Fundamentals of Java Enterprise Components

Session: 3

Introduction to
Web Application Development

Objectives

- Explain the basics of Web application development
- Describe the different components of a Web application and how they interact with each other
- Describe the protocols and communication standards associated with Web applications
- Explain the basic process of creating a Web application through the Netbeans IDE
- Describe the technologies used to develop Web applications
- Describe Web services and standards associated with Web services

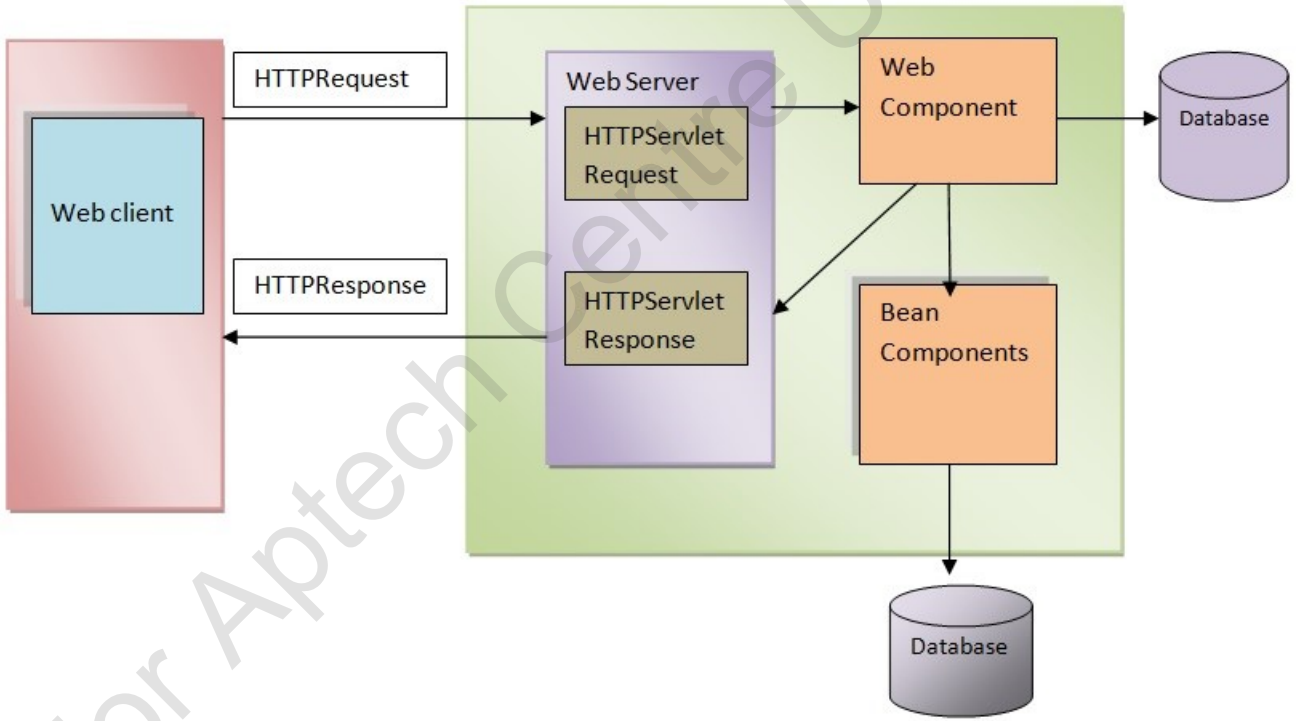
Introduction to Web Applications

Data flow in a Web application:

- Starts from the client application when it sends a HTTPRequest to the server.
- HTTPRequest received by the Web server and appropriate servlet is invoked. The Web server converts the HTTPRequest object to a HTTPServletRequest object.
- Servlets interact with the business component of the application and generate Web pages as a response to the request.
- The Web pages are then sent to the client over the Internet through HTTP protocol.

Web Application Components

The following figure shows various components of the Web application and interaction among them:



Web Application Life Cycle

- Decide the scope and requirement of the application.
- Decide on the application architecture, whether to have a two-tier or three-tier architecture.
- Identify the functional layers and also the required components such as database.
- Create UML diagrams representing the classes and objects in the application.
- Identify the required technology to be used to develop the components of the Web application.

Web Application Development

Develop code for different components of the application such as Web pages, CSS style sheets for Web pages, Servlets, JSPs, and JSFs.

Define the deployment descriptor of the application.

- It is a XML file which has instructions for the deployment of the application.
- web.xml is the default deployment descriptor located in WEB-INF directory.

Compile the Web application along with the required helper classes.

The application may be packaged into an archive file for deployment.

