse project.

```

<%@ page contentType = "text/html; charset = UTF-8" %>
<%@ taglib prefix = "s" uri = "/struts-tags" %>
<html>

```

```

<head>
  <title>Hello World</title>
</head>
<body>
  Hello World, <s:property value = "name"/>
</body>
</html>

```

Create Main Page

- Also create index.jsp in the WebContent folder. This file will serve as the initial action URL where a user can click to tell the Struts 2 framework to call the a defined method of the HelloWorldAction class and render the HelloWorld.jsp view.

```

<%@ page language = "java" contentType = "text/html; charset =
ISO-8859-1"
    pageEncoding = "ISO-8859-1"%> <%@ taglib prefix = "s" uri =
"/struts-tags"%>
    <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
  <title>Hello World</title>
</head>
<body>
  <h1>Hello World From Struts2</h1>
  <form action = "hello">
    <label for = "name">Please enter your name</label><br/>
    <input type = "text" name = "name"/>
    <input type = "submit" value = "Say Hello"/>
  </form>
</body>
</html>

```

Configuration Files

- Register an interceptor and then call it as we had called the default interceptor in the previous program. To register a newly defined interceptor, the <interceptors>...</interceptors> tags are placed directly under the <package> tag insstruts.xml file.

- We can skip this step for a default interceptors as we did in our previous example. But here let us register and use it as follows

```
<?xml version = "1.0" Encoding = "UTF-8"?>
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration
    2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
    <constant name = "struts.devMode" value = "true" />
    <package name = "helloworld" extends = "struts-default">
        <interceptors>
            <interceptor name = "myinterceptorone"
                class = "com.idoluniversity.struts2.MyInterceptor" />
        </interceptors>
        <action name = "hello"
            class = "com.idoluniversity.struts2.HelloWorldAction"
            method = "execute">
            <interceptor-ref name = "params"/>
            <interceptor-ref name = "myinterceptorone" />
            <result name = "success">/HelloWorld.jsp</result>
        </action>
    </package>
</struts>
```

- The web.xml file needs to be created under the WEB-INF folder under WebContent as follows

```
<?xml version = "1.0" Encoding = "UTF-8"?>
<web-app xmlns:xsi =
    "http://www.w3.org/2002/XMLSchemainstance"
    xmlns = "http://java.sun.com/xml/ns/javaee"    xmlns:web
    = "http://java.sun.com/xml/ns/javaee/webapp_2_5.xsd"
    xsi:schemaLocation = "http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"    id =
    "WebApp_ID" version = "3.0">
    <display-name>Struts 2</display-name>
    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>
    <filter>
        <filter-name>struts2</filter-name>
```

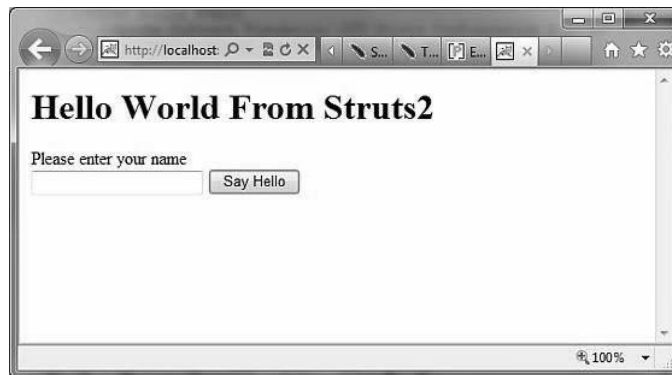


```

<filter-class>
    org.apache.struts2.dispatcher.FilterDispatcher
</filter-class>
</filter>
<filter-mapping>
    <filter-name>struts2</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
</web-app>

```

- Right click on the project name and click Export > WAR File to create a War file. Then deploy this WAR in the Tomcat's webapps directory. Finally, start the Tomcat server and try to access the URL <http://localhost:8080/HelloWorldStruts2/index.jsp>. This will produce the following screen



- Now enter any word in the given text box and click the Say Hello button to execute the defined action. Now if you will check the log generated, you will find the following text at the bottom

Pre-Processing

Inside action....

Post-Processing

6.4 RESULTS & RESULT TYPES

- The <results> tag plays the role of a view in the Struts2 MVC framework. The action is responsible for executing the business logic. The next step after executing the business logic is to display the view using the <results> tag.
- There are some navigation rules attached with the results.

