JAVA WEB SERVICES

Realizado por: Diana Alfaro
Complementado por: Carlos Sánchez

CONTENIDO

- ¿Qué es un Servicio Web?
- Historia
- ¿Qué es XML, SOAP, WSDL, UDDI?
- XML - Extensible Markup Language
- Soap - XML-RPC (Xml Remote Procedure Call
- WSDL - Web Services Description Language
- UDDI - Universal Discovery Description and Integration
- Ventajas de un Web Services
- Desventajas de un Web Services
- ¿Por qué crear un WS?
- Razones para crear servicios web
- Java y los servicios web
- *Integración de SERVLETS y JSP*
- Modelo de funcionamiento
- Instalación del "ECLIPSE IDE FOR JAVA EE DEVELOPERS" y el servidor "APACHE TOMCAT"
- SERVLET
- Ventajas fundamentales
- Concepto de aplicación web
- Ciclo de vida de un SERVLET
- SERVLETS HTTP
- JAVA JSP (java server page)
- Características
- Ventajas
- ¿Por qué JSP?
- Sesiones y Cookies
- Archivo WAR

¿QUÉ ES UN SERVICIO WEB?

Un Servicio Web es una tecnología que utiliza un conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones. Distintas aplicaciones de software desarrolladas en lenguajes de programación diferentes, y ejecutadas sobre cualquier plataforma, pueden utilizar los servicios web para intercambiar datos en redes de ordenadores como Internet. La interoperabilidad se consigue mediante la adopción de estándares abiertos.

HISTORIA

Los Servicios Web surgieron ante una necesidad de estandarizar la comunicación entre distintas plataformas (PC, Mainframe, Mac, etc.) y lenguajes de programación (PHP, C, Java, etc.). Anteriormente se habían realizado intentos de crear estándares pero fracasaron o no tuvieron el suficiente éxito, algunos de ellos son DCOM y CORBA, por ser dependientes de la implementación del vendedor DCOM – Microsoft, y CORBA – ORB (a pesar que CORBA de múltiples vendedores pueden operar entre sí, hay ciertas limitaciones para aplicaciones de niveles más altos en los cuales se necesite seguridad o administración de transacciones).

Otro gran problema es que se hacía uso de RPC (Remote Procedure Call) para realizar la comunicación entre diferentes nodos. Esto, además de presentar ciertos problemas de seguridad, tiene la desventaja de que su implementación en un ambiente como es Internet, es casi imposible (muchos Firewalls bloquean este tipo de mensajes, lo que hace prácticamente imposible a dos computadoras conectadas por Internet comunicarse). Los Servicios Web surgieron para finalmente poder lograr la tan esperada comunicación entre diferentes plataformas. En la actualidad muchos sistemas legacy están pasando a ser servicios web. Es por esto que en 1999 se comenzó a plantear un nuevo estándar, el cual terminaría utilizando XML, SOAP, WSDL, y UDDI.

¿QUÉ ES XML, SOAP, WSDL, UDDI?

Son estándares empleados en un servicio web

XML - EXTENSIBLE MARKUP LANGUAGE

Es el formato estándar para los datos que se vayan a intercambiar.

SOAP - Simple Object Access Protocol o XML-RPC (XML Remote PROCEDURE CALL

Protocolos sobre los que se establece el intercambio.

WSDL - WEB SERVICES DESCRIPTION LANGUAGE

Es el lenguaje de la interfaz pública para los servicios Web. Es una descripción basada en XML de los requisitos funcionales necesarios para establecer una comunicación con los servicios Web.

UDDI - UNIVERSAL DISCOVERY DESCRIPTION AND INTEGRATION

UDDI son las siglas del catálogo de negocios de Internet. El registro en el catálogo se hace en XML.

VENTAJAS DE UN WEB SERVICES

- Aportan interoperabilidad entre aplicaciones de software independientemente de sus propiedades o de las plataformas sobre las que se instalen.
- Los servicios Web fomentan los estándares y protocolos basados en texto, que hacen más fácil acceder a su contenido y entender su funcionamiento.
- Permiten que servicios y software de diferentes compañías ubicadas en diferentes lugares geográficos puedan ser combinados fácilmente para proveer servicios integrados.

DESVENTAJAS DE UN WEB SERVICES

- Para realizar transacciones no pueden compararse en su grado de desarrollo con los estándares abiertos de computación distribuidacomo CORBA (Common Object Request Broker Architecture).
- Su rendimiento es bajo si se compara con otros modelos de computación distribuida, tales como RMI (Remote Method Invocation), CORBA o DCOM (Distributed Component Object Model). Es uno de los inconvenientes derivados de adoptar un formato basado en texto. Y es que entre los objetivos de XML no se encuentra la concisión ni la eficacia de procesamiento.
- Al apoyarse en HTTP, pueden esquivar medidas de seguridad basadas en firewall cuyas reglas tratan de bloquear o auditar la comunicación entre programas a ambos lados de la barrera

¿POR QUÉ CREAR UN WS?

- Para realizar transacciones no pueden compararse en su grado de desarrollo con los estándares abiertos de computación distribuida como CORBA (Common Object Request Broker Architecture).
- Su rendimiento es bajo si se compara con otros modelos de computación distribuida, tales como RMI (Remote Method Invocation), CORBA o DCOM (Distributed Component Object Model). Es uno de los inconvenientes derivados de adoptar un formato basado en texto. Y es que entre los objetivos de XML no se encuentra la concisión ni la eficacia de procesamiento.
- Al apoyarse en HTTP, pueden esquivar medidas de seguridad basadas en firewall cuyas reglas tratan de bloquear o auditar la comunicación entre programas a ambos lados de la barrera.

RAZONES PARA CREAR SERVICIOS WEB

La principal razón para usar servicios Web es que se pueden utilizar con HTTP sobre TCP (Transmission Control Protocol) en el puerto 80. Dado que las organizaciones protegen sus redes mediante firewalls -que filtran y bloquean gran parte del tráfico de Internet-, cierran casi todos los puertos TCP salvo el 80, que es, precisamente, el que usan los navegadores. Los servicios Web utilizan este puerto, por la simple razón de que no resultan bloqueados. Es importante señalar que los servicios web se pueden utilizar sobre cualquier protocolo, sin embargo, TCP es el más común.

Otra razón es que, antes de que existiera SOAP, no había buenas interfaces para acceder a las funcionalidades de otros ordenadores en red. Las que había eran ad hoc y poco conocidas, tales como EDI (Electronic Data Interchange), RPC (Remote Procedure Call), u otras APIs.

Una tercera razón por la que los servicios Web son muy prácticos es que pueden aportar gran independencia entre la aplicación que usa el servicio Web y el propio servicio. De esta forma, los cambios a lo largo del tiempo en uno no deben afectar al otro. Esta flexibilidad será cada vez más importante, dado que la tendencia a construir grandes aplicaciones a partir de componentes distribuidos más pequeños es cada día más utilizada.

Integración de Servlets y JSP

Una aplicación Web realiza tareas de procesado y presentación:

- Los Servlets son adecuados para procesado.
- Las páginas JSP son adecuadas presentación.

Una aplicación Web puede combinar Servlets y páginas JSP:

- Procesado de parámetros de la petición: Servlets.
- Acceso a bases de datos: Servlets.
- Lógica de la aplicación: Servlets.
- Presentación (vistas): JSP.

MODELO DE FUNCIONAMIENTO

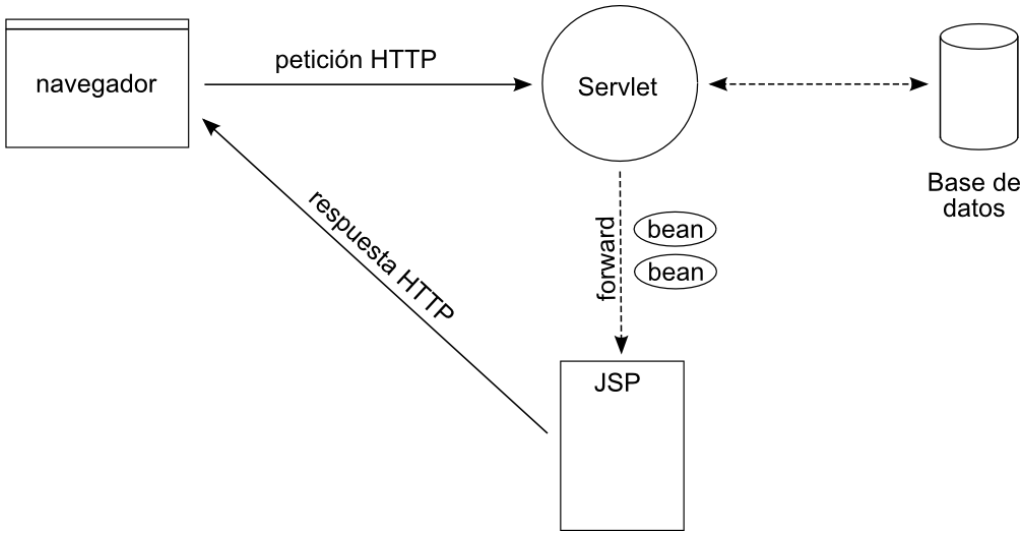
1. El cliente envía la petición HTTP a un servlet.
2. El servlet procesa la petición.

   Si es necesario, se conecta a la base de datos.

3. El servlet redirige la petición a un JSP.

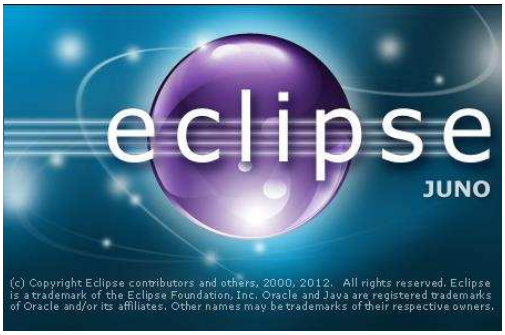
   Si es necesario, añade beans como parámetros.

4. El JSP lee los parámetros y devuelve la respuesta formateada visualmente al usuario.



INSTALACIÓN DEL "ECLIPSE IDE FOR JAVA EE DEVELOPERS"

Y EL SERVIDOR "APACHE TOMCAT"

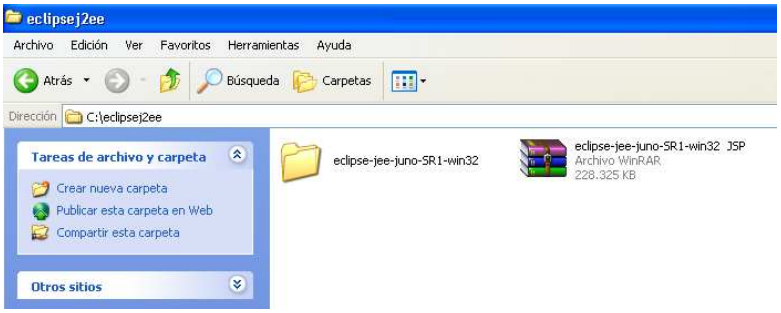


ECLIPSE IDE FOR JAVA EE DEVELOPERS

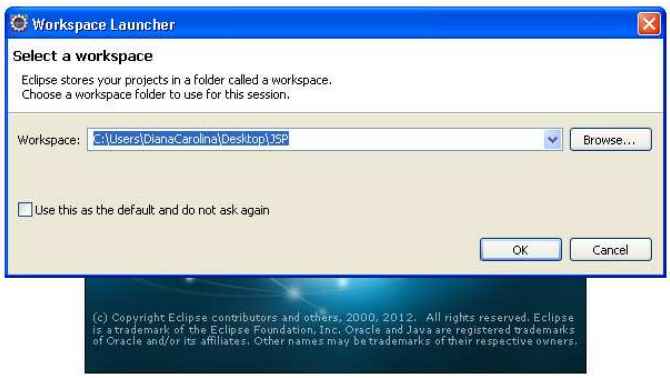
Para desarrollar aplicaciones que se ejecuten en un servidor web debemos utilizar la versión de Eclipse que viene con todos los complementos que facilitan el desarrollo.