Deploy the application into the Web container, after this the application can be accessed through a URL.

HTTP Request and Response

- HTTP is the standard protocol for communication over the Internet.
- It functions according to request-response model.
- It uses services of other protocols such as TCP and UDP, where TCP is Transmission Control Protocol and UDP is User Datagram Protocol.
- HTTP identifies various resources through URLs (Universal Resource Locators).
- It has a predefined set of methods which are used by various components on the Internet to communicate with each other.

HTTP Methods 1-2

Following table shows various methods of HTTP that can be used by the developer:

Method	Description
GET	Retrieves data from a server, where the server is identified through a URL. Data sent using GET is visible in the URL.
PUT	Creates a resource or modifies an existing resource on the server. In this method, the URL is also required to identify the server.
POST	Submits certain values to the server, such as values from a Web form after submission of the form and so on. Data sent using POST is not visible in the URL.
DELETE	As the name suggests, DELETE method is used to delete resources.
TRACE	This method is used to identify the servers between the HTTP client and server. It is generally used to obtain routing statistics.

HTTP Methods 2-2

Method	Description
OPTIONS	This method can be used to check the functionalities that can be extended by the Web server for certain URLs.
CONNECT	This method is used to connect to the server. The connection may or may not be secure. It may be a TCP connection or UDP connection. All these properties are defined while creating a connection.
PATCH	Patch generally means a partial modification. For example, partial modifications to the HTTP server.

Java Classes for HTTP Communication

Java provides the following classes for HTTP communication:

- HttpURLConnection - Handles all the HTTP connectivity related tasks in Java EE7.
- HttpServletRequest – Creates an object of servlet request which uses HTTP.
- HttpServletResponse – Creates an object of servlet response which also uses HTTP.
- HttpsURLConnection – Used to make secure URL connections.
- HTTP.HttpRequest – Used to create HTTP request.
- HttpResponse - Used to create HTTP response.

Introduction to Web Components 1-2

JavaServerPages

- Creates dynamic interactive pages
- Has a standard tag library

Servlets

- Server-side programs for processing client requests
- javax.servlet and javax.servlet.http are the packages in Java used for servlet implementation

Java Beans

- Implements the business logic of the application
- Java beans implements Serializable interface

Introduction to Web Components 2-2

JavaServerFaces

- Used to develop component based user interface for Web applications
- Makes use of view templates known as Facelets
- Allows the developer to create user interface components from templates
- Supports JSP tags to access the user interface components
- Saves the state of the UI components, transparently
- Capable of event handling and component rendering mechanisms, which makes it compatible with other markup languages, such as HTML

Introduction to Web Services 1-2

Web services are services which can be accessed by the client programs over the Internet.

Services are accessed through Web browsers.

These services can also be used by other service providers on the Internet.

Various standards and technologies are used for implementation of Web services such as SOAP, REST, XML, and so on.