- For example, if the action method is to authenticate a user, there are three possible outcomes.
- Successful Login
- Unsuccessful Login - Incorrect username or password
- Account Locked

The Dispatcher Result Type

- The dispatcher result type is the default type, and is used if no other result type is specified. It's used to forward to a servlet, JSP, HTML page, and so on, on the server. It uses the `RequestDispatcher.forward()` method.
- Provided a JSP path as the body of the result tag.

```
<result name = "success">
    /HelloWorld.jsp
</result>
```

- Specify the JSP file using a `<param name = "location">` tag within the `<result...>` element as follows

```
<result name = "success" type = "dispatcher">
    <param name = "location">
        /HelloWorld.jsp
    </param >
</result>
```

The FreeMaker Result Type

- Freemaker is a popular templating engine that is used to generate output using predefined templates.
- Let us now create a Freemaker template file called `hello.fm` with the following contents

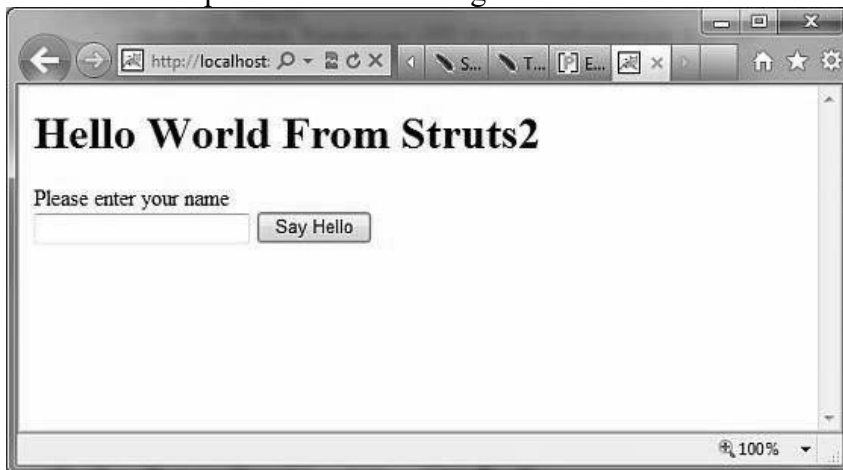
Hello World \${name}

- The above file is a template where `name` is a parameter which will be passed from outside using the defined action. keep this file in `CLASSPATH`.

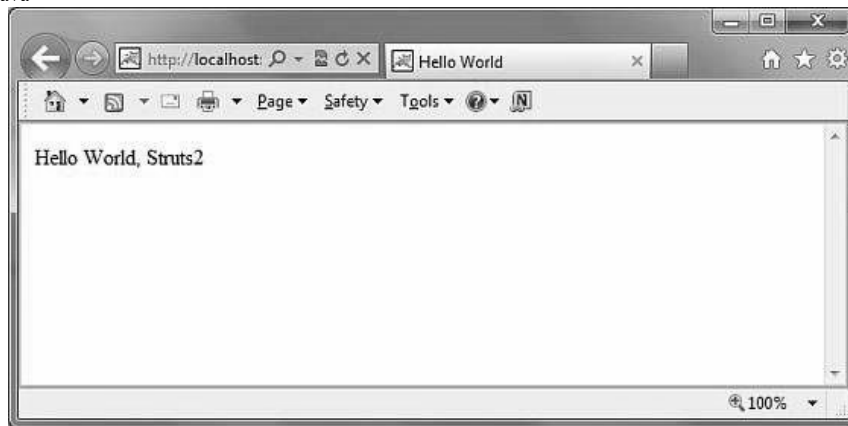
- Next, let us modify the struts.xml to specify the result as follows

```
<?xml version = "1.0" Encoding = "UTF-8"?>
<!DOCTYPE struts PUBLIC
"-//Apache Software Foundation//DTD Struts Configuration
2.0//EN"
"http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
  <constant name = "struts.devMode" value = "true" />
  <package name = "helloworld" extends = "struts-default">
    <action name = "hello"
      class = "com.idoluniversity.struts2.HelloWorldAction"
      method = "execute">
      <result name = "success" type = "freemarker">
        <param name = "location">/hello.fm</param>
      </result>
    </action>
  </package>
</struts>
```

- Let us keep HelloWorldAction.java, HelloWorldAction.jsp and index.jsp files
- Now Right click on the project name and click Export > WAR File to create a War file.
- Then deploy this WAR in the Tomcat's webapps directory. Finally, start the Tomcat server and try to access the URL <http://localhost:8080/HelloWorldStruts2/index.jsp>. This will produce the following screen .



