

LENGUAJE DE PROGRAMACION II

Docente: Ing. Díaz Leva Teodoro

Tema: Servlet API

Introducción

Qué es un Servlet de Java

Es una tecnología que nos permite crear aplicaciones web interactivas (dinámicas), es decir, le permite al usuario interactuar con la aplicación (hacer consultas, insertar, eliminar y actualizar datos, ...).

Un Servlet es un objeto java que pertenece a una clase que extiende de `javax.servlet.http.HttpServlet`. Son pequeños programas escritos en Java que admiten peticiones a través del protocolo HTTP. Los servlets reciben peticiones desde un navegador web, las procesan y devuelven una respuesta al navegador, normalmente en HTML. Para realizar estas tareas podrán utilizar las clases incluidas en el lenguaje Java. Estos programas son los intermediarios entre el cliente (casi siempre navegador web) y los datos (Base de Datos).

Qué es un contenedor de Servlets

Un contenedor de Servlet es un programa capaz de recibir peticiones de páginas web y redireccionar estas peticiones a un objeto Servlet.

Ejemplo: Apache Tomcat

Cómo funciona un contenedor de Servlets

1. El navegador (cliente) pide una página al servidor HTTP que es un contenedor de Servlets.
2. El servlet procesa los argumentos de la petición, es decir, el contenedor de Servlets delega la petición a un Servlet en particular elegido de entre los Servlets que contiene.
3. El Servlet, que es un objeto java, se encarga de generar el texto de la página web que se entrega al contenedor.
4. El contenedor devuelve la página web al navegador (cliente) que la solicitó, normalmente en HTML.

Por lo tanto nos encontramos en una arquitectura Cliente/Servidor. Lo normal para esto es utilizar Apache Tomcat como contenedor de servlets. Recordar que apache es un servidor HTTP.

Qué es Apache Tomcat

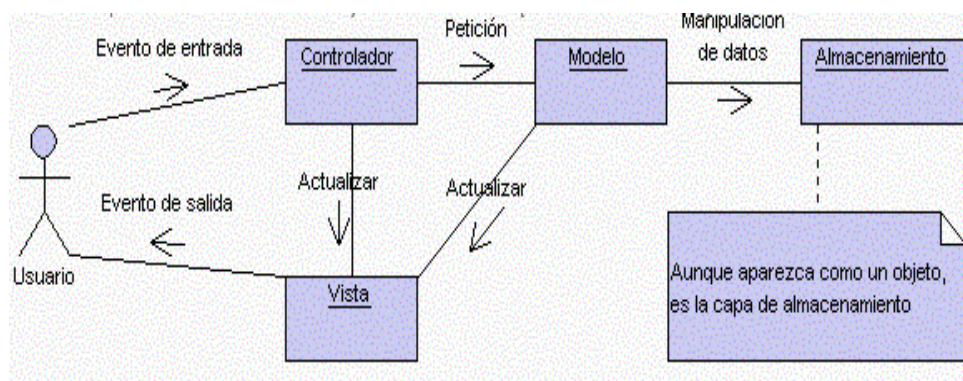
Es un contenedor de servlets, no es un servidor de aplicaciones. La diferencia principal radica en que un contenedor de servlets está pensado únicamente para tecnología web (acceso vía HTTP), mientras que en un servidor de aplicaciones se pueden hacer sistemas más complejos y multicapa. Por ejemplo, un sistema que poseerá soluciones de escritorio, web y móvil deberá alojarse en un servidor de aplicaciones, ya que posee capas distintas que requerirán mayor complejidad en el lado servidor.

Veamos algunas definiciones interesantes que para entenderlo.

- ☉ **Desplegar una aplicación (deployment):** significa ponerla en producción.
- ☉ **Servidor de aplicaciones:** un servidor de aplicaciones consiste en un contenedor que abarca la lógica de negocio de un sistema (según el patrón MVC, sería el Modelo), y que provee de respuestas a las peticiones de distintos dispositivos que tienen acceso a ella. Son un claro ejemplo del modelo **cliente/servidor**, cuyo lado cliente ejecuta requerimientos de procesamiento al otro lado, donde el servidor se encarga de procesar y responder.
- ☉ **Servidores de aplicaciones J2EE:**
 - **JBoss:** libre, es el más utilizado.
 - **WebSphere:** conjunto de aplicaciones desarrolladas por IBM. La más importante es WebSphere Application Server. También se incluyen aplicaciones para diseñar modelos de negocio, ejecutar y monitorizar procesos.
 - **GlassFish:** libre, desarrollado por Sun. Incorpora un componente llamado Grizzly, que aumenta la escalabilidad y velocidad del servidor.
 - **WebLogic**
- ☉ **Servidores de aplicaciones .NET:**
 - Internet Information Server (IIS)
 - Base4 Server
 - Zope

Qué es MVC

El Modelo Vista Controlador (MVC) es un patrón de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos (Modelo, Vista y Controlador). El Patrón MVC se ve frecuentemente en aplicaciones Web, donde la Vista es la página HTML y el código que provee de datos dinámicos a la página; el Modelo es el Sistema de Gestión de Base de Datos y la Lógica de negocio; el Controlador es el responsable de recibir los eventos de entrada desde la Vista.



Un modelo puede tener diversas vistas, cada una con su correspondiente controlador. Un ejemplo clásico es el de la información de una base de datos, que se puede presentar de diversas formas: diagrama de tarta, de barras, tabular, etc. Veamos cada componente.

● **Modelo**

Es la representación específica de la información con la cual el sistema opera. La lógica de datos asegura la integridad de estos y permite derivar nuevos datos; por ejemplo, no permitiendo comprar un número de unidades negativo, o calculando los totales e impuestos del carrito de compra. Esto quiere decir que aquí se operan los datos y las reglas de negocio asociadas al sistema, incluyendo el análisis sintáctico y el procesamiento de los datos de entrada y de los datos de salida.