La versión que debemos descargar es Eclipse IDE for Java EE Developers (http://www.eclipse.org/downloads/packages/release/juno/r), como podemos ver el tamaños es mayor que la versión "Eclipse IDE for Java Developers"

Crearemos la carpeta eclipsej2ee y dentro de la misma descomprimamos el entorno de Eclipse que acabamos de descargar "Eclipse IDE for Java EE Developers".



Seleccionamos una otra carpeta donde se almacenaran los proyectos que realicemos.

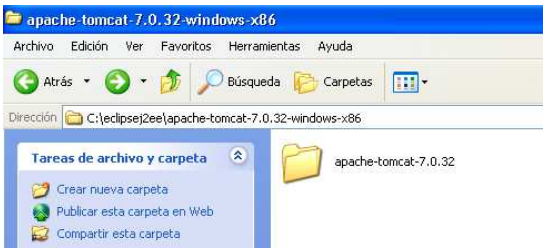


Cuando ejecutamos el Eclipse nos pide seleccionar la carpeta donde se almacenarán los proyectos que crearemos y aparece el siguiente entorno (como podemos ver prácticamente igual que la versión "Java Developers" con un título distinto):

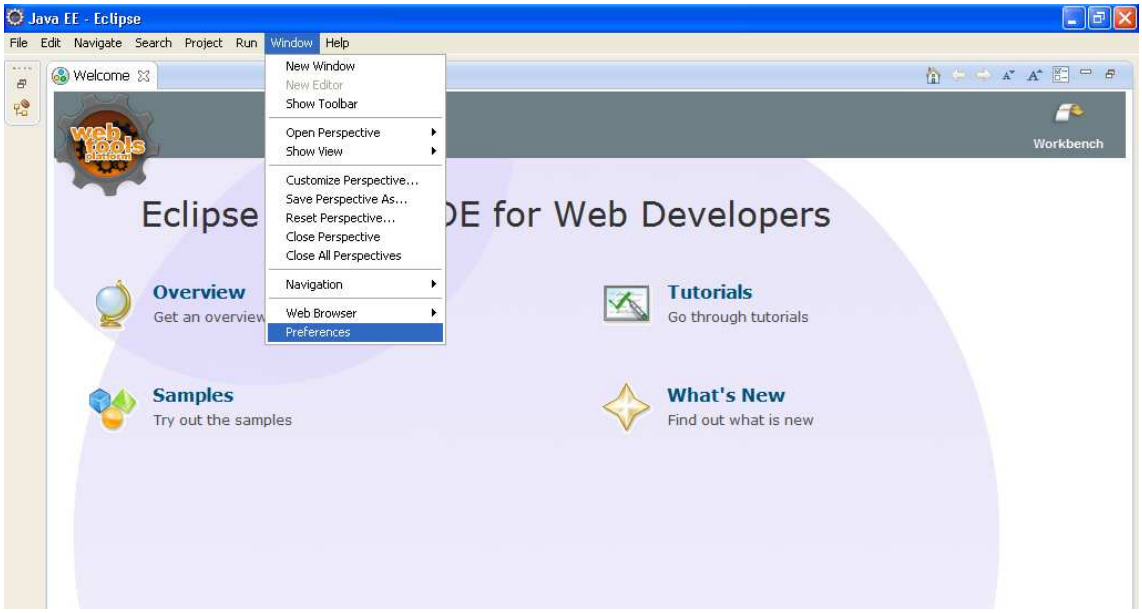
APACHE TOMCAT

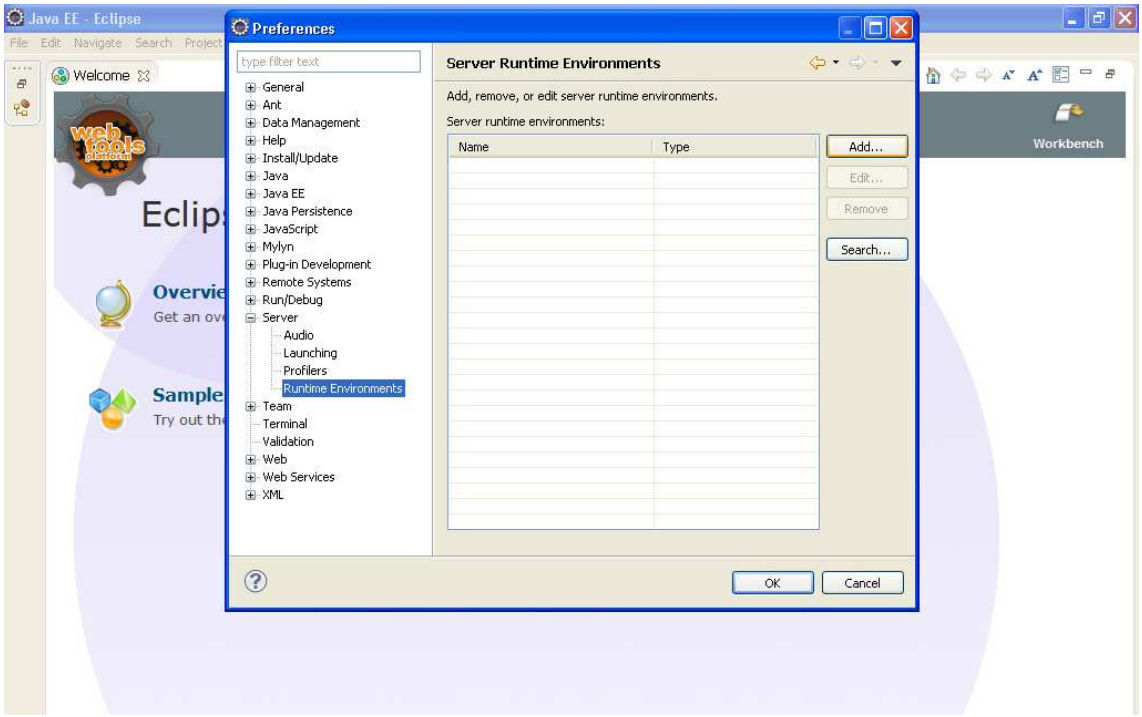
El servidor web "Apache Tomcat" nos permitirá ejecutar servlet y páginas dinámicas.

Podemos descargar el "Apache Tomcat" de http://tomcat.apache.org/download-70.cgi (descargar el archivo Binary Distributions Core 32-bit Windows zip) y descomprimirlo en una carpeta.

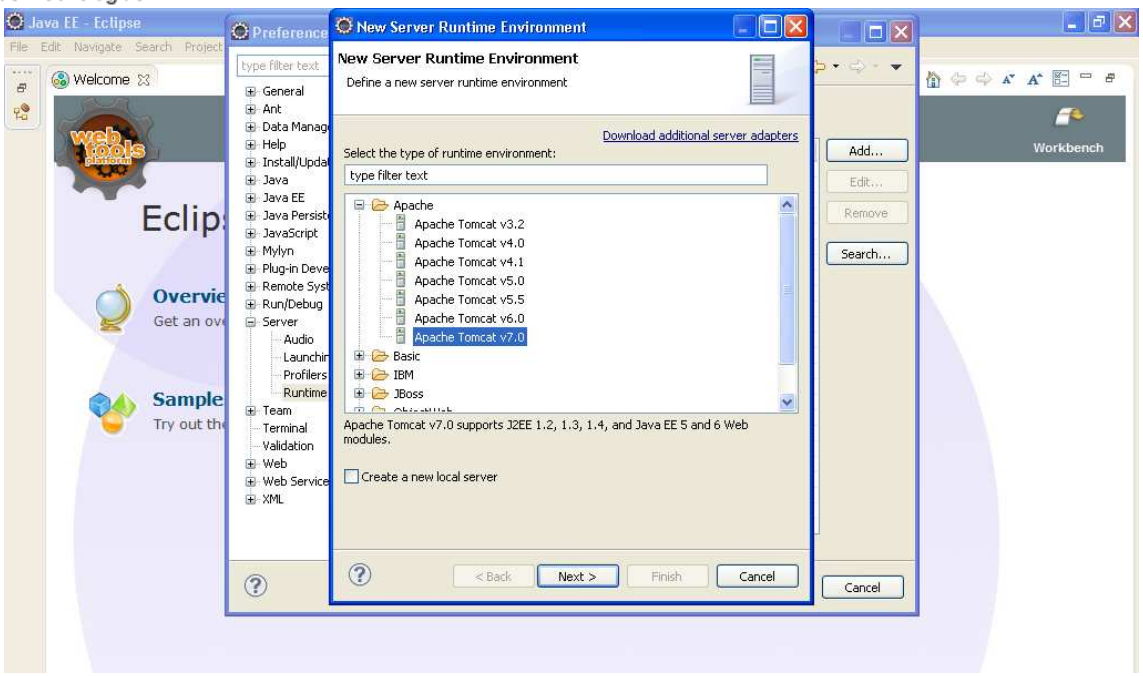


Una vez descomprimido procedemos a registrarlo en Eclipse. Desde el menú de opciones seleccionamos *Window -> Preferences* y en el diálogo que aparece debemos seleccionar Server -> *Runtimes Environments* y presionar el botón "*Add...*":

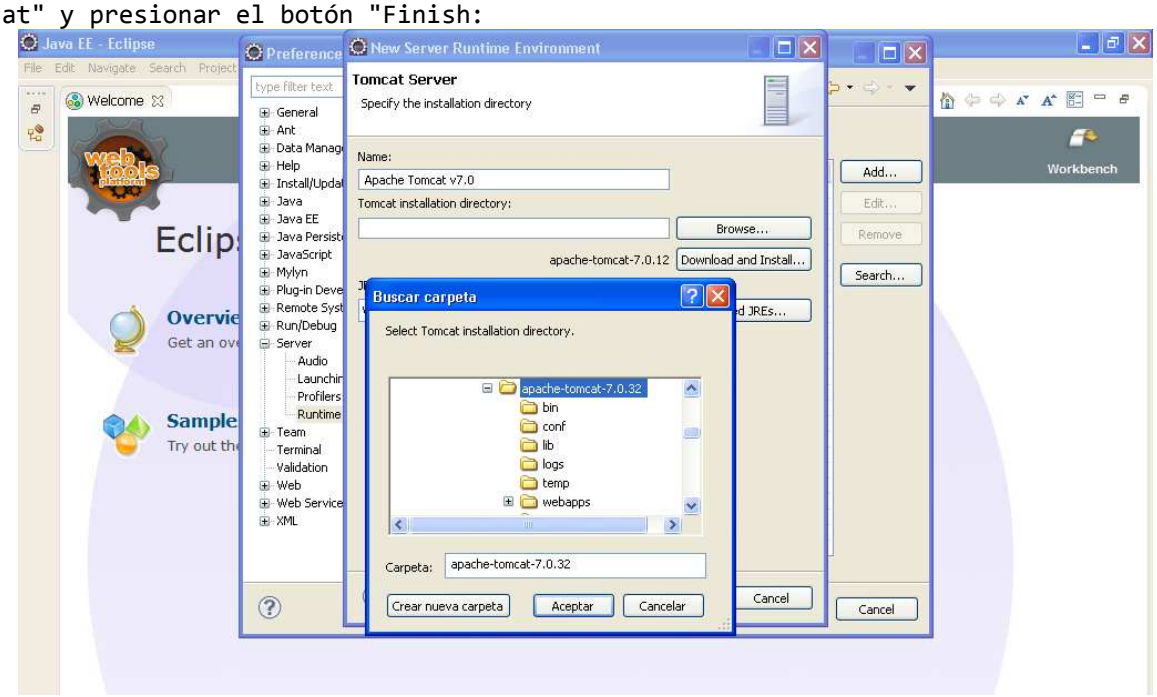




En el nuevo diálogo que aparece seleccionamos de la carpeta "Apache" la versión 7 que es la que acabamos de descargar y descomprimir en una carpeta de nuestro disco duro:

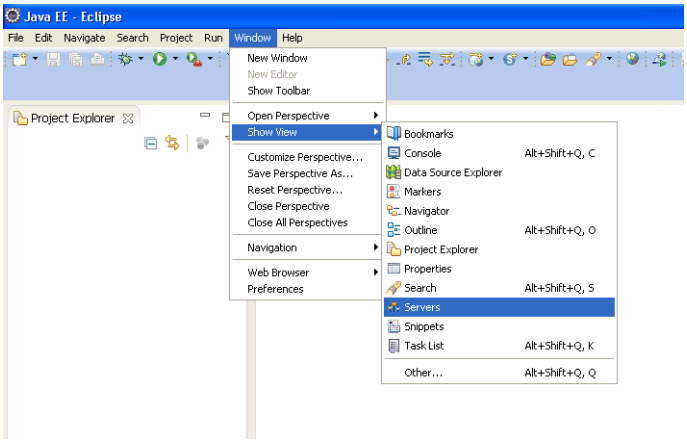


En el último diálogo que aparece debemos seleccionar la carpeta donde hemos descomprimido el "Apache Tomcat" y presionar el botón "Finish":

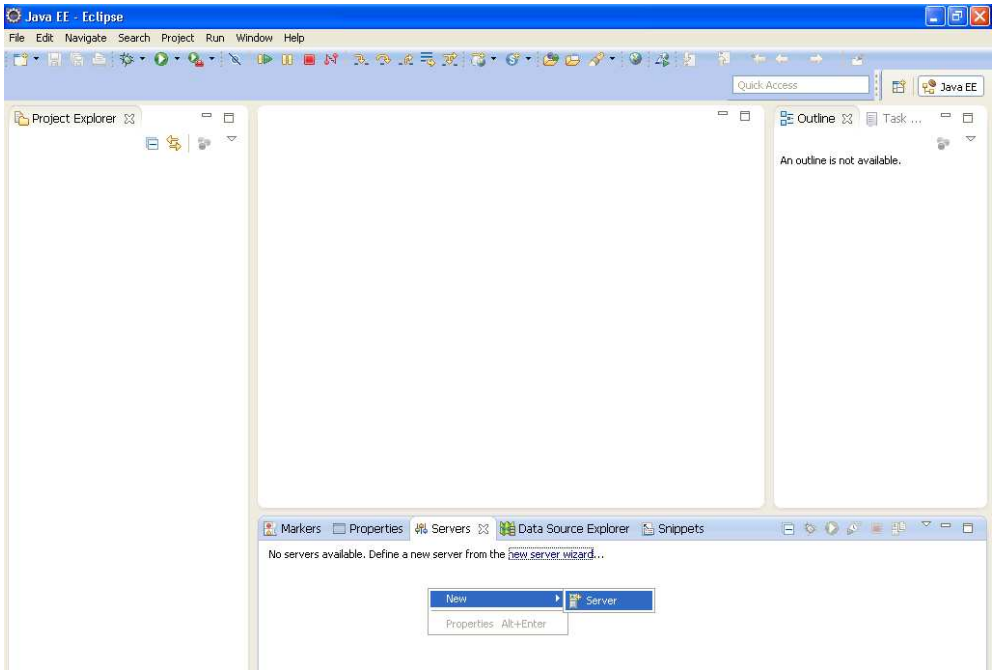


Ahora debemos iniciar los servicios del servidos "Apache Tomcat" para podes hacer aplicaciones que hagan peticiones.

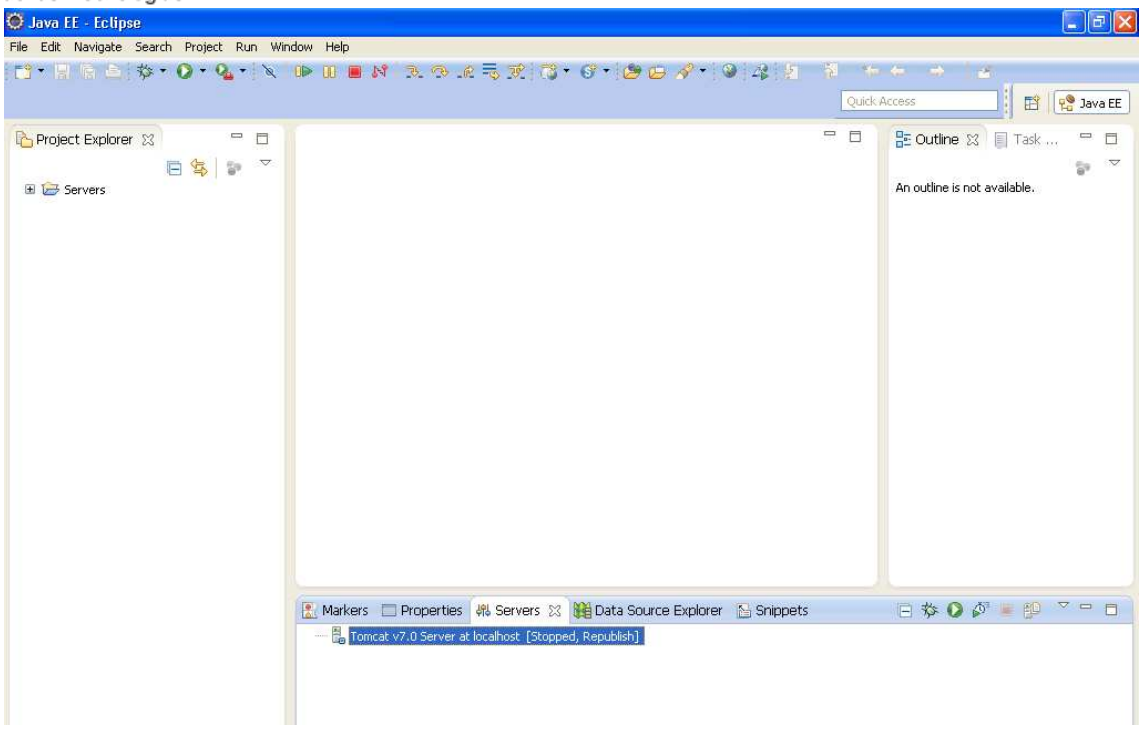
Para arrancar el Tomcat debemos presionar el botón derecho del mouse sobre la ventana "Server", si no parece esta ventana podemos activarla desde el menú (*Window -> Show View -> Servers*) y seguidamente seleccionar del menú contextual la opción *New -> Server*:



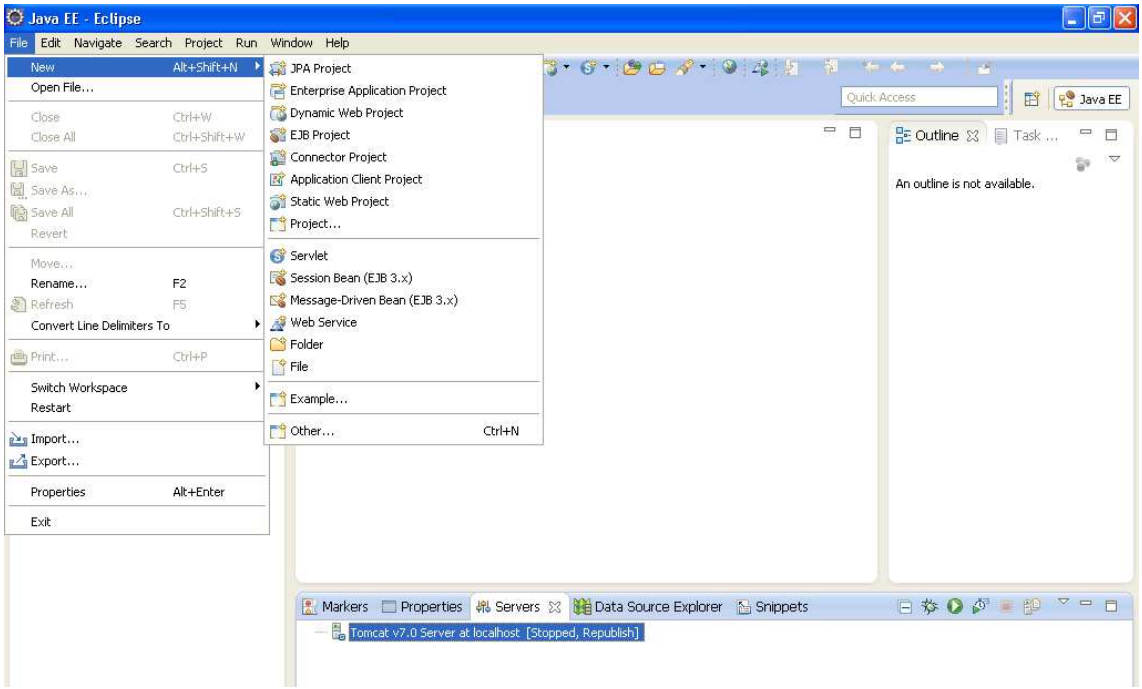
En este diálogo seleccionamos "Apache" Tomcat V7.0 y presionamos el botón "Finish":



Como podemos ver ya tenemos el "Tomcat" listo para poderlo utilizar en los distintos proyectos que implementaremos:



Si ingresamos al menú de opciones *File -> New* veremos que nos permite crear una serie de proyectos muy distintos a la otra versión de Eclipse:



SERVLET

Un servlet es una clase que se ejecuta en el contexto de un servidor web (en nuestro caso el Apache Tomcat).

Un servlet se ejecuta en un servidor web y el resultado de ejecución viaja por internet para ser visualizado en un navegador web (normalmente un servlet genera HTML, pero puede generar otros formatos de archivos).

Cada petición HTTP recibida se procesa en un hilo, e invoca un método del servlet.

VENTAJAS FUNDAMENTALES

- Portabilidad entre plataformas y servidores.
- Potencia:
  APIs de Java
  Utilización de código externo.
- Eficiencia :
  Instancia permanententemente cargada en memoria por cada servlet.
  Ejecución de peticiones mediante invocación de un método.
  Cada petición se ejecuta en un hilo.
  Mantiene automáticamente su estado y recursos externos: conexiones a bases de datos, conexiones de red, etc.
- Seguridad:
  Lenguaje java: máquina virtual, chequeo de tipos, gestión de memoria, excepciones, etc.
  Gestor de seguridad de Java.
- Elegancia:
  Código java: modular, orientado a objetos, limpio y simple.
- API servlets: potente y fácil de utilizar.
- Integración:
  Integración fuerte entre servlets y servidor: permite colaboración entre ambos.
- Extensibilidad y flexibilidad:
  API Servlet extensible.
  Filtros (cadenas de servlets).
  Integrable con JSP (Java Server Pages).
  Comunidad grande de desarrolladores.