Enter the value "Struts2" and submit the page. You should see the next page.



The Redirect Result Type

- The redirect result type calls the standard `response.sendRedirect()` method, causing the browser to create a new request to the given location.
- Provide the location either in the body of the `<result...>` element or as a `<param name = "location">` element. Redirect also supports the `parse` parameter. Here's an example configured using XML

```
<action name = "hello"
  class = "com.idoluniversity.struts2.HelloWorldAction"  method
= "execute">
  <result name = "success" type = "redirect">
    <param name = "location">
      /NewWorld.jsp
    </param >
  </result>
</action>
```

6.5 VALUE STACK/OGNL

The Value Stack

- The value stack is a set of several objects which keeps the following objects in the provided order

Sr.No	Objects & Description
-------	-----------------------

1	Temporary Objects There are various temporary objects which are created during execution of a page. For example the current iteration value for a collection being looped over in a JSP tag.
2	The Model Object If you are using model objects in your struts application, the current model object is placed before the action on the value stack.
3	The Action Object This will be the current action object which is being executed.
4	Named Objects These objects include #application, #session, #request, #attr and #parameters and refer to the corresponding servlet scopes.

- The value stack can be accessed via the tags provided for JSP, Velocity or Freemarker.
- get valueStack object inside your action as follows
 ActionContext.getContext().getValueStack()
- Once you have a ValueStack object, you can use the following methods to manipulate that object

Sr.No	ValueStack Methods & Description
1	Object findValue(String expr) Find a value by evaluating the given expression against the stack in the default search order.
2	CompoundRoot getRoot() Get the CompoundRoot which holds the objects pushed onto the stack.
3	Object peek() Get the object on the top of the stack without changing the stack.

4	Object pop() Get the object on the top of the stack and remove it from the stack.
5	void push(Object o) Put this object onto the top of the stack.
6	void set(String key, Object o) Sets an object on the stack with the given key so it is retrievable by findValue(key,...)
7	void setDefaultType(Class defaultType) Sets the default type to convert to if no type is provided when getting a value.
8	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.
9	int size() Get the number of objects in the stack.

The OGNL

- The Object-Graph Navigation Language (OGNL) is a powerful expression language that is used to reference and manipulate data on the ValueStack.
- OGNL also helps in data transfer and type conversion.
- The OGNL is very similar to the JSP Expression Language.
- OGNL is based on the idea of having a root or default object within the context.
- The properties of the default or root object can be referenced using the markup notation, which is the pound symbol.
- OGNL is based on a context and Struts builds an ActionContext map for use with OGNL. The ActionContext map consists of the following

Application – Application scoped variables

Session – Session scoped variables

Root / value stack – All your action variables are stored here

Request – Request scoped variables

Parameters – Request parameters

Attributes – The attributes stored in page, request, session and application scope

- It is important to understand that the Action object is always available in the value stack. So, therefore if an Action object has properties “x” and “y” they are readily available to use.
- Objects in the ActionContext are referred to using the pound symbol, however, the objects in the value stack can be directly referenced.
- For example, if employee is a property of an action class, then it can be referenced as follows

`<s:property value = "name"/>` instead
of

`<s:property value = "#name"/>`

- If you have an attribute in session called "login" you can retrieve it as follows

`<s:property value = "#session.login"/>`

- OGNL also supports dealing with collections - namely Map, List and Set. For example to display a dropdown list of colors, you could do

`<s:select name = "color" list = "{ 'red','yellow','green' }" />`

- The OGNL expression is clever to interpret the
"red","yellow","green" as colors and build a list based on that.