● **Vista**

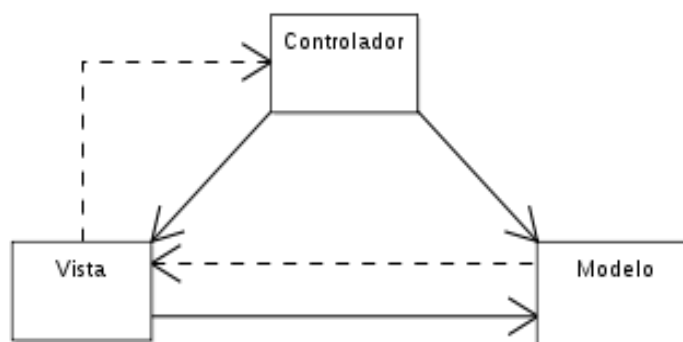
Este presenta el Modelo, usualmente la interfaz de usuario. La vista es la capa de la aplicación que ve el usuario en un formato adecuado para interactuar, en otras palabras, es nuestra interfase gráfica.

● **Controlador**

El Controlador es la capa que controla todo lo que puede realizar nuestra aplicación. Responde a eventos, usualmente acciones del usuario e invoca cambios en el modelo y probablemente en la vista. Está compuesto por acciones que se representan con funciones en una clase. Por ejemplo, yo tengo mi controlador llamado “Clientes”, y este controlador puede realizar las acciones “crear”, “editar”, “listar” entre otras.

Ejemplo:

Bien, pero esto ¿cómo se implementa? Existe una pequeña dificultad: la mayor parte de las herramientas de desarrollo incorporan en las clases de la vista gran parte o todo el procesamiento de eventos. Con lo que el controlador queda semioculto dentro de la vista. A pesar de ello, podemos acercarnos bastante al patrón.



Modelo

Un ejemplo de la vida real de un Modelo sería una clase llamada Cliente, la cual tiene las mismas propiedades de una tabla cliente en mi base de datos.

Controlador

Un Controlador sería el Controlador Cliente, generalmente las clases Controladoras llevan el sufijo “Controlador”, así que en nuestro caso se llamaría **ClientesControlador** que vendría ser un **servlet**.

El controlador llevaría las acciones que nosotros podemos realizar en un cliente como por ejemplo, agregar, borrar, modificar, agregar orden, etc.

Vista

La Vista es el más fácil de entender, simplemente es nuestra página HTML. A través de la acción del Controlador especificamos a que vista queremos enviar el resultado de la acción del Controlador. En algunos casos es necesario pasar información a la Vista desde el Controlador, esto se logra fácilmente en el código de la acción.

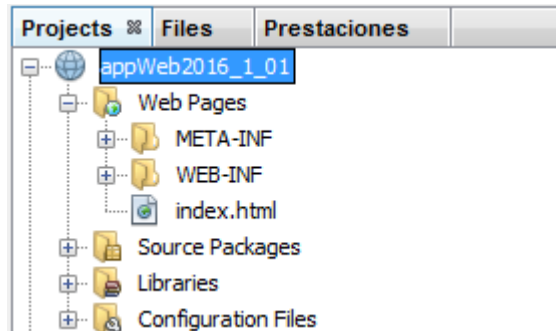
En **aplicaciones JavaWeb**, brevemente podemos decir que el patrón MVC separa la lógica (y acceso a datos) de una aplicación de su presentación, usando 3 componentes:

1. **Modelo:** Representa las reglas de negocio de la aplicación (y el acceso a datos subyacente).
2. **Vistas:** Es la presentación de la aplicación, generalmente son archivos con extensión .JSP.
3. **Controlador:** Actúan de intermediario entre el usuario y el Modelo y las Vistas. Recogen las peticiones del usuario, interaccionan con el modelo y deciden que vista es la que debe mostrar los datos. Son los **servlets** que cumplirán esta tarea de intermediario.

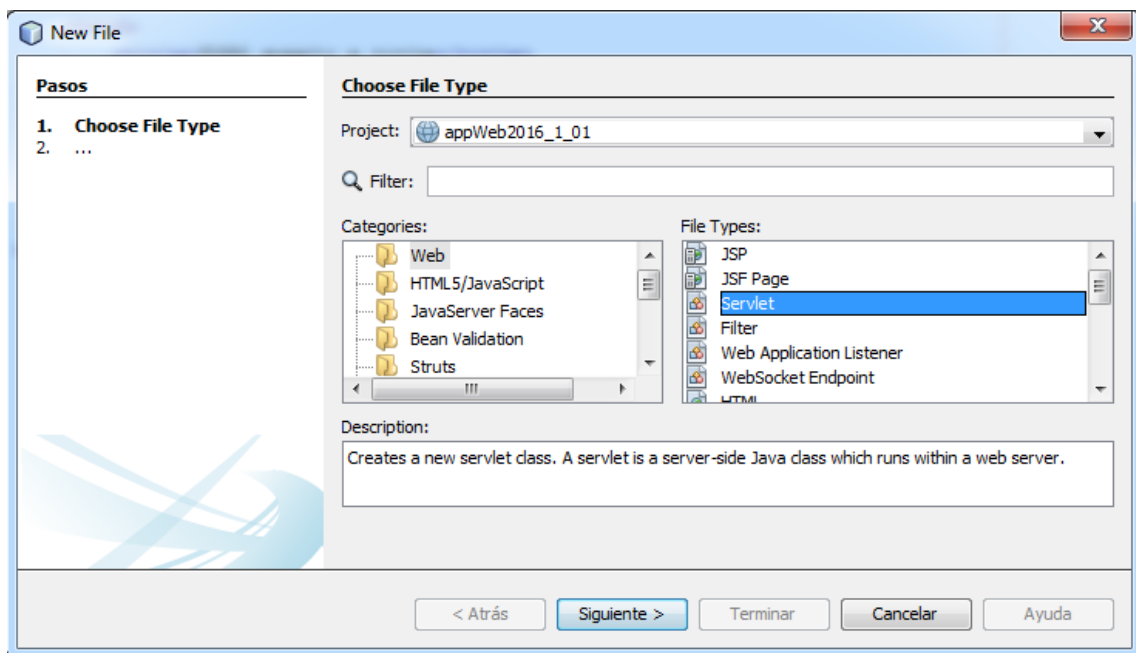
Laboratorio 1:

Objetivo: crear un servlet (controlador) “AlumnoServlet” para recibir datos de un alumno ingresados a través de un formulario “FormAlumno.jsp” y los redirija hacia la vista alumnos.jsp para visualizarlos.