CONCEPTO DE APLICACIÓN WEB

Conjunto de servlets, JSPs y otros recursos (ficheros HTML, imágenes, ficheros de configuración, etc.) relacionados entre sí por formar parte de la misma aplicación.

Los recursos de una aplicación Web comparten un prefijo de URL.

Una aplicación Web se puede empaquetar en un fichero WAR.

CICLO DE VIDA DE UN SERVLET

- Cuando arranca el servidor:

  Se crea una instancia.

  Se inicializa el servlet (método init())

- Cuando llega una petición:

  Se invoca el método service() sobre un nuevo hilo.

- Cuando se cierra el servidor:

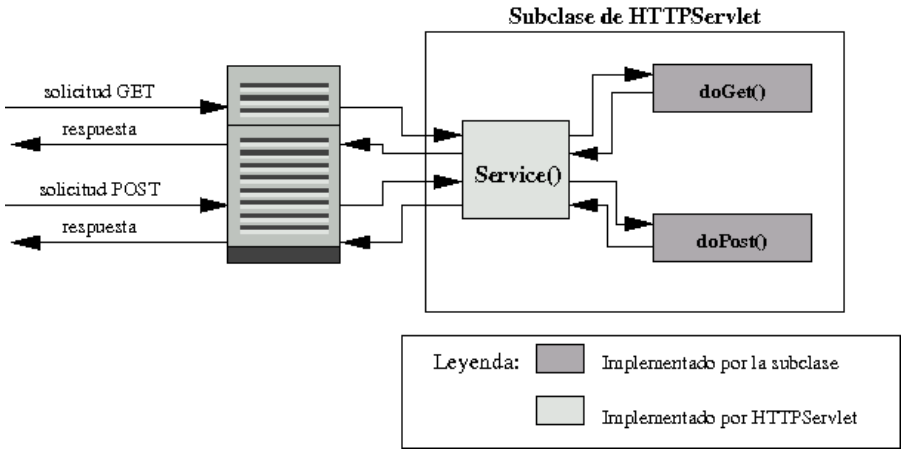
  Se invoca el método destroy() y después se destruye el servlet.

SERVLETS HTTP

Heredan de HttpServlet, que implementa el método service() para que invoque a:

- void doGet(HttpServletRequest req, HttpServletResponse resp)
- void doPost(HttpServletRequest req, HttpServletResponse resp)
- void do...(HttpServletRequest req, HttpServletResponse resp)
- getLastModified(HttpServletRequest req)

Los servlets reescriben sólo los métodos doXXX que necesiten.

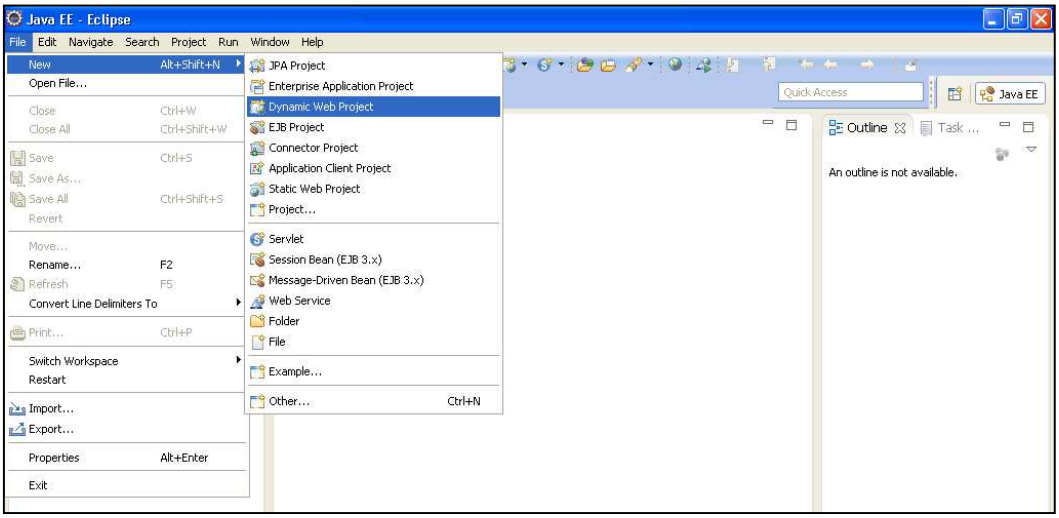


La figura anterior muestra que para programar un servlet HTTP es suficiente heredar de HTTPServlet y sobreescribir los métodos doGet, doPost, etc. dependiendo de a qué métodos HTTP se desea que responda el servlet. La implementación de service que proporciona HTTPServlet analiza automáticamente el método HTTP de la petición recibida para decidir a qué método del servlet debe invocar (doGet, doPost, etc.)

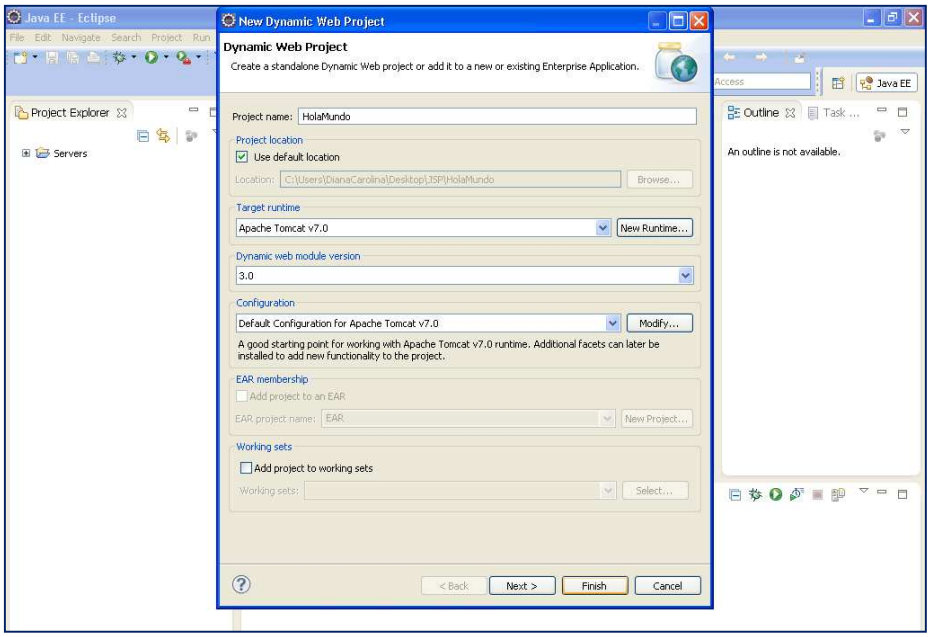
Ejemplo:

Crear un servlet que nos muestre un mensaje y los números del 1 al 10000.

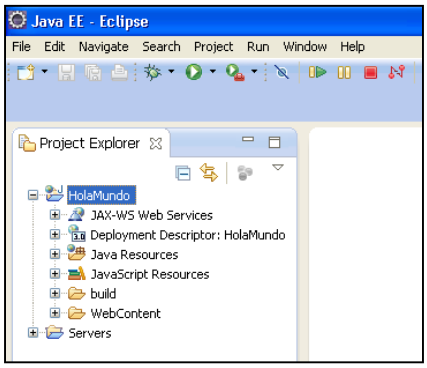
Desde el menú de opciones seleccionamos File -> New -> Dynamic Web Project:



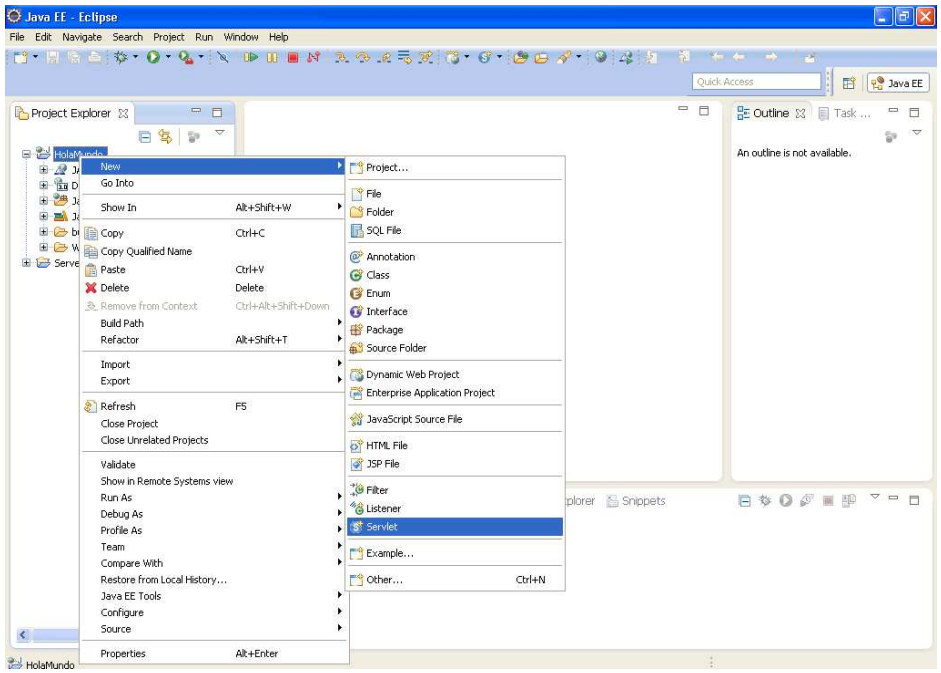
En el diálogo siguiente especificamos el nombre del proyecto (en nuestro caso le llamaremos HolaMundo) y presionamos el botón "Finish":



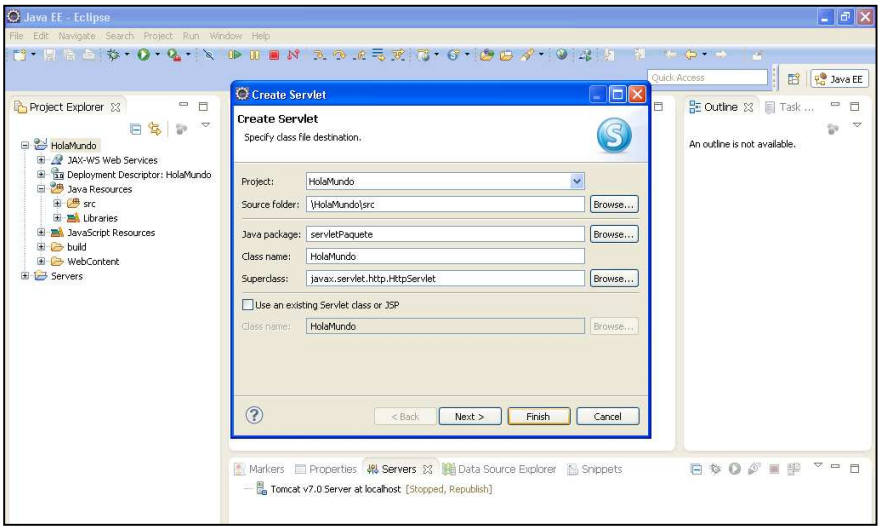
El Eclipse nos crea una serie de carpetas y archivos donde alojaremos los servlet:



Ahora presionamos el botón derecho sobre el nombre del proyecto y seleccionamos la opción New -> Servlet:



En el diálogo siguiente especificamos el nombre de nuestro servlet (en nuestro ejemplo le llamaremos HolaMundo), presionamos el botón "Finish" y ya tenemos el esqueleto básico de un servlet:



El código fuente generado es el siguiente:

```
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
/**
 * Servlet implementation class HolaMundo
 */
@WebServlet("/HolaMundo")
public class HolaMundo extends HttpServlet {
	private static final long serialVersionUID = 1L;
    /**
     * @see HttpServlet#HttpServlet()
     */
    public HolaMundo() {
        super();
        // TODO Auto-generated constructor stub
    }
	/**
	 * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
	 */
	protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
		// TODO Auto-generated method stub
	}
	/**
	 * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
	 */
	protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
		// TODO Auto-generated method stub
	}
}
```

Todo servlet debe heredar de la clase HttpServlet que se encuentra en el paquete javax.servlet.http

Esta clase debe sobreescribir el método doGet o doPost (o ambos) En el protocolo HTTP las peticiones pueden ser de tipo post (cuando llamamos a una página desde un formulario HTML) y de tipo get (páginas sin formulario)

Nuestro problema es mostrar un mensaje e imprimir los números del 1 al 10000, esta actividad la haremos en el método doGet.