ValueStack/OGNL Example

Create Action

- Let us consider the following action class where we are accessing valueStack and then setting a few keys which we will access using OGNL in our view, i.e., JSP page.

```

package com.idoluniversity.struts2;
import java.util.*;
import com.opensymphony.xwork2.util.ValueStack;
import com.opensymphony.xwork2.ActionContext;
import com.opensymphony.xwork2.ActionSupport; public
class HelloWorldAction extends ActionSupport {
    private String name;

    public String execute() throws Exception {
        ValueStack          stack
        ActionContext.getContext().getValueStack();
        Map<String, Object> context = new HashMap<String,
Object>();    context.put("key1", new String("This
is key1"));    context.put("key2", new
String("This is key2"));

        stack.push(context);
        System.out.println("Size of the valueStack: " + stack.size());
        return "success";
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
}

```

Create Views

- Let us create the below jsp file HelloWorld.jsp in the WebContent folder in your eclipse project. This view will be displayed in case action returns success

```

<%@ page contentType = "text/html; charset = UTF-8" %>
<%@ taglib prefix = "s" uri = "/struts-tags" %>
<html>
<head>
    <title>Hello World</title>
</head>

<body>

```



```

Entered value : <s:property value = "name"/><br/>
Value of key 1 : <s:property value = "key1" /><br/>
Value of key 2 : <s:property value = "key2" /> <br/>
</body>
</html>

```

- Also create index.jsp in the WebContent folder whose content is as follows

```

<%@ page language = "java" contentType = "text/html; charset =
ISO-8859-1" pageEncoding = "ISO-8859-1"%>
<%@ taglib prefix = "s" uri = "/struts-tags"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Hello World</title>
</head>
<body>
<h1>Hello World From Struts2</h1>
<form action = "hello">
<label for = "name">Please enter your name</label><br/>
<input type = "text" name = "name"/>
<input type = "submit" value = "Say Hello"/>
</form>
</body>
</html>

```

Configuration Files

Following is the content of struts.xml file

```

<?xml version = "1.0" Encoding = "UTF-8"?>
<!DOCTYPE struts PUBLIC
"-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
"http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
<constant name = "struts.devMode" value = "true" />
<package name = "helloworld" extends = "struts-default">

```

```

    <action name = "hello"
        class = "com.idoluniversity.struts2.HelloWorldAction"
    method = "execute">
        <result name = "success">/HelloWorld.jsp</result>
    </action>
</package>
</struts>

```

- Following is the content of web.xml file

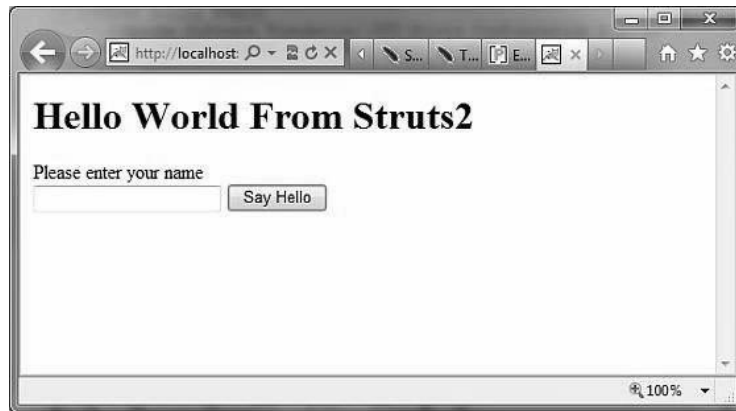
```

<?xml version = "1.0" Encoding = "UTF-8"?>
<web-app xmlns:xsi =
"http://www.w3.org/2002/XMLSchemainstance"
    xmlns = "http://java.sun.com/xml/ns/javaee"
    xmlns:web =
"http://java.sun.com/xml/ns/javaee/webapp_2_5.xsd"
    xsi:schemaLocation = "http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id =
"WebApp_ID" version = "3.0">
    <display-name>Struts 2</display-name>
    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>
    <filter>
        <filter-name>struts2</filter-name>
        <filter-class>
            org.apache.struts2.dispatcher.FilterDispatcher
        </filter-class>
    </filter>
    <filter-mapping>
        <filter-name>struts2</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
</web-app>

```

- Right click on the project name and click Export > WAR File to create a War file. Then deploy this WAR in the Tomcat's webapps directory. Finally, start the Tomcat server and try to access the URL

<http://localhost:8080/HelloWorldStruts2/index.jsp>. This will produce the following screen



- Now enter any word in the given text box and click the "Say Hello" button to execute the defined action. Now, if you will check the log generated, you will find the following text at the bottom

Size of the valueStack: 3

6.6 SUMMARY

- Struts 2 framework provides a list of out-of-the-box interceptors that come pre-configured and ready to use.
- The <results> tag plays the role of a view in the Struts2 MVC framework. The action is responsible for executing the business logic.
- The value stack is a set of several objects.
- The Object-Graph Navigation Language (OGNL) is a powerful expression language that is used to reference and manipulate data on the ValueStack.