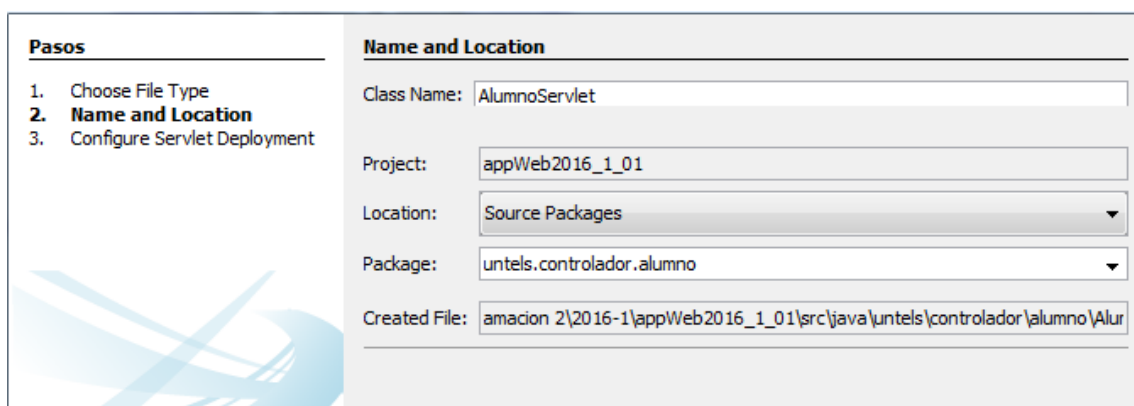
1. Cree proyecto web “appWeb2016_1_01”, podría tener otro nombre.

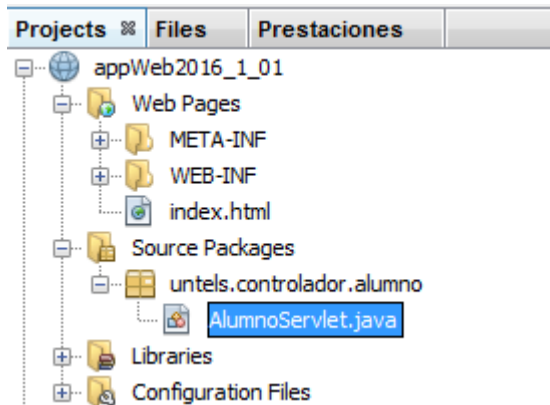


2. Cree Servlet con el nombre “AlumnoServlet”, en un paquete “untels.controlador.alumno”: clic derecho sobre el proyecto>New, si en caso no esté la opción Servlet, clic en opción Other...y seleccionar Servlet.



Clic en Next y asignar nombre y paquete:





El código por defecto, que luego cambiaríamos para cumplir objetivo de nuestra aplicación:

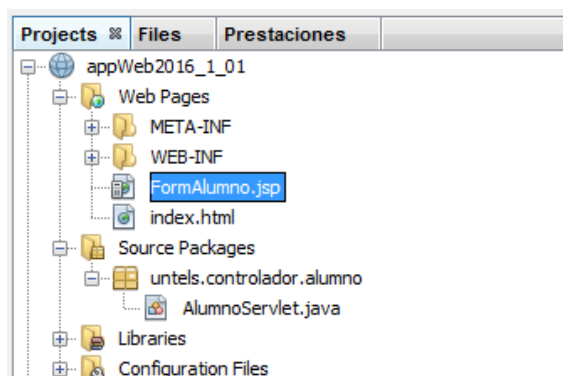
```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**...4 lines */
@WebServlet(name = "AlumnoServlet", urlPatterns = {"/AlumnoServlet"})
public class AlumnoServlet extends HttpServlet {

    /** Processes requests for both HTTP <code>GET</code> and <code>POST</code> ...9 lines */
    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        try (PrintWriter out = response.getWriter()) {
            /* TODO output your page here. You may use following sample code. */
            out.println("<!DOCTYPE html>");
            out.println("<html>");
            out.println("<head>");
            out.println("<title>Servlet AlumnoServlet</title>");
            out.println("</head>");
            out.println("<body>");
            out.println("<h1>Servlet AlumnoServlet at " + request.getContextPath() + "</h1>");
            out.println("</body>");
            out.println("</html>");
        }
    }

    HttpServlet methods. Click on the + sign on the left to edit the code.
}
```

3. Crear formulario “formAlumno”, es un archivo JSP:



Agregar etiquetas HTML en FormAlumno:

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
```

```
<title>Servlet</title>
```

```
</head>
```

```
<body>
```

```
<fieldset>
```

```
<legend>Ingrese sus datos</legend>
```

```
<form action="AlumnoServlet" method="post">
```

```
<table>
```

```
<tr>
```

```
<td>Nombres</td>
```

```
<td><input name="txtnom" type="text"></td>
```

```
</tr>
```

```
<tr>
```

```
<td>Apellidos</td>
```

```
<td><input name="txtape" type="text"></td>
```

```
</tr>
```

```
<tr>
```

```
<td>Edad</td>
```

```
<td><input name="txtedad" type="text"></td>
```

```
</tr>
```

```
<tr>
```

```
<td><input value="Enviar" type="submit"></td>
```

```
<td></td>
```

```
</tr>
```

```
</table>
```

```
</form>
```

```
</fieldset>
```

```
</body>
```

```
</html>
```

En la propiedad action del formulario va el nombre del servlet **"AlumnoServlet"**

Servlet

localhost:8681/AppWeb2015_2_01/FormAlumno.jsp

Aplicaciones Nueva pestaña RTIN

Ingrese sus datos

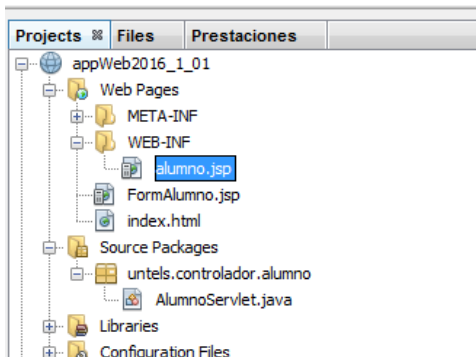
Nombres

Apellidos

Edad

Enviar

4. Crear archivo Alumno.jsp, este archivo lo crearemos dentro de la carpeta WEB-INF, el cual no se podrá ejecutar directamente por el usuario.



5. Modificamos el código del **AlumnoServlet**:

```
@WebServlet(name = "AlumnoServlet", urlPatterns = {"/AlumnoServlet"})
public class AlumnoServlet extends HttpServlet {
    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        //obtenemos los datos ingresados a traves de la vista "formulario alumno"
        String nombre = request.getParameter("txtnom");
        String apellidos = request.getParameter("txtape");
        String edad = request.getParameter("txtedad");

        //creamos un arreglo de objetos tipo cadena para almacenar los datos del alumno
        ArrayList<String> arralum = new ArrayList<String>();
        arralum.add(nombre);
        arralum.add(apellidos);
        arralum.add(edad);

        //asignamos el arreglo alumnos a un atributo alumnos que podría tener otro nombre
        request.setAttribute("alumnos", arralum);

        //redirigir hacia la página alumnos.jsp
        request.getRequestDispatcher("WEB-INF/alumno.jsp").forward(request, response);
    }
}
```

request : set y get Attribute

Dentro de request, los attribute son lo que nosotros como programadores podemos usar a gusto. Podemos guardar y recibir cualquier Object java que queramos sin más que ponerles un nombre. Sin embargo, el request desaparece en cuanto terminamos de procesar la página/Servlet actual, así que la única forma de pasar estos attributes a la siguiente es haciendo un forward() desde nuestro request. En nuestro ejemplo tenemos:

```
request.setAttribute("alumnos", arralum);
request.getRequestDispatcher("WEB-INF/alumno.jsp").forward(request, response);
```


6. En la vista **alumno.jsp**, obtenemos el arreglo e imprimimos datos del alumno:

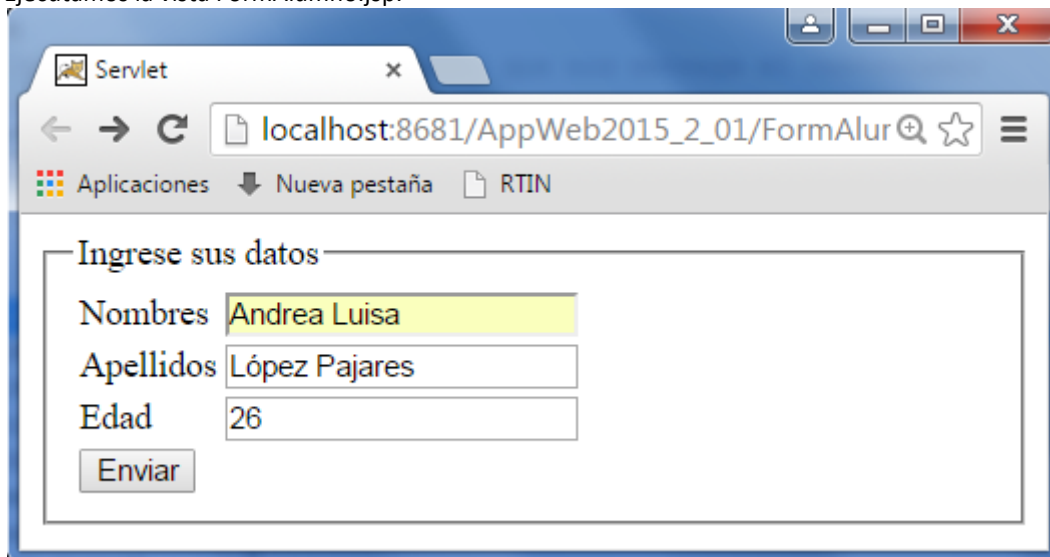
```
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
  <title>JSP Page</title>
</head>
<body>
  <%
    //obtenemos los datos del alumno que nos entrega el controlador alumnoServlet
    ArrayList<String> alumnos = (ArrayList<String>) request.getAttribute("alumnos");

    //imprimir datos del arreglo
  %>
  Mi nombre es <%=alumnos.get(0)%> <%=alumnos.get(1)%> y tengo <%=alumnos.get(2)%> años.