El algoritmo a implementar en el método doGet para dicha salida es:

```
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;
/**
 * Servlet implementation class HolaMundo
 */
@WebServlet("/HolaMundo")
public class HolaMundo extends HttpServlet {
    private static final long serialVersionUID = 1L;

    public HolaMundo() {
        super();
        // TODO Auto-generated constructor stub
    }
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        // TODO Auto-generated method stub
         PrintWriter out = response.getWriter();
         out.println("<html>");
         out.println("<head></head>");
         out.println("<body>");
         out.println("<h1>Hola Mundo</h1>");
         for(int f=1;f<=10000;f++) {
             out.println(f);
             out.println(" - ");
         }
         out.println("</body>");
         out.println("</html>");
    }
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        // TODO Auto-generated method stub
    }
}
```

Una parte importante de la declaración del servlet que nos genera automáticamente el Eclipse es la anotación @WebServlet (esta línea registra el servlet para todas las peticiones al servidor con la sintaxis

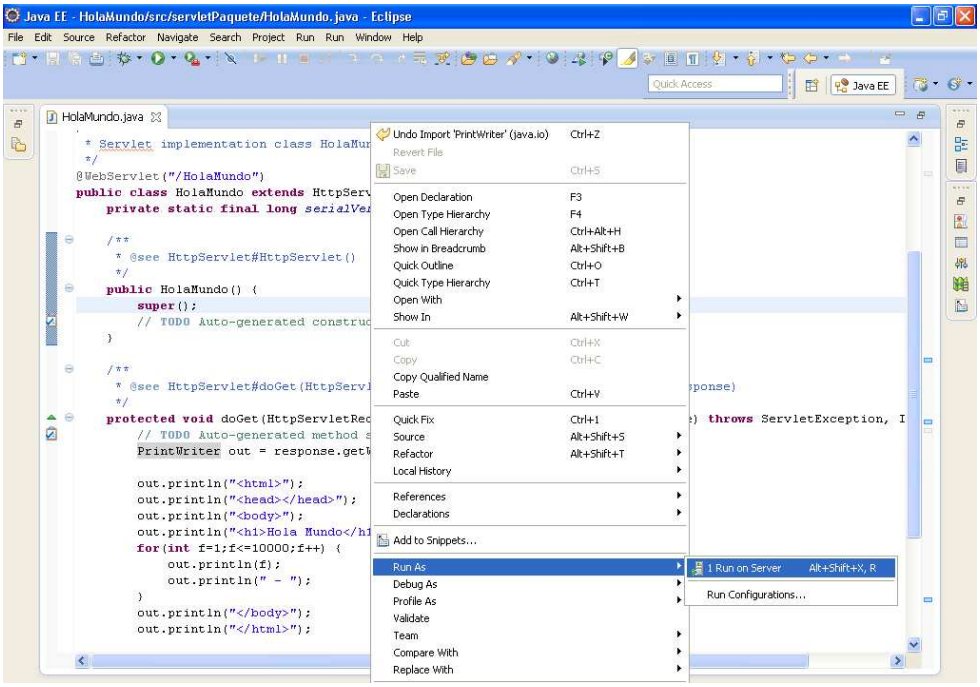
http://localhost:8080/nombreProyecto/nombredelServlet):@WebServlet("/HolaMundo")

Obtenemos una referencia de un objeto de la clase PrintWriter (debemos importar la clase PrintWriter) mediante la llamada al método getWriter del objeto response que llega como parámetro al método doGet:

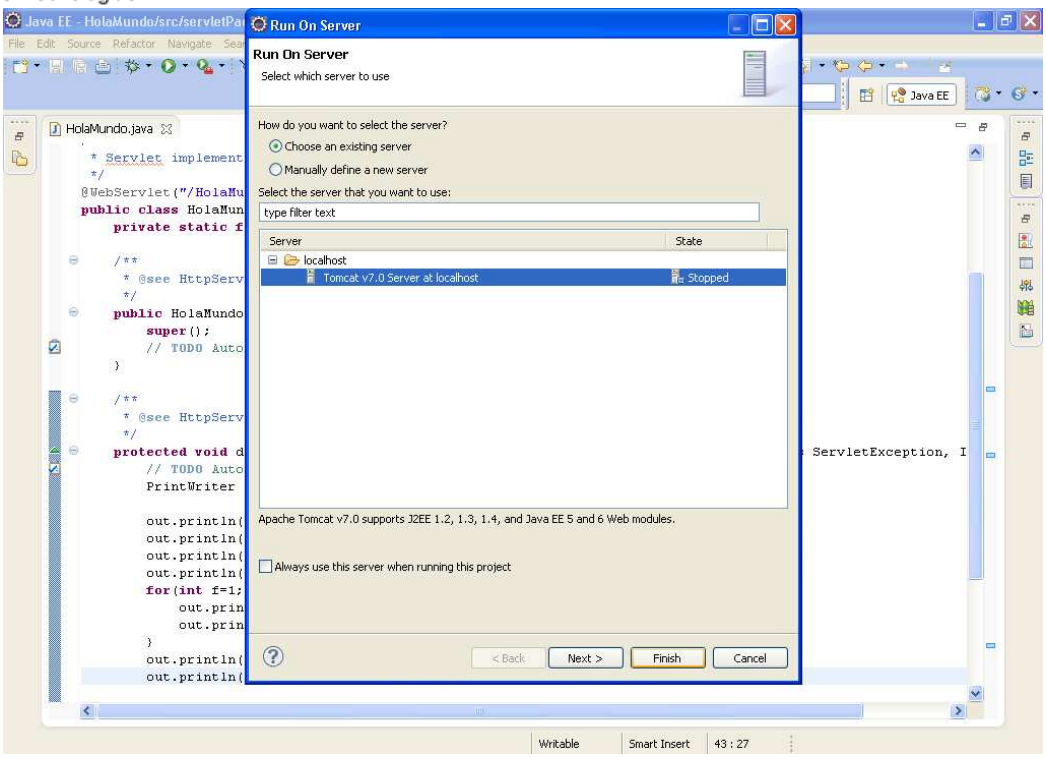
```
PrintWriter out = response.getWriter();
```

Todas las salidas son llamando al método println del objeto out de la clase PrintWriter. Como vemos generamos como salida HTML, para mostrar los números del 1 al 10000 es más conveniente utilizar una estructura repetitiva que hacer una salida secuencial.

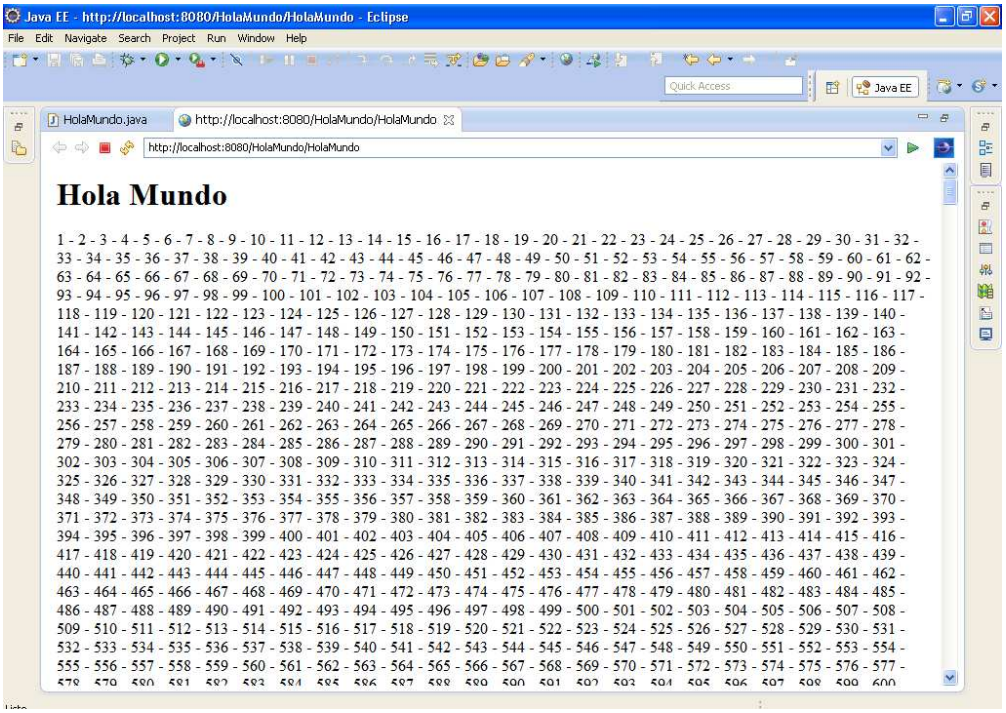
Para probar el servlet que acabamos de codificar debemos presionar el botón derecho del mouse sobre el nombre de la clase y seleccionar "Run on Server":



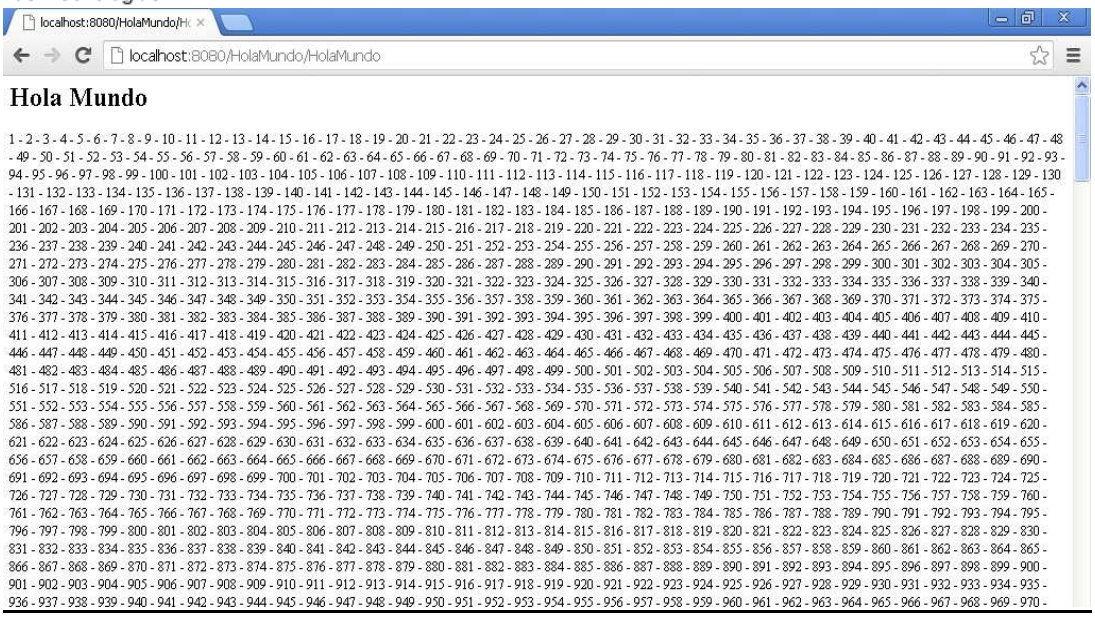
Aparece un diálogo que debemos seleccionar el botón "Finish" ya que está seleccionado el servidor "Tomcat" para ejecutar el servlet:



El resultado de la ejecución del servlet lo podemos ver dentro de una ventana dentro del mismo Eclipse:



Si queremos que el resultado aparezca en otro navegador podemos configurar desde el menú de Eclipse el navegador que muestra el resultado que devuelve Tomcat:



JAVA JSP (JAVA SERVER PAGE)

Java Server Page, es otra de las nuevas tecnologías para tratar de hacer más eficiente el modelo cliente-servidor y sobre todo la construcción de sistemas de comercio electrónico.