6.7 REFERENCE FOR FURTHER READING

1. Herbert Schildt, Java2: The Complete Reference, Tata McGrawHill, 5th Edition
2. Joe Wigglesworth and Paula McMillan, Java Programming: Advanced Topics, Thomson Course Technology (SPD), 3rd Edition

6.8 UNIT END EXERCISES

1. What are interceptors?
2. Explain the Results & Result Types?
3. Write a short note on Value Stack & OGNL.
4. What is value stack in struts? State and explain the execution flow of value stack.
5. What is OGNL? What does it consist of? How its contents can be accessed?

7 JSON

Unit Structure:

- 7.1 Objective
- 7.2 Introduction
- 7.3 Overview,
- 7.4 Syntax,
- 7.5 DataTypes,
- 7.6 Objects,
- 7.7 Schema,
- 7.8 Comparison with XML,
- 7.9 JSON with Java
- 7.10 Summary
- 7.11 Reference for further reading
- 7.12 Unit End Exercises

7.1 OBJECTIVE

- To study the overview of JSON.
 - To understand the JSON syntax, DataTypes & Objects.
 - To learn the use of annotating and validating JSON documents.
 - To understand the difference between JSON & XML and JSON with Java.
-

7.2 INTRODUCTION

- JSON is an acronym for JavaScript Object Notation, is an open standard format, which is lightweight and text-based, designed explicitly for human-readable data interchange.
- It is a language-independent data format.
- It supports almost every kind of language, framework, and library.
- JSON is an open standard for exchanging data on the web. It supports data structures like objects and arrays. So, it is easy to write and read data from JSON.
- In JSON, data is represented in key-value pairs and curly braces hold objects, where a colon is followed after each name. The comma is used to separate key-value pairs. Square brackets are used to hold arrays, where each value is comma separated.

7.3 OVERVIEW

- JSON stands for JavaScript Object Notation.
- JSON is an open standard data-interchange format.
- JSON is lightweight and self-describing.
- JSON originated from JavaScript.
- JSON is easy to read and write. ● JSON is language independent.
- JSON supports data structures such as arrays and objects.

Features of JSON

- Simplicity
- Openness
- Self-Describing
- Internationalization
- Extensibility ● Interoperability

Advantages of JSON:

- **Less Verbose:** In contrast to XML, JSON follows a compact style to improve its users' readability. While working with a complex system, JSON tends to make substantial enhancements.
- **Faster:** The JSON parsing process is faster than that of the XML because the DOM manipulation library in XML requires extra memory for handling large XML files. However, JSON requires less data that ultimately results in reducing the cost and increasing the parsing speed.
- **Readable:** The JSON structure is easily readable and straightforward. Regardless of the programming language that you are using, you can easily map the domain objects.
- **Structured Data:** In JSON, a map data structure is used, whereas XML follows a tree structure. The key-value pairs limit the task but facilitate the predictive and easily understandable model.

7.4 SYNTAX

- JSON structure is based on the JavaScript object literal syntax; they share a number of similarities.
- The core elements of JSON syntax:
 - Data is presented in key/value pairs.
 - Data elements are separated by commas.
 - Curly brackets {} determine objects.
 - Square brackets [] designate arrays.
- As a result, JSON object literal syntax looks like this:

{“key”:“value”,“key”:“value”,“key”:“value”}. Example:

```
{
  "book": [
    {
      "id": "01",
      "language": "Java",
      "edition": "third",
      "author": "Herbert Schildt"
    },
    {
      "id": "07",
      "language": "C++",
      "edition": "second",
      "author": "E.Balagurusamy"
    }
  ]
}
```

7.5 DATATYPES

JSON supports the following data types

Sr.No.	Type & Description
1	Number double- precision floating-point format in JavaScript
2	String double-quoted Unicode with backslash escaping