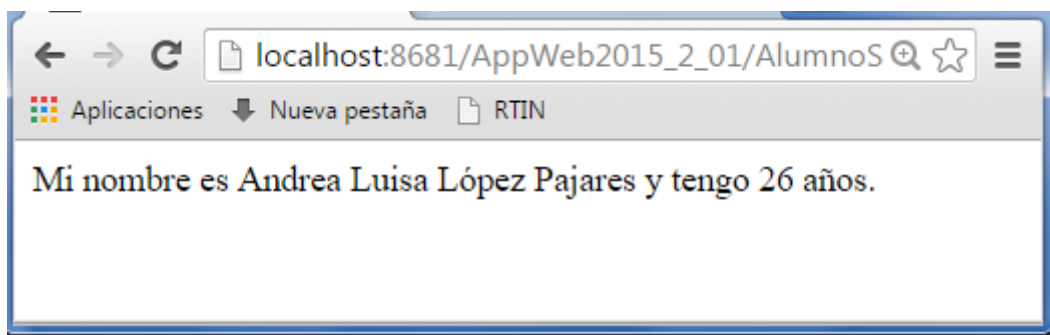
</body>
</html>
```

//*****Ejecución de la aplicación:

Ejecutamos la vista FormAlumno.jsp:

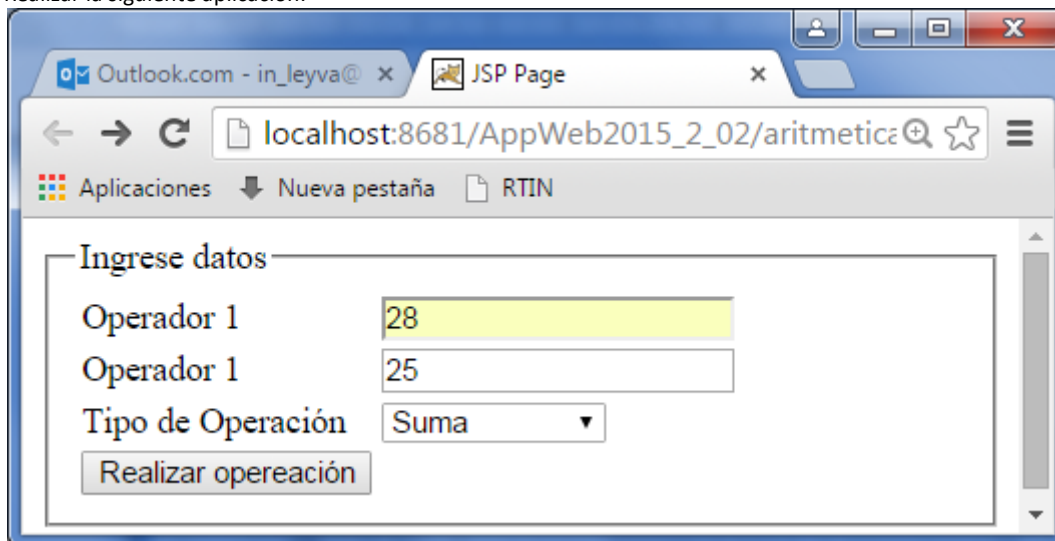


Presionamos sobre botón enviar. Recuerde que estos datos se trasladan al controlador AlumnoServlet, el cual lo almacena en un arreglo "arralum" y estos datos son recogidos en la vista **alumno.jsp**:



Ejercicio:

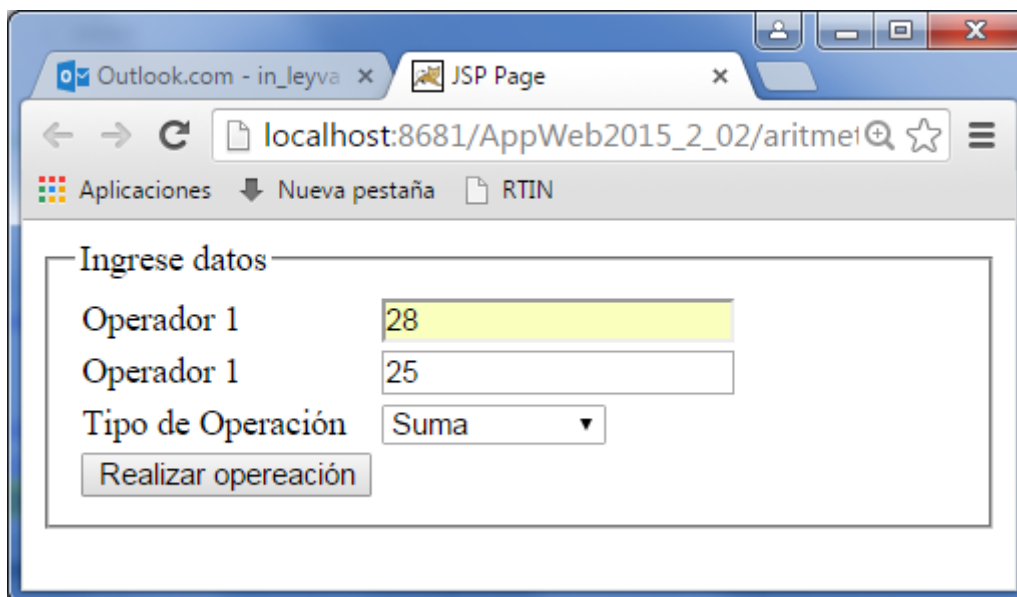
Realizar la siguiente aplicación:



The screenshot shows a web browser window with two tabs: 'Outlook.com - in_leyva@' and 'JSP Page'. The address bar displays 'localhost:8681/AppWeb2015_2_02/aritmetica'. The page content is titled 'Ingrese datos' and contains a form with the following elements:

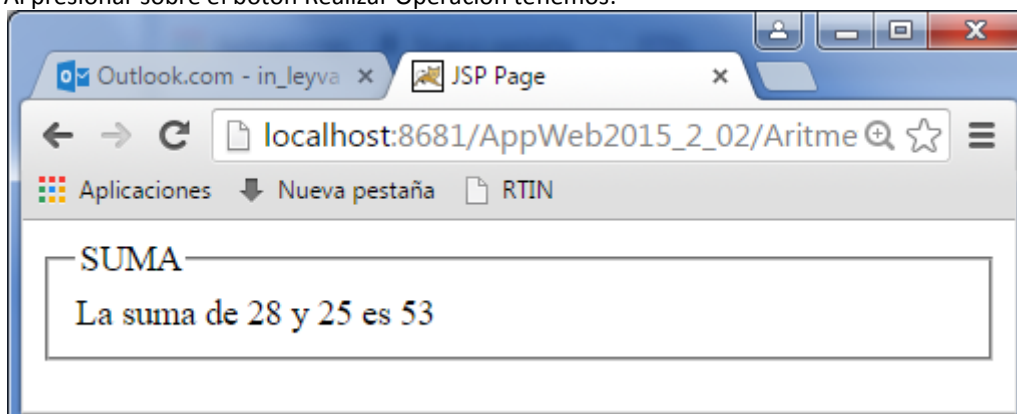
- Two input fields for numbers, both labeled 'Operador 1'. The first field contains the value '28' and is highlighted in yellow.
- A second input field containing the value '25'.
- A dropdown menu labeled 'Tipo de Operación' with 'Suma' selected.
- A button labeled 'Realizar operación'.

La aplicación debe procesar el ingreso de datos y redirigir hacia una página para mostrar el operador 1, operador 2 ingresado; además el resultado según tipo de operación aritmética elegida, por ejemplo con la operación suma:



This screenshot is identical to the one above, showing the 'Ingrese datos' form with the values 28 and 25, and the 'Suma' operation selected.

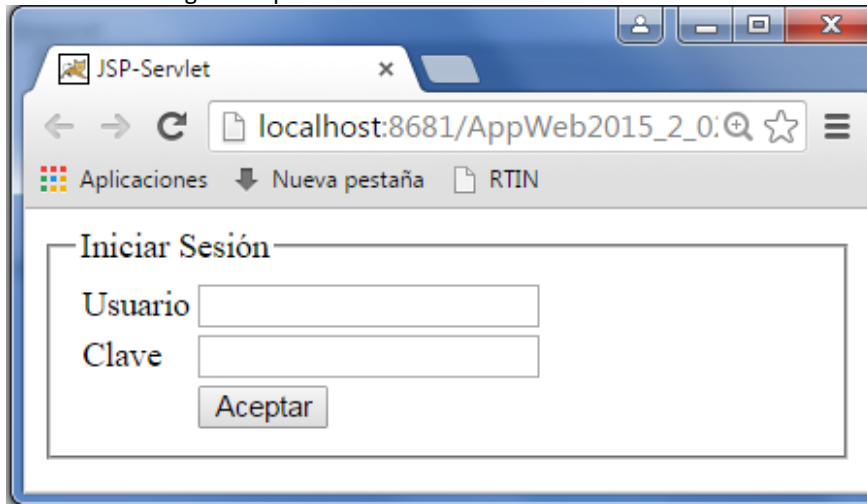
Al presionar sobre el botón Realizar Operación tenemos:



The screenshot shows the same web browser window, but the page content has changed. It now displays the title 'SUMA' and a text box containing the result: 'La suma de 28 y 25 es 53'.

Laboratorio 2:

Utilizando la base de datos “tienda”, construir un login que cuando intentemos acceder a la aplicación nos muestre la siguiente pantalla de inicio de sesión:

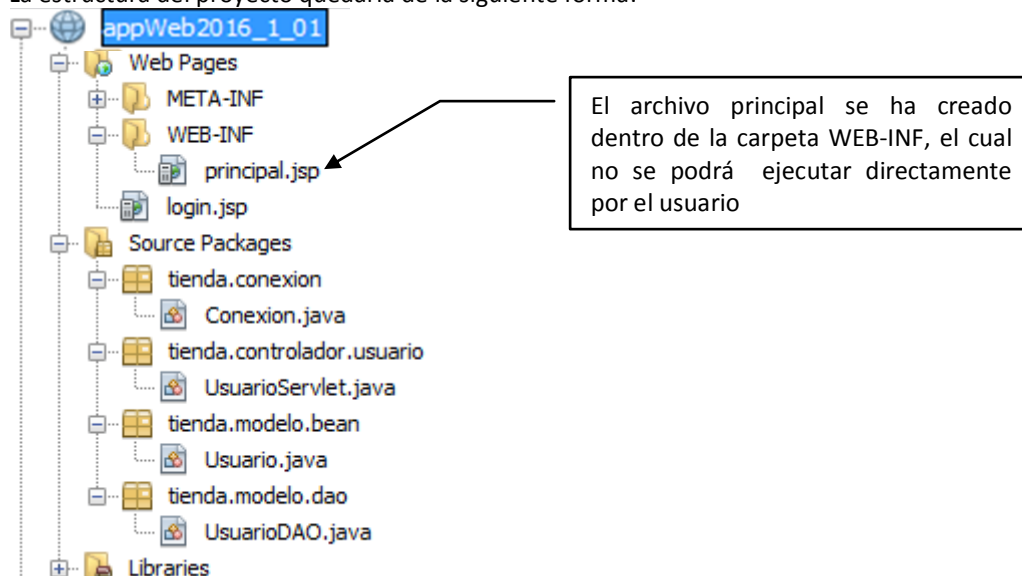