Introduction to Web Services 2-2

Simple Object Access Protocol (SOAP) Web services

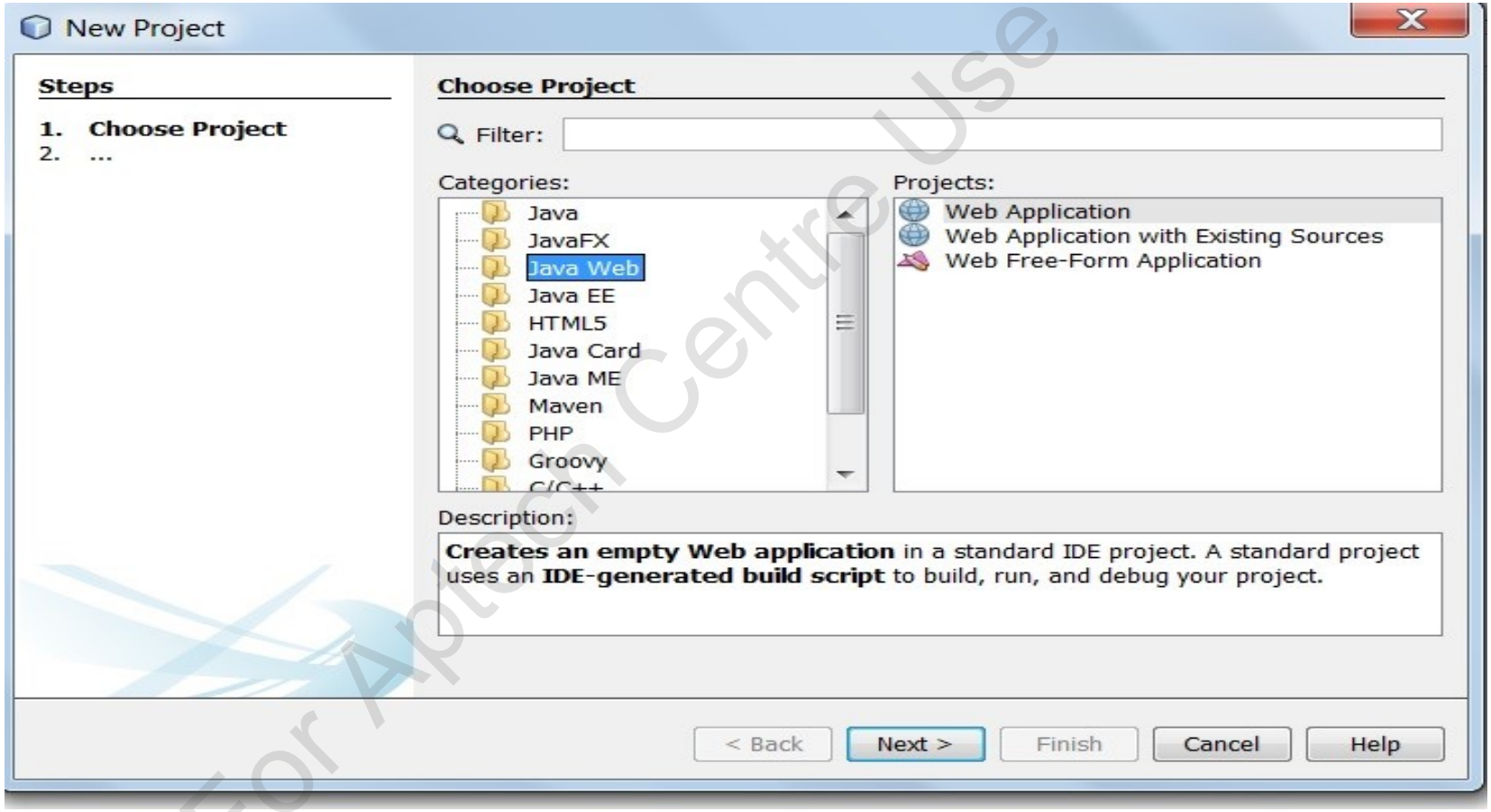
- JAX-WS is the Java specification for implementing SOAP Web services.

Representational State Transfer (REST) Web Services

- JAX-RS is the Java specification for implementing REST Web services.