3	Boolean true or false
4	Array an ordered sequence of values
Sr.No.	Type & Description
5	Value it can be a string, a number, true or false, null etc
6	Object an unordered collection of key:value pairs
7	Whitespace can be used between any pair of tokens
8	null empty

1. Number

- It is a double precision floating-point format in JavaScript and it depends on implementation.
- Octal and hexadecimal formats are not used.
- No NaN or Infinity is used in Number. The following table shows the number types –

Sr.No.	Type & Description
1	Integer Digits 1-9, 0 and positive or negative
2	Fraction Fractions like .3, .9
3	Exponent Exponent like e, e+, e-, E, E+, E-

Syntax & Example:

```
var json-object-name = { string : number_value, .....} var
obj = {marks: 97}
```

2. String

- It is a sequence of zero or more double quoted Unicode characters with backslash escaping.

- Character is a single character string i.e. a string with length 1.

Sr.No.	Type & Description
1	" double quotation
2	\ backslash
3	/ forward slash
Sr.No.	Type & Description
4	b backspace
5	f form feed
6	n new line
7	r carriage return
8	t horizontal tab
9	u four hexadecimal digits

Syntax & Example

```
var json-object-name = { string : "string value", .....} var
obj = {name: 'Amit'}
```

3. Boolean

- It includes true or false values.

Syntax & Example

```
var json-object-name = { string : true/false, .....} var
obj = {name: 'Amit', marks: 97, distinction: true}
```

4. Array

- It is an ordered collection of values.
- These are enclosed in square brackets which means that array begins with `[` and ends with `]`.
- The values are separated by `,` (comma).
- Array indexing can be started at 0 or 1.
- Arrays should be used when the key names are sequential integers.

Syntax & Example

[value,]

```
{
  "books": [
    { "language": "Java" , "edition": "second" },
    { "language": "C++" , "lastName": "fifth" },
    { "language": "C" , "lastName": "third" }
  ]
}
```

5. Object

- It is an unordered set of name/value pairs.
- Objects are enclosed in curly braces, that is, it starts with '{' and ends with '}'.
- Each name is followed by ':' (colon) and the key/value pairs are separated by , (comma).
- The keys must be strings and should be different from each other.
- Objects should be used when the key names are arbitrary strings.

Syntax & Example:

{ string : value,

```
{
  "id": "011A",
  "language": "JAVA",
  "price": 500,
}
```

6. Whitespace

- It can be inserted between any pair of tokens. It can be added to make a code more readable. Example shows declaration with and without whitespace – Syntax & Example:

{string:" ",....}

```
var obj1 = {"name": "Sachin Tendulkar"}
var obj2 = {"name": "SauravGanguly"}
```

7. Null

- It means empty type.

Syntax & Example:

Null

```
var i = null if(i
== 1) {
    document.write("<h1>value is 1</h1>");
}
else
{
    document.write("<h1>value is null</h1>"); }
```

8. Value

- It includes
 - number (integer or floating point)
 - String
 - Boolean
 - Array
 - Object
 - Null

Syntax & Example

String | Number | Object | Array | TRUE | FALSE | NULL

```
var i = 1; var j
= "sachin"; var
k = null;
```

7.6 OBJECTS

- JSON objects can be created with JavaScript. Let us see the various ways of creating JSON objects using JavaScript.
- Creation of an empty Object var JSONObj =
{};

- Creation of a new Object `var jsonObj = new Object();`
- Creation of an object with attribute bookname with value in string, attribute price with numeric value. Attributes are accessed by using ' '
Operator – `var jsonObj = { "bookname ":"VB BLACK BOOK", "price":500 };`

Example that shows creation of an object in javascript using JSON, save the below code as json_object.htm

```
<html>
  <head>
    <title>Creating Object JSON with JavaScript</title>
    <script language = "javascript" >      var jsonObj = { "name" :
"mumbaiuniversity.com", "year" :
2005 };
    document.write("<h1>JSON with JavaScript example</h1>");
    document.write("<br>");
    document.write("<h3>Website Name =
"+jsonObj.name+"</h3>");
    document.write("<h3>Year = "+jsonObj.year+"</h3>");
    </script>
  </head>
  <body>
    </body>
</html>
```

7.7 SCHEMA

- JSON Schema is a specification for JSON based format for defining the structure of JSON data.
- Describe your existing data format.
 - Clear, human- and machine-readable documentation.
 - Complete structural validation, useful for automated testing.
 - Complete structural validation, validating client-submitted data. •

JSON Schema Validation Libraries

There are several validators currently available for different programming languages. Currently the most complete and compliant JSON Schema validator available is JSV.