En este modelo una página HTML también incluye código en java, es el servidor de páginas quien al estar mandando la página a la PC remota la compila y la convierte en un servlet.

Esta tecnología combina en una sola aplicación, tanto código HTML como código java.

El proceso de crear un jsp, es sencillo se crea un archivo normal con notepad combinando código HTML y código java, se graba con extensión .jsp, se hace un ftp al servidor y listo.

Cuando el usuario requiere un jsp el servidor lo carga, lo compila, lo convierte a servlet y manda la página resultante al usuario remoto.

Características

- Código separado de la lógica del programa.
- Las páginas son compiladas en la primera petición.
- Permite separar la parte dinámica de la estática en las páginas web.
- El código JSP puede ser incrustado en código HTML.

Ventajas

- Ejecución rápida.
- Crear páginas del lado del servidor.
- Multiplataforma.
- Código bien estructurado.
- Integridad con los módulos de Java.
- La parte dinámica está escrita en Java.

Notas:

1.- Para insertar código java dentro de una página HTML se deberán usar una serie de tags o delimitadores (en el ejemplo se está usando <% una serie de instrucciones de java %>) donde cada uno de ellos tiene un propósito definido.

Otros delimitadores son:

Comentarios <%– comentario –%>

Ignorados cuando jsp es convertida a servlet y muy útiles para documentar nuestros programas jsp.

Declaración <%! Variables, métodos, etc. %>

Recordar que todo buen programa, empieza declarando variables.

Instrucción <%= instrucción %>

Para poner una y solo una instrucción de java, además recordar que ya existen aparte ciertas instrucciones o variables predefinidas tales como request, response, out, session, application, config, and pageContext(tambien disponibles en scriptlets).

Recordar además que cuando se use <%= una sola instrucción %>, la instrucción no debe terminar con punto y coma.

Scriptlet <% todo un programa completo %>

Un scriptlet es un grupo de instrucciones de java, como se deduce de esta definición, se usara muchos scriptlets en nuestros jsp.

Aquí si, las instrucciones deben terminar con punto y coma

Un bloque de instrucciones <% bloque java %>, puede empezar (<%) en un scriptlet y terminar en otro scriptlet, pero asegurarse de que todos los scriptlets se abran y se cierren.

Include Directive <%@ include file="url" %>

Se usa para incluir archivos en la PC que compila la jsp, esto se realiza al tiempo que la jsp es convertida en servlet, el url debe ser relativo.

Para este caso también es válido:

jsp:include action

Para incluir el archivo al tiempo de request por parte de un usuario remoto

jsp:forward Action <jsp:forward page="URL relativo"/>
Manda llamar o enlazar otra página.

¿POR QUÉ JSP?

Los Servlets Java son más eficientes, fáciles de usar, más portables, y más baratos que el CGI tradicional y otras muchas tecnologías del tipo CGI.

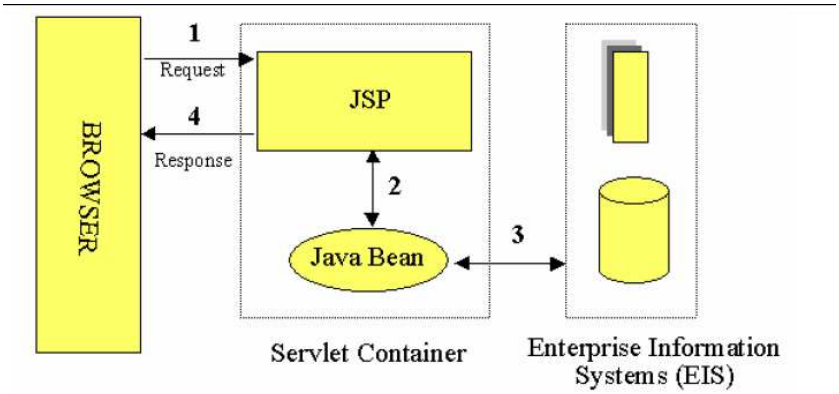


Figura 3.1 Esquema funcionamiento tecnología JSP

Como se ve en la figura anterior, en general, el esquema de funcionamiento de una aplicación bajo Servlets y JSP es muy sencilla. Los usuarios de la aplicación se comunican con ésta mediante objetos “request" y “response”. Las variables que se guardan en éstos objetos tienen poco tiempo de vida (lo que se tarda en mostrar la información al usuario mediante JSP). Si se quieren guardar variables para el control de los usuarios que entran en el sistema, se utilizarán objetos “session” para guardarlas. Estos objetos “session” permanecen activos con los datos hasta que el usuario cierra la sesión o directamente se cierra el navegador. Los JSP no son más que HTML en el cual se puede insertar fragmentos de código Java.

NOTA:

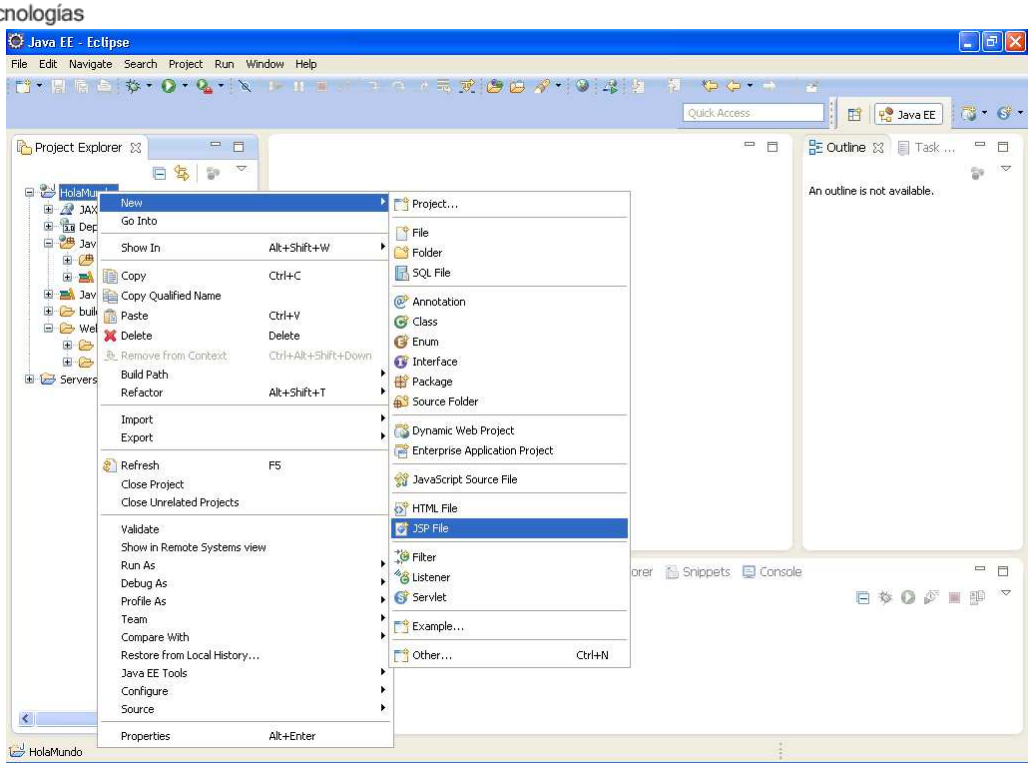
Para utilizar JDBC en un archivo JSP, se necesita el driver de conexión con la BD (en nuestro caso MYSQL, "mysql-connector-java-5.1.18-bin.jar"). Este jar o librería debe estar alguna ubicación del classpath que use Tomcat para tu proyecto.

Para ello, lo colocamos dentro de la carpeta "WEB-INF/lib" del proyecto que lo requiera, o en la carpeta "common/lib" dentro de Tomcat, que como su nombre indica son las librerias comunes a todos los proyectos.

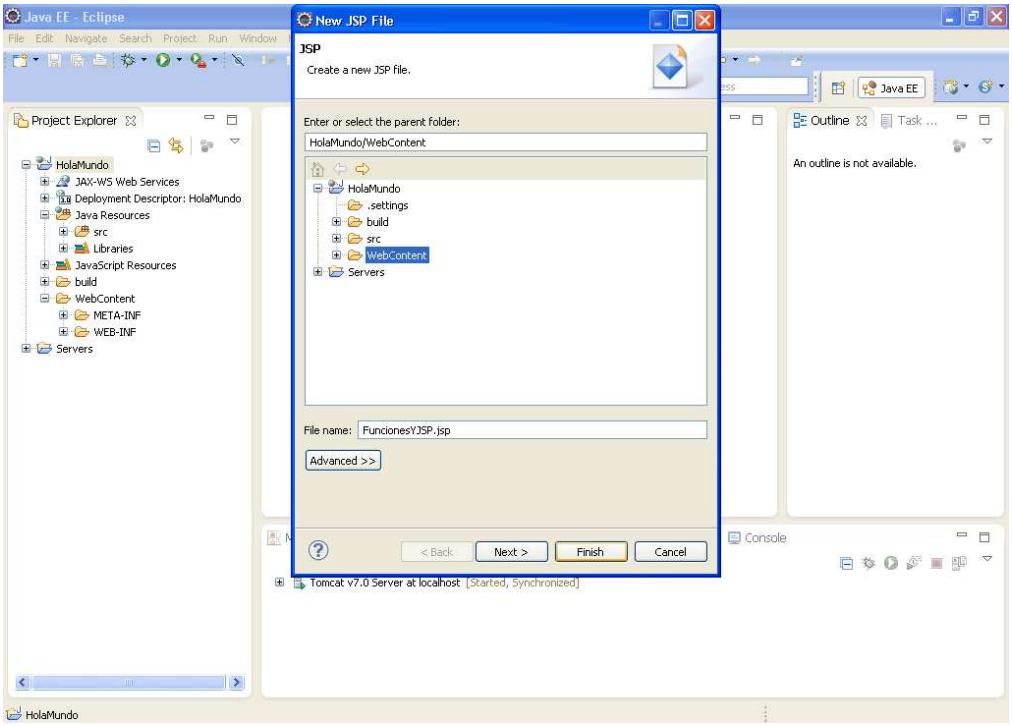
Ejemplo:

Uso de métodos dentro de un archivo .jsp

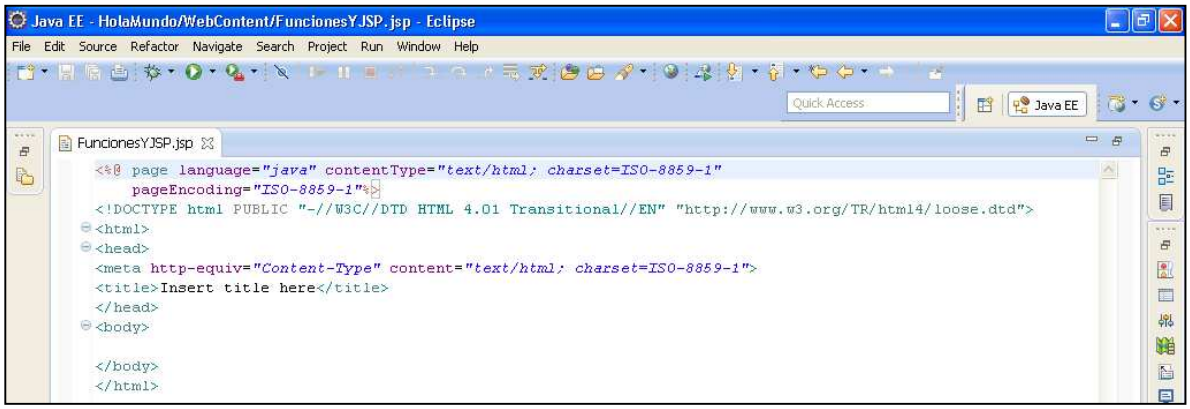
Para ello creamos un archivo con extensión .jsp



El archivo debe ir dentro de la carpeta de WebContent, en esta carpeta podemos colocar también archivos .html o .css



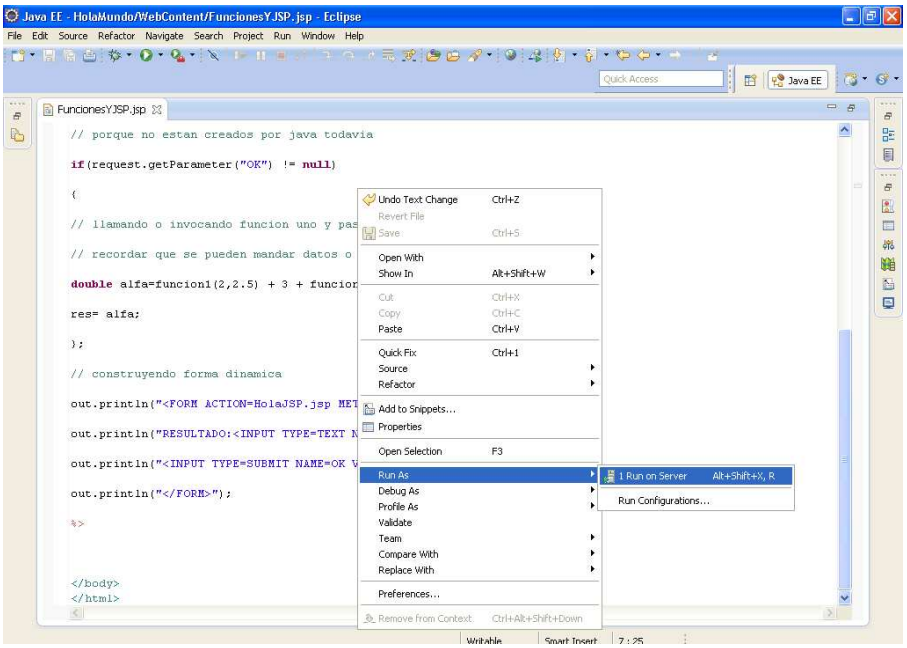
El código que nos genera es el siguiente:

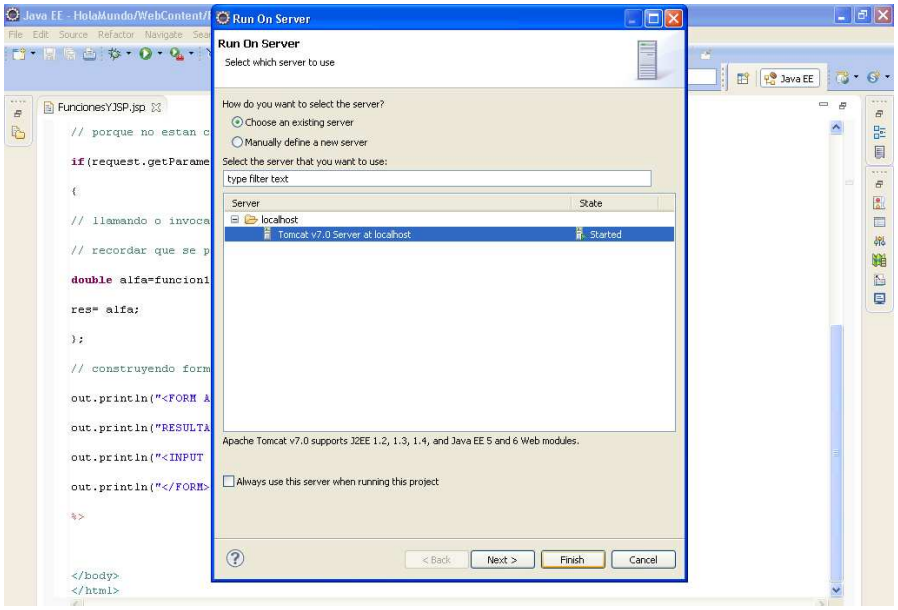


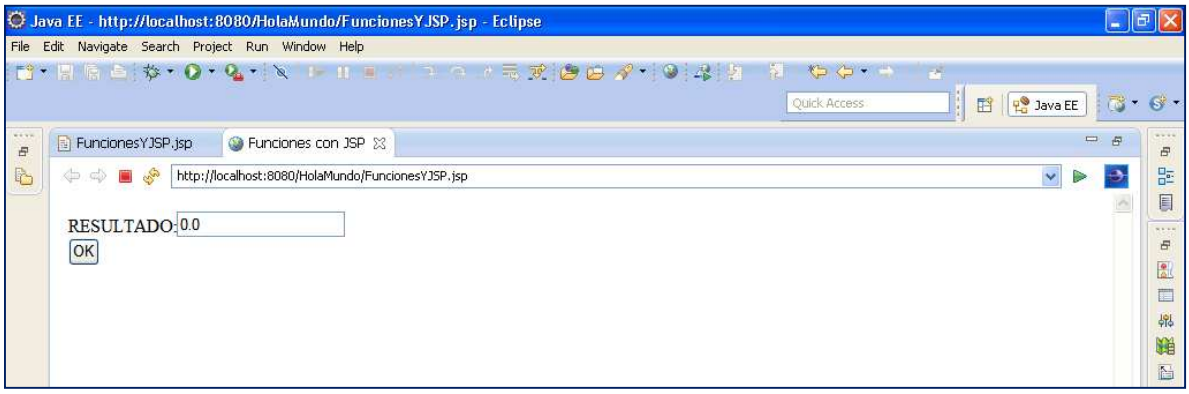
El código de nuestra aplicación es:

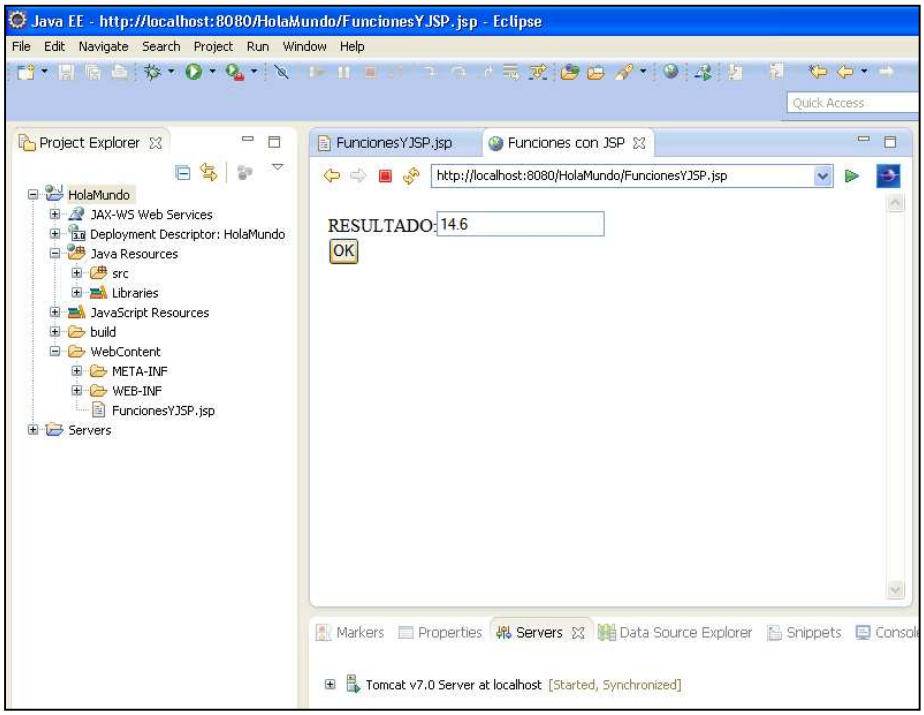
```
<%!
double res=0;
double funcion1(int a, double b){
return a * b; };
%>
<%
// no usar objetos request y out fuera de scriptlet
// porque no estan creados por java todavia
if(request.getParameter("OK") != null){
// llamando o invocando funcion uno y pasando parametros
// recordar que se pueden mandar datos o variables
double alfa=funcion1(2,2.5) + 3 + funcion1(2, 3.3);
res= alfa;
};
// construyendo forma dinamica
out.println("<FORM ACTION=prog14.jsp METHOD=post>");
out.println("RESULTADO:<INPUT TYPE=TEXT NAME=RESULTADO value="+res+"><BR>");
out.println("<INPUT TYPE=SUBMIT NAME=OK VALUE=evento1 ><BR>");
out.println("</FORM>");
%>
```

Lo ejecutamos









SESIONES

Tomcat mantiene automáticamente las sesiones de usuario:

- Por defecto, utiliza cookies para que el cliente envíe su identificador de sesión en cada petición.
- Cada sesión se representa con un objeto HttpSession.
- Una sesión caduca tras un tiempo (configurable) sin recibir peticiones correspondientes a la misma.

Obtención del objeto sesión desde el servlet:

HttpServletRequest.getSession(boolean create):

- Devuelve el objeto de sesión correspondiente a la petición.

- Con parámetro true, crea una nueva sesión si la petición no corresponde a ninguna sesión.

Se puede almacenar objetos en la sesión:

- HttpSession.setAttribute(String name, Object value)
- HttpSession.getAttribute(String name)

COOKIES

1.-Crear un objeto cookie a partir de la clase Cookie con

Cookie Galleta = new Cookie(NomCookie, ValCookie);

donde "NomCookie" será el nombre de la cookie y "ValCookie" será el valor asociado a esta cookie.

2.-Vamos a fijar su fecha de caducidad/expiración por medio del método,

Galleta.setMaxAge(1*365*24*60*60); //Expira dentro de un año.