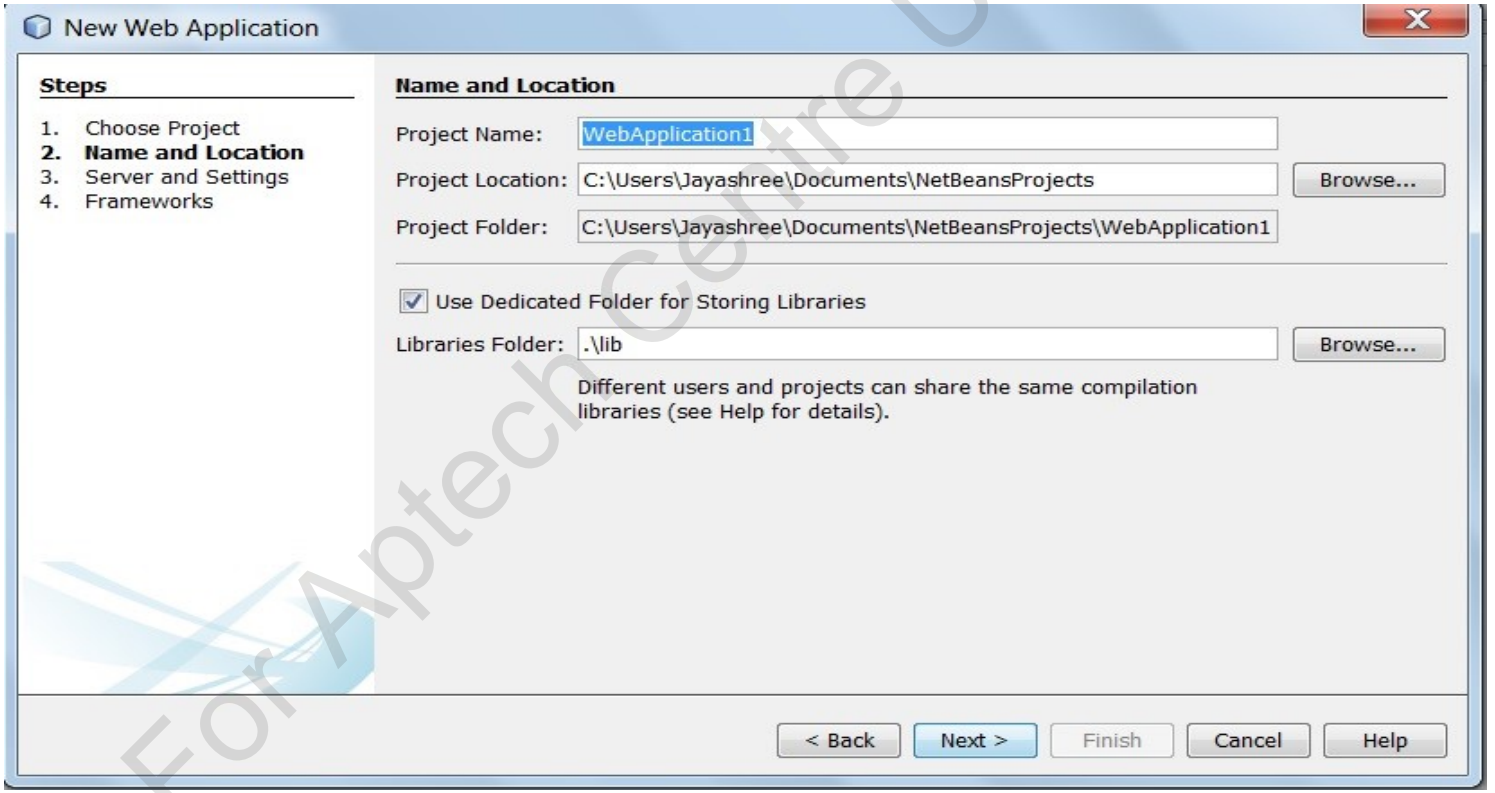
Creating a Simple Web Application 1-16

Select File → New Project. The New Project dialog box is displayed as shown in the following figure:



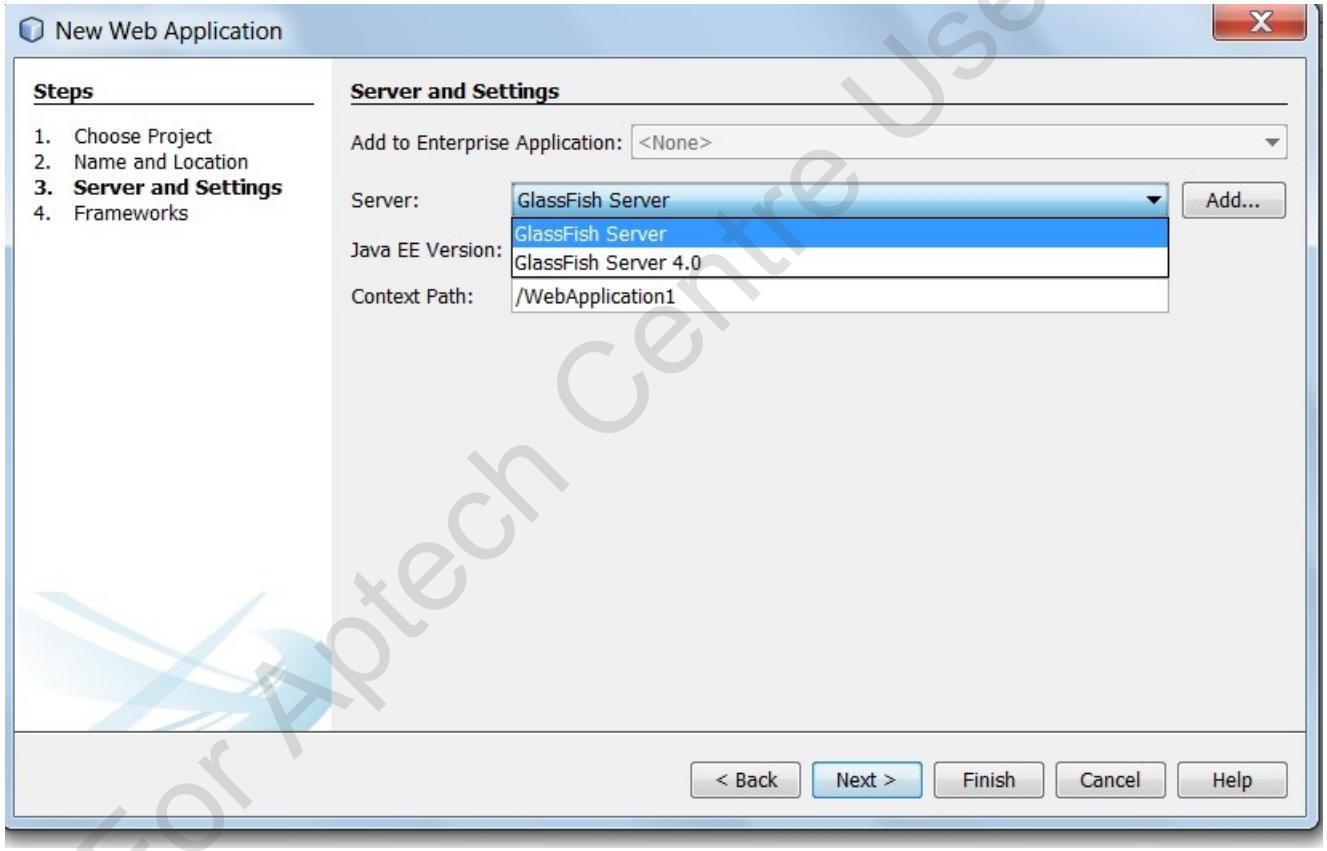
Creating a Simple Web Application 2-16

Select Java Web → Web Application and click Next. The New Web Application screen will be displayed prompting for a Web application name. Specify an appropriate Web application name and click Next. The target folder for saving the project are specified by default.



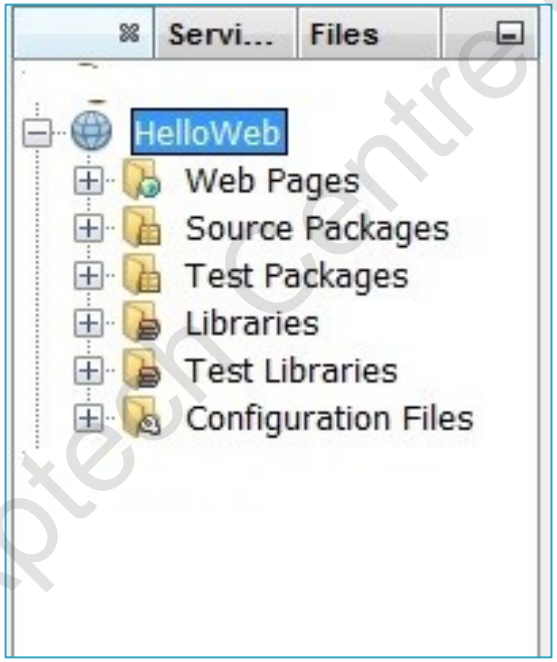
Creating a Simple Web Application 3-16

In the Server and Settings screen, select the Web server on which the application is supposed to be deployed and click Next.



Creating a Simple Web Application 4-16

After selecting the Web server, the IDE prompts for any frameworks that will be used in the application. This application does not use any frameworks hence, click Next. Click Finish. The Web application is created with folders pertaining to the project as shown in the following figure:



Creating a Simple Web Application 5-16

There are three folders created in the Web application, where the code for the application reside:

- **Source packages** – comprises of all the Java classes and beans pertaining to the application.
- **Web pages** - comprises of the JSPs, html pages, xml pages, and so on.
- **Libraries** - comprises the JDK and Glassfish server (or any other server used by the application).