Languages	Libraries
C	WJElement (LGPLv3)
Java	json-schema-validator (LGPLv3)
.NET	Json.NET (MIT)
ActionScript 3	Frigga (MIT)
Haskell	aeson-schema (MIT)
Python	Jsonschema
Ruby	autoparse (ASL 2.0); ruby-jsonschema (MIT)
PHP	php-json-schema (MIT). json-schema (Berkeley)
JavaScript	Orderly (BSD); JSV; json-schema; Matic (MIT); Dojo; Persevere (modified BSD or AFL 2.0); schema.js.

Example

Given below is a basic JSON schema, which covers a classical product catalog description –

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "Product",
  "description": "A product from Acme's catalog",
  "type": "object",
  "properties": {

    "id": {
      "description": "The unique identifier for a product",
      "type": "integer"
    },

    "name": {
      "description": "Name of the product",
      "type": "string"
    },

    "price": {
      "type": "number",
      "minimum": 0,
      "exclusiveMinimum": true
    }
  },

  "required": ["id", "name", "price"] }
}
```

- Various important keywords that can be used in this schema –

Sr.No.	Keyword & Description
1	\$schema The \$schema keyword states that this schema is written according to the draft v4 specification.
2	title You will use this to give a title to your schema.
3	description A little description of the schema.
4	type The type keyword defines the first constraint on our JSON data: it has to be a JSON Object.
5	properties Defines various keys and their value types, minimum and maximum values to be used in JSON file.

6	required
Sr.No.	Keyword & Description
	This keeps a list of required properties.
7	minimum This is the constraint to be put on the value and represents minimum acceptable value.
8	exclusiveMinimum If "exclusiveMinimum" is present and has boolean value true, the instance is valid if it is strictly greater than the value of "minimum".
9	maximum This is the constraint to be put on the value and represents maximum acceptable value.
10	exclusiveMaximum If "exclusiveMaximum" is present and has boolean value true, the instance is valid if it is strictly lower than the value of "maximum".
11	multipleOf A numeric instance is valid against "multipleOf" if the result of the division of the instance by this keyword's value is an integer.
12	maxLength The length of a string instance is defined as the maximum number of its characters.
13	minLength The length of a string instance is defined as the minimum number of its characters.
14	pattern A string instance is considered valid if the regular expression matches the instance successfully.

7.8 COMPARISON WITH XML

Similarities between the JSON and XML.

- **Self-describing:** Both json and xml are self-describing as both xml data and json data are human-readable text.

- **Hierarchical:** Both json and xml support hierarchical structure. Here hierarchical means that the values within values.
- **Data interchange format:** JSON and XML can be used as data interchange formats by many different programming languages.
- **Parse:** Both the formats can be easily parsed.
- **Retrieve:** Both formats can be retrieved by using HTTP requests. The methods used for retrieving the data are GET, PUT, POST.

Differences between the JSON and XML

JSON	XML
JSON stands for javascript object notation.	XML stands for an extensible markup language.
The extension of json file is .json.	The extension of xml file is .xml.
The internet media type is application/json.	The internet media type is application/xml or text/xml.
The type of format in JSON is data interchange.	The type of format in XML is a markup language.
It is extended from javascript.	It is extended from SGML.
It is open source means that we do not have to pay anything to use JSON.	It is also open source.
The object created in JSON has some type.	XML data does not have any type.
The data types supported by JSON are strings, numbers, Booleans, null, array.	XML data is in a string format.
It does not have any capacity to display the data.	XML is a markup language, so it has the capacity to display the content.
JSON has no tags.	XML data is represented in tags, i.e., start tag and end tag.
	XML file is larger. If we want to represent the data in XML then it would create a larger file as compared to JSON.

JSON is quicker to read and write.	XML file takes time to read and write because the learning curve is higher.
JSON can use arrays to represent the data.	XML does not contain the concept of arrays.
It can be parsed by a standard javascript function. It has to be parsed before use.	XML data which is used to interchange the data, must be parsed with respective to their programming language to use that.
It can be easily parsed and little bit code is required to parse the data.	It is difficult to parse.
File size is smaller as compared to XML.	File size is larger.
JSON	XML
JSON is data-oriented.	XML is document-oriented.
It is less secure than XML.	It is more secure than JSON.

7.9 JSON WITH JAVA

- Encode and decode JSON objects using Java programming language.

Environment

- Install any of the JSON modules available.
- For this download and installed JSON.simple and have added the location of **json-simple-1.1.1.jar** file to the environment variable CLASSPATH.

Mapping between JSON and Java entities

- JSON.simple maps entities from the left side to the right side while decoding or parsing, and maps entities from the right to the left while encoding.

JSON	Java
string	java.lang.String
number	java.lang.Number

true false	java.lang.Boolean
null	null
array	java.util.List
object	java.util.Map

Encoding JSON in Java

- To encode a JSON object using Java JSONObject which is a subclass of java.util.HashMap. No ordering is provided. If you need the strict ordering of elements, use JSONValue.toJSONString (map) method with ordered map implementation such as java.util.LinkedHashMap.