La petición se realizará al controlador (Servlet: UsuarioServlet), este recibe el usuario y clave los cuales verificará en el modelo (en la base de datos). Si el usuario existe el controlador redirigirá hacia una página principal, de lo contrario redirigirá a la misma página “login.jsp” mostrando un mensaje Usuario y/clave incorrectos.

Para desarrollo del ejercicio se requiere crear los siguientes archivos:

- **Login.jsp:** para ingresar usuario y clave.
- **Principal.jsp:** página de inicio del sistema si el usuario se autenticado correctamente (usuario existe en la base de datos).
- **Clase Conexión.java:** encargada de realizar la conexión a la base de datos “tienda”.
- **Clase UsuarioDAO.java:** encargada de verificar si el usuario está en la base de datos, y demás operaciones sobre la base de datos tienda.
- **Clase Usuario.java:** encargada de encapsular los datos de la tabla usuario, es decir representa a la tabla usuario de la base de datos tienda.
- **Servlet UsuarioServlet.java:** es el controlador que recibe la petición de la vista(login.jsp) y interactúa con el modelo y decide que vista es la que debe mostrar los datos(principal.jsp o login.jsp).

La estructura del proyecto quedaría de la siguiente forma:



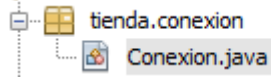
Desarrollo:

1. Empecemos diseñando el archivo login.jsp:

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
    <title>JSP-Servlet</title>
  </head>
  <body>
    <fieldset>
      <legend>Iniciar Sesión</legend>
      <form action="UsuarioServlet" method="post">
        <table>
          <tr>
            <td>
              Usuario
            </td>
            <td>
              <input type="text" name="txtusuario">
            </td>
          </tr>
          <tr>
            <td>
              Clave
            </td>
            <td>
              <input type="password" name="txtclave">
            </td>
          </tr>
          <tr>
            <td>
            </td>
            <td>
              <input type="submit" value="Aceptar">
            </td>
          </tr>
        </table>
      </form>
    </fieldset>
  </body>
</html>
```