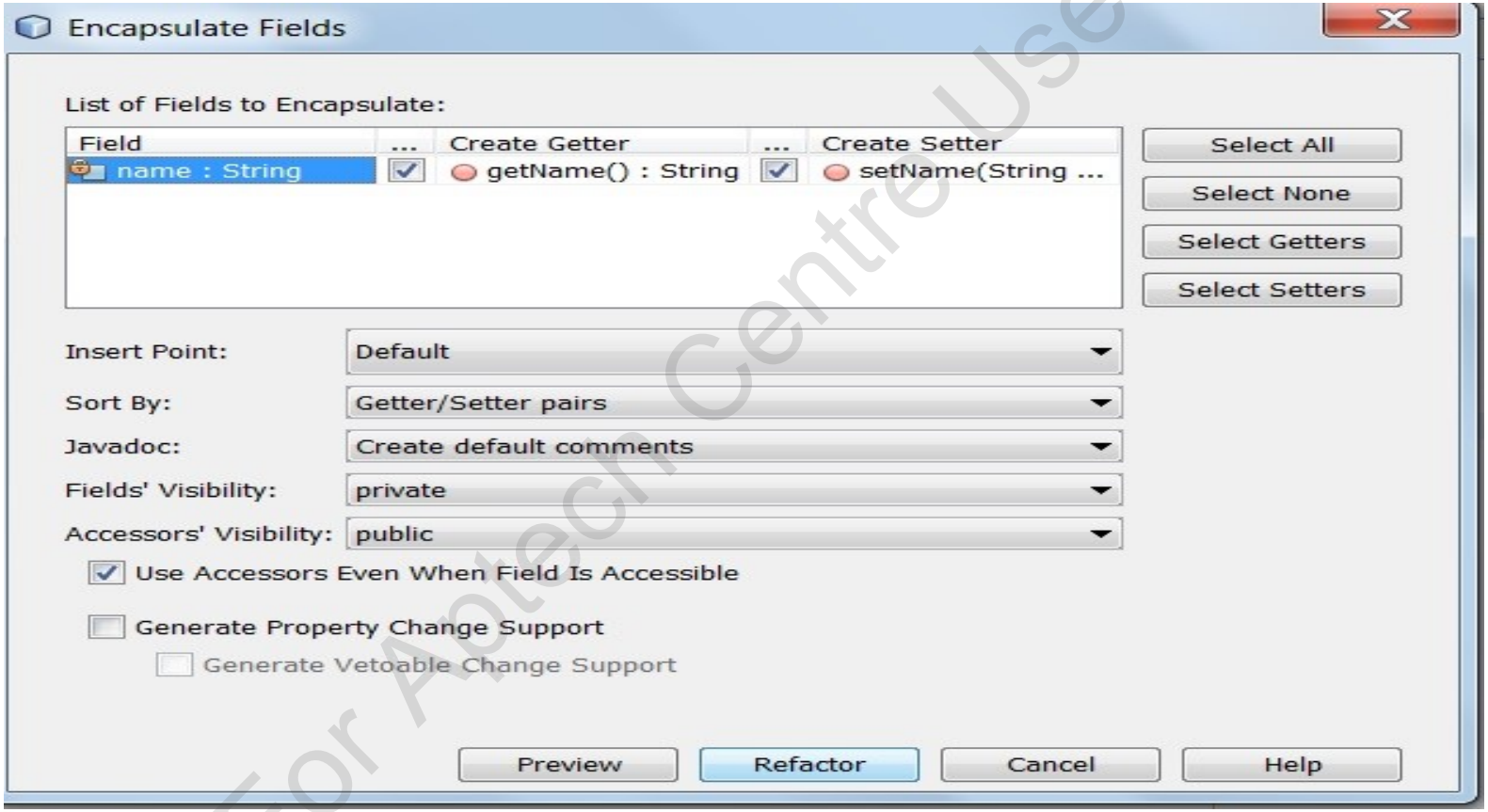
Right-click Source Packages and add a new package named, hello by selecting New Package from the menu. Add a new Java class to the package. Right-click the hello package and select New → JavaClass. Name the class as TextInput.java.

Add a string variable to the class as follows:

```
String name;
```

Creating a Simple Web Application 6-16

Right-click the variable name and select Refactor → Encapsulate Fields from the menu. This selection will open a window as shown in figure:



Creating a Simple Web Application 7-16

Select the required checkboxes and click Refactor. The selected getter and setter methods will be created.

Following code appears in the bean for the application:

```
public class TextInput {
    private String name;
    public TextInput(){
        name = null;
    }
    public String getName() {
        return username;
    }
    public void setName(String name) {
        this.name = name;
    }
}
```

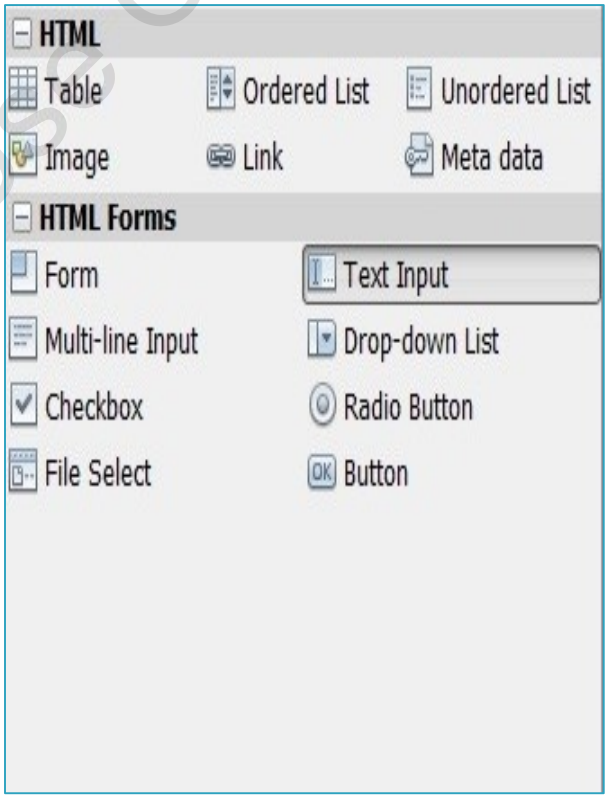
Creating a Simple Web Application 8-16

Once the bean for the application is created, the user interface is created of the application through Web pages.