```
import org.json.simple.JSONObject;
class JsonEncodeDemo {
    public static void main(String[] args) {
        JSONObject obj = new JSONObject();

        obj.put("name", "foo");
        obj.put("num", new Integer(100));
        obj.put("balance", new
Double(1000.21));
        obj.put("is_vip", new Boolean(true));

        System.out.print(obj);
    }
} Output:
{"balance": 1000.21, "num":100, "is_vip":true, "name":"foo"}
```

Decoding JSON in Java

- The following example makes use of **JSONObject** and **JSONArray** where JSONObject is a java.util.Map and JSONArray is a java.util.List, so you can access them with standard operations of Map or List.

```
import org.json.simple.JSONObject; import
org.json.simple.JSONArray; import
org.json.simple.parser.ParseException; import
org.json.simple.parser.JSONParser;

class JsonDecodeDemo {

    public static void main(String[] args) {

        JSONParser parser = new JSONParser();
        String s =
"[0,{\"1\":{\"2\":{\"3\":{\"4\":[5,{\"6\":7}]}}}}]";

        try{
            Object obj = parser.parse(s);
            JSONArray array = (JSONArray)obj;

            System.out.println("The 2nd element of array");
            System.out.println(array.get(1));
            System.out.println();
            JSONObject obj2 = (JSONObject)array.get(1);
            System.out.println("Field \"1\"");

            System.out.println(obj2.get("1"));
            s = "{}";
            obj = parser.parse(s);
            System.out.println(obj);
            s = "[5,]";
```

```

        obj = parser.parse(s);
        System.out.println(obj);
        s = "[5,,2]";
        obj = parser.parse(s);
        System.out.println(obj);
    } catch(ParseException pe) {
        System.out.println("position: " + pe.getPosition());
        System.out.println(pe);
    }
}
}

```

Output:

The 2nd element of array

```
{"1":{"2":{"3":{"4":[5,{"6":7}]}}}}
```

Field "1"

```
{"2":{"3":{"4":[5,{"6":7}]}}}
```

```
{}
```

```
[5]
```

```
[5,2]
```

7.10 SUMMARY

1. JSON is an acronym for JavaScript Object Notation, is an open standard format, which is lightweight and text-based, designed explicitly for human-readable data interchange
2. JSON stands for JavaScript Object Notation.
3. JSON is language independent.
4. JSON structure is based on the JavaScript object literal syntax, they share a number of similarities.
5. JSON objects can be created with JavaScript
6. JSON Schema is a specification for JSON based format for defining the structure of JSON data.

7.11 REFERENCE FOR FURTHER READING

1. Herbert Schildt, Java2: The Complete Reference, Tata McGrawHill, 5th Edition

2. Joe Wigglesworth and Paula McMillan, Java Programming: Advanced Topics, Thomson Course Technology (SPD) , 3rd Edition

7.12 UNIT END EXERCISES

1. Mention what is the rule for JSON syntax rules? Give an example of JSON object?
2. Explain the difference between JSON & XML.
3. Explain the difference between JSON & Java.
4. What are the data types supported by JSON?
5. What are the advantages of JSON?
6. Explain the role of JSON object in detail.
7. Explain what do you mean by JSON Schema? Explain it with suitable example?
8. Explain the steps involved in executing JSON with java.