The image shows a visual representation of the login form. It features a title 'Iniciar Sesión' at the top. Below the title, there are two input fields: one labeled 'Usuario' and another labeled 'Clave'. At the bottom of the form, there is a button labeled 'Aceptar'.

2. Conexión a la base de datos:



```
package tienda.conexion;

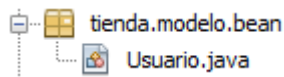
import java.sql.Connection;
import java.sql.DriverManager;

public class Conexion {

    static String url = "jdbc:mysql://localhost/tienda";
    static String usuario = "root";
    static String Password = "123";

    public static Connection abrir() {
        try {
            Class.forName("com.mysql.jdbc.Driver");
            Connection cn = (Connection)
DriverManager.getConnection(url, usuario, Password);
            return cn;
        } catch (Exception ex) {
            return null;
        }
    }
}
```

3. Clase Usuario:



```
public class Usuario {
    //propiedades

    private int idusuario;
    private String usuario;
    private String clave;
    private String estado;
    private int idempleado;
    //metodos setts y getts

    public int getIdusuario() {
        return idusuario;
    }

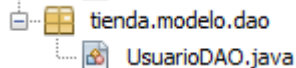
    public void setIdusuario(int idusuario) {
        this.idusuario = idusuario;
    }

    public String getUsuario() {
        return usuario;
    }

    public void setUsuario(String usuario) {
        this.usuario = usuario;
    }
}
```

```
public String getClave() {  
    return clave;  
}  
  
public void setClave(String clave) {  
    this.clave = clave;  
}  
  
public String getEstado() {  
    return estado;  
}  
  
public void setEstado(String estado) {  
    this.estado = estado;  
}  
  
public int getIdempleado() {  
    return idempleado;  
}  
  
public void setIdempleado(int idempleado) {  
    this.idempleado = idempleado;  
}  
}
```

4. Clase UsuarioDAO:



```
package tienda.modelo.dao;  
  
import java.sql.Connection;  
import java.sql.PreparedStatement;  
import java.sql.ResultSet;  
import java.sql.SQLException;  
import tienda.conexion.Conexion;  
import tienda.modelo.bean.Usuario;  
  
public class UsuarioDAO {  
    public static Usuario validarUsuario(String us, String pas) {  
        //variables  
        Usuario usuario = null;  
        PreparedStatement stm;  
        ResultSet rs;  
  
        //instruccion sql para verificar si usuario esta registrado en la base  
        //de datos  
        String sql = "SELECT*FROM USUARIO WHERE USUARIO=? AND  
PASSWORD=?";  
        //abrir conexion a la base de datos tienda.  
        Connection cn = Conexion.abrir();  
        try {  
            //asociar instruccion sql al objeto PreparedStatement a  
            //través de la aconexion  
            stm = cn.prepareStatement(sql);  
            //asignar valores a los parametros ? ?  
            stm.setString(1, us);  
            stm.setString(2, pas);  
            rs = stm.executeQuery();  

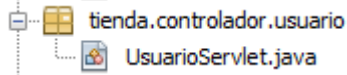
```

```
//if porque dentro del rs solo tengo un registro
if (rs.next()) {
    //crear objeto usuario
    usuario = new Usuario();
    //encapsular datos en el objeto usuario
    usuario.setIdusuario(rs.getInt(1));
    usuario.setUsuario(rs.getString(2));
    usuario.setClave(rs.getString(3));
    usuario.setEstado(rs.getString(4));
    usuario.setIdempleado(rs.getInt(5));

}
//cerrar objetos
cn.close();
stm.close();
rs.close();

} catch (SQLException ex) {
    usuario = null;
}
//rerornar objeto usaurio con sus respectivos datos
return usuario;
}
}
```

5. UsuarioServlet:



```
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import tienda.modelo.bean.Usuario;
import tienda.modelo.dao.UsuarioDAO;

@WebServlet(name = "UsuarioServlet", urlPatterns = {"/UsuarioServlet",
"/CerrarSesion"})
public class UsuarioServlet extends HttpServlet {
    protected void processRequest(HttpServletRequest request,
HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");

        //capturar path
        String path = request.getServletPath();

        //si path es UsuarioServlet llamar al método autenticar
        if (path.equals("/UsuarioServlet")) {
            autenticar(request, response);
        }

        //si path es cerrarSesion llamar al método cerrarSesion
        if (path.equals("/CerrarSesion")) {
            cerrarSesion(request, response);
        }
    }
}
```

```
//-----método autenticar-----
protected void autenticar(HttpServletRequest request,
    HttpServletResponse response)
    throws ServletException, IOException {

    //obtener valores de usuario clave del formulario
    String user = request.getParameter("txtusuario");
    String clav = request.getParameter("txtclave");

    //llamar al método validarUsuario, retorna un objeto usuario
    Usuario usuario = UsuarioDAO.validarUsuario(user, clav);

    //verificamos si usuario existe
    if (usuario != null) {
        //crear variable de sesión us y asignar el objeto usuario
        request.getSession().setAttribute("us", usuario);
        //redireccionamiento hacia el archivo principal.jsp
        request.getRequestDispatcher("/WEB-INF/principal.jsp").forward(request, response);
    } else {
        //crear variable de sesión msg y asignar mensaje
        request.getSession().setAttribute("msg", "Usuario y/o
clave incorrecto...!");
        //redireccionamiento a la página login.jsp
        request.getRequestDispatcher("login.jsp").forward(request,
response);
    }
}

//-----método cerrar sesion-----
protected void cerrarSesion(HttpServletRequest request,
    HttpServletResponse response)
    throws ServletException, IOException {
    //cierra la sesion
    request.getSession().invalidate();
    response.sendRedirect("login.jsp");
}
}
```

Aclaración

Una **sesión** es un mecanismo de programación de las tecnologías de web que permite conservar información sobre un usuario al pasar de una página a otra. A diferencia de una cookie, los datos asociados a una sesión se almacenan en el servidor y nunca en el cliente.