The first Web page which is loaded when a Web application is loaded, is **index.jsp**.

Edit the index.jsp file which is present in the Web pages folder.

Add GUI components to the Web page. NetBeans IDE provides a drag and drop mechanism for creating the user interface using a Palette as shown in figure.

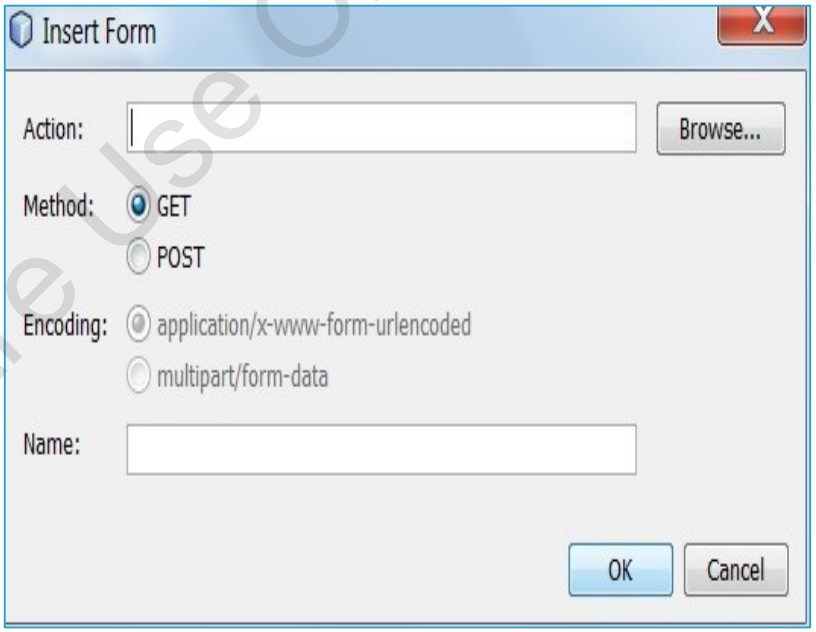


Creating a Simple Web Application 9-16

Drag the Form component from the HTML Forms section of the Palette onto the body section of the html code, which will result in the dialog box shown in figure.

In the dialog box, populate the fields as follows:

- **Action** -Specify the name, response.jsp. This refers to the JavaServer page that will act in response to the actions taken in the Web page.
- **Method** - Should be 'GET' in this case.
- **Name** - Should refer to the name of the form, here it is named 'Input Name'.



Creating a Simple Web Application 10-16

Add components to the Web page by dragging and placing the UI components in the Web page. For the given example, add a text box and a button control. Following code displays inside the index.jsp file:

```
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
        <title>JSP Page</title>
    </head>
 <body>
     <form name="Input Name Form"
action="http://localhost:8080/HelloWeb/response.jsp">
         Enter your name:
            <input type="text" name="name" value="" />
            <input type="submit" value="ok" />
        </form>
    </body>
</html>
```

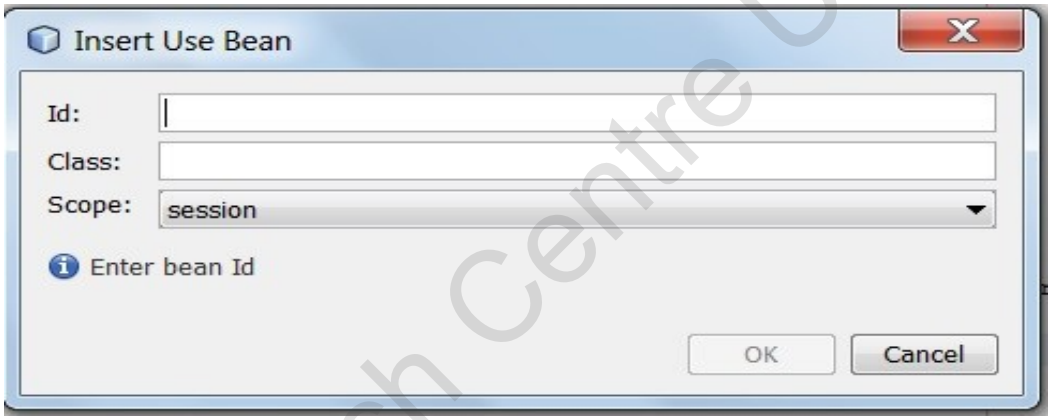
Creating a Simple Web Application 11-16

Once the index.jsp is modified, define the response.jsp file which acts as a response to the action taken in the index page. Right-click the Web Pages folder and create a new JSP file by selecting New → JSP from the menu and name it response.jsp. Following code is automatically generated by the IDE on creating the JSP file:

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

Creating a Simple Web Application 12-16

Drag and drop the Use Bean component from the JSP section of the Palette onto the body section of the html code. This component specifies the bean to be used by the Web page. The Insert Use Bean dialog box is displayed as shown in figure:



Specify a name for the Use Bean in the Id field. Here, the name mybean is given to the bean. In the Class field, specify the location of the bean as hello.TextInput.

Select the Scope of the bean as session.

Creating a Simple Web Application 13-16

With the Use Bean component, now the bean which is to be used by the application is specified. The variables of the bean are set next, for this the Set Bean Property is used. Drag and drop Set Bean Property from the JSP section of the Palette and place it after the Use Bean statement.

Use Get Bean Property to retrieve the value of the variable.

Creating a Simple Web Application 14-16

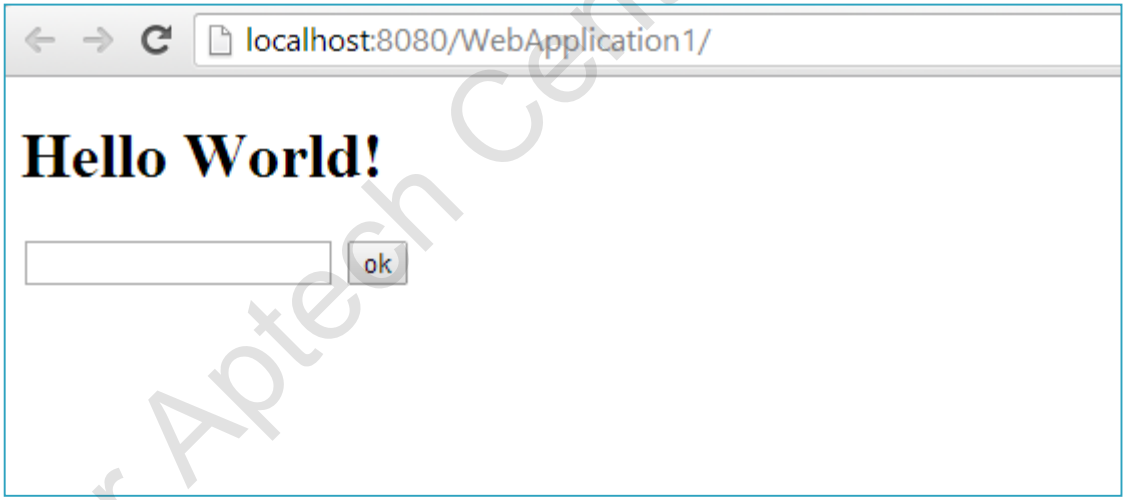
Following code defines the usage of the UseBean component in the response page:

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>JSP Page</title>
    </head>
    <body>
        <jsp:useBean id="mybean" scope="session" class="hello.TextInput" />
        <jsp:setProperty name="mybean" property="name" value="<%=
request.getParameter("name")%>" />
        <h1>Hello, <jsp:getProperty name="mybean" property="name" /> />
!</h1>
    </body>
</html>
```

Creating a Simple Web Application 15-16

Right-click the project and select Clean and Build. This will generate a .war file in the dist folder of the application. The Web application can be run after deployment. To run the application, click the Run icon on the standard toolbar.

The final output of the application is a Web page as shown in the figure. Enter the text Good Morning in the text box and click ok.



Creating a Simple Web Application 16-16

Specify the username as John in the textbox and click ok. The result is the response.jsp page with the username displayed as shown in the figure.

Summary

- Web applications comprise components such as Servlets, JSPs, static HTML pages, classes, and resources such as databases working in collaboration with each other to implement certain functionality of a particular domain.
- HTTP protocol is used by the Web application components to communicate over the Internet. It has a predefined set of messages used for communication over the Internet.
- Deployment descriptor is an XML file which comprises annotations. It is used for application deployment on the Web server.
- Helper classes are also Java classes created for independent functionalities while developing an application.
- JavaServer Pages enable the developers to create dynamic, interactive Web pages.
- Servlets are server-side programs which extend the capabilities of the server to support processing of data.
- JavaServer Faces is a component based UI development technology which makes use of view templates written in XML. These view templates are known as Facelets.
- Web services are developed based on two standards – Simple Object Access Protocol (SOAP) and Representable State Transfer (REST).