6. Página principal.jsp:

```
<%@page import="tienda.modelo.bean.Usuario"%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
        <title>::Perú Deportes</title>
    </head>
    <body>
        <h1>PerúDeportes</h1>
        <%
            //recibir objeto usuario del controlador
            Usuario usuario=(Usuario)request.getSession().getAttribute("us");
            %>
        <!-- imprimir usuario-->
        Usuario :<%=usuario.getUsuario()%>
        <!-- cerrar sesion-->
        <a href="CerrarSesion">Cerrar Sesión</a>
    </body>
</html>
```

7. Página login.jsp: en este archivo codificamos la recepción e impresión del mensaje de “usuario y/o clave incorrectos”:

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
        <title>JSP-Servlet</title>
    </head>
    <body>
        <fieldset>
            <legend>Iniciar Sesión</legend>
            <form action="UsuarioServlet" method="post">
                <table>
                    <tr>
                        <td>
                            Usuario
                        </td>
                        <td>
                            <input type="text" name="txtusuario">
                        </td>
                    </tr>
                    <tr>
                        <td>
                            Clave
                        </td>
                        <td>
                            <input type="password" name="txtclave">
                        </td>
                    </tr>
                </table>
            </form>
        </fieldset>
    </body>
</html>
```

```
<tr>
    <td>
    </td>
    <td>
        <input type="submit" value="Aceptar">
    </td>
</tr>

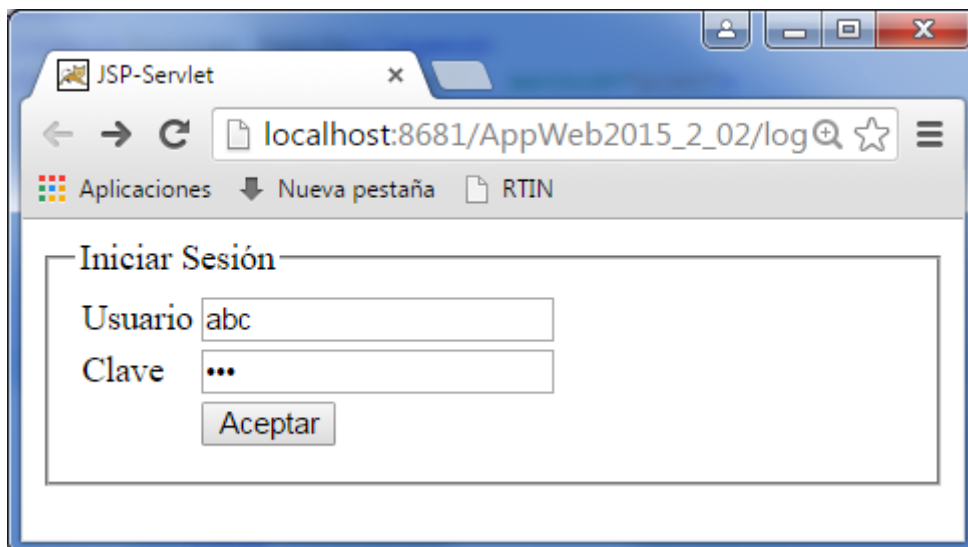
<tr>
    <td colspan="2">
        <!--recibe el mensaje del controlador-->
<%
    String msg=(String) request.getSession().getAttribute("msg");%>
    <!--si el mensaje es diferente de null,imprimir mensaje
    .De lo contrario imprimir vacio-->
    <span><%=msg!=null? msg:""%> </span>
    </td>
</tr>

</table>
</form>
</fieldset>
</body>
</html>
```

Se ha creado una fila debajo del botón aceptar y se ha combinado a una sola celda para recepcionar e imprimir mensaje "usuario y/o clave incorrectos".

*****EJECUCION DE LA APLICACIÓN*****

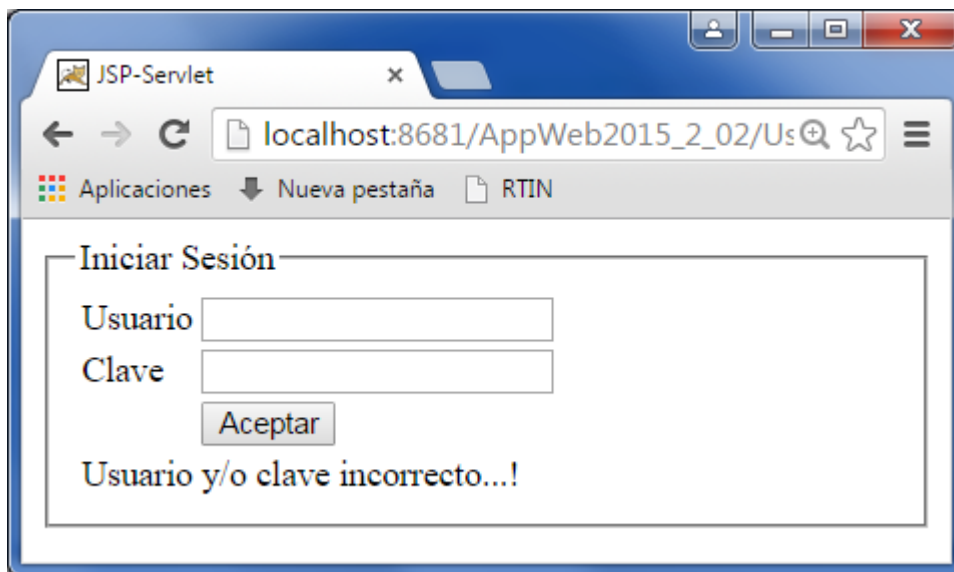
Se asume que está registrado en la base de datos tienda un usuario: **abc** con password: **123**



Nos redirige hacia la página principal.jsp:



En caso que usuario sea incorrecto:



Conclusión:

Los servlets son Clases en Java que se ejecutan en el Servidor Tomcat. Los servlets atenderán a los requerimientos HTTP. En la presente sesión, se detalló la estructura de los servlets, donde colocarlos para desplegarlos y, finalmente, cómo ejecutarlos.

Ejercicio:

Programa sobre la aplicación para que en la página principal.jsp se visualice el nombre del empleado correspondiente al usuario ingresado. Recuerde que la tabla usuario está relacionada con la tabla empleado.