Galleta.setMaxAge(-1); //Expira al finalizar la sesión del cliente/navegador.

3.- Vamos a eliminar una cookie haciendo que su expiración sea cero, con

Galleta.setMaxAge(0); //El cliente/navegador eliminará la cookie en la siguiente llamada.

4.- Vamos a limitar el directorio donde se encuentra la página jsp o servlet que tendrá acceso a las cookies, con

Galleta.setPath("/"); //Directorio por defecto desde donde se capturarán las cookies.

3.-Vamos a enviar esta cookie al solicitante de la página a través del objeto implícito response con

response.addCookie(Galleta);

5.-Vamos a recuperar todas las posibles cookies almacenadas en el cliente con,

Cookie[] galleta = request.getCookies(); //Recuperar todas las cookies

Cookie[] galleta = request.getCookies("kuki"); //Recuperar una cookies llamada kuki

ARCHIVO WAR

Normalmente cuando se trabaja sobre un proyecto web J2EE se hace sobre una herramienta de desarrollo como eclipse y para probar la aplicación la herramienta suele hacer un despliegue automático para poder hacer pruebas de forma rápida.

Cuando tenemos la aplicación terminada el despliegue se hará sobre un servidor de producción. Este servidor es el que usarán los usuarios de la aplicación Web. Este despliegue normalmente se realiza utilizando un archivo WAR.

- Se publica nuestra aplicación en Tomcat. Paso conocido como Despliegue (Deployment)
- Se inicia el servidor Tomcat
- Se abre un browser o cliente http interno apuntando a http://localhost:8080/first-jee/login.html

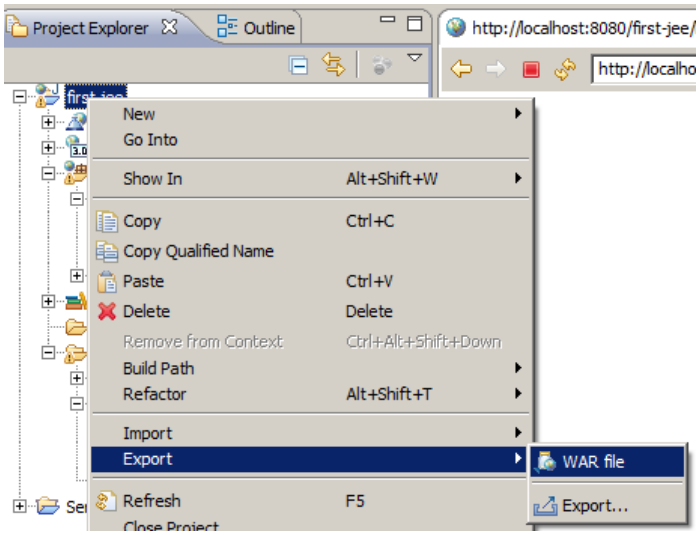
Lo primero que uno creería es que en el paso 1 eclipse publica el contenido de estas carpetas Java Resources y WebContent en el directorio webapps de Tomcat, pero Eclipse publica en un directorio temporal dentro del workspace. En mi caso

C:\eclipseJEE2\workspace\.metadata\.plugins\org.eclipse.wst.server.core\tmp0\wtpwebapps

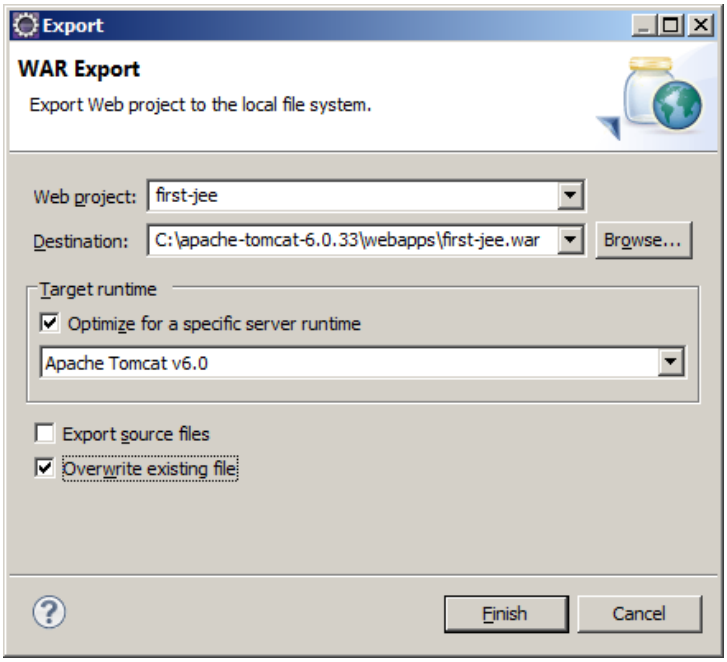
Un archivo WAR es un archivo comprimido con todos los archivos que hasta ahora desplegamos manualmente en webapps. Este archivo internamente tiene la misma estructura que usamos anteriormente: directorio WEB-INF, lib, classes, etc. Para desplegar un WAR en Tomcat basta con copiarlo al directorio webapps.

Una buena razón para utilizar archivos *.war es que normalmente los equipos de desarrollo de aplicaciones y los de instalación están conformados por distintas personas. Enviar un solo archivo para desplegar es más sencillo y presta a menos confusiones que enviar varios.

Eclipse JEE permite generar el archivo WAR de la aplicación seleccionando el proyecto, click en botón derecho del ratón - Export -WAR file:



En Destination colocamos el lugar donde se guardara el archivo WAR. En este caso será en el directorio webapps de la instalación de Tomcat.



Para probar la aplicación sólo tengo que iniciar Tomcat ejecutando

C:\apache-tomcat-6.0.33\bin\start.bat

y a continuación abrir un browser con

http://localhost:8080/first-jee/login.html

por defecto en el url se pone el nombre de WAR sin la extensión:

first-jee.

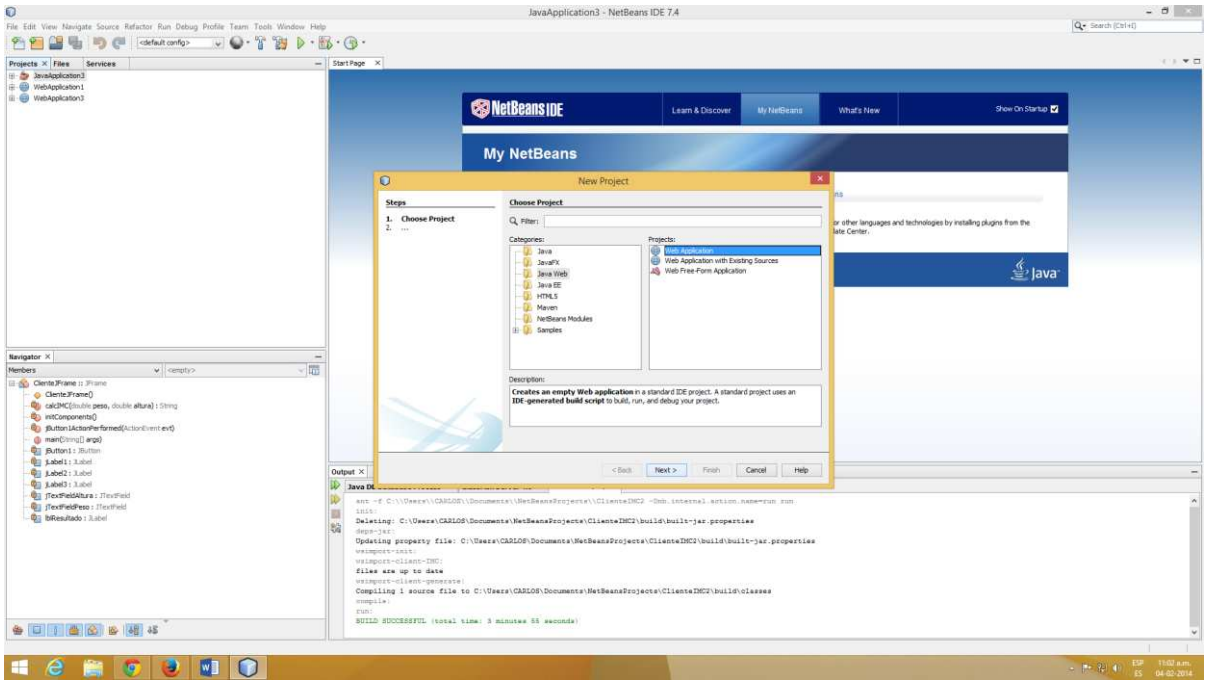
Complemento

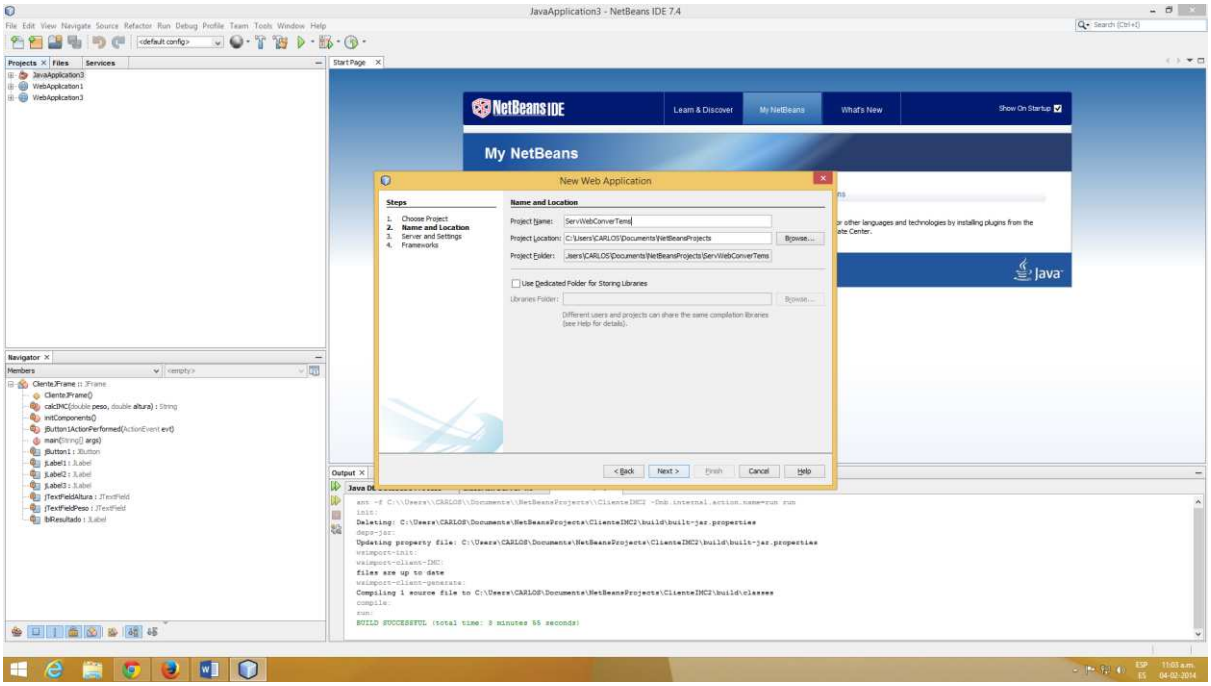
Servicios Web desarrollados con herramientas RAD

1. Crear un nuevo servicio Web XML
2. Escribir la interfaz del servicio Web.
3. Instalar el servicio Web en un servidor.
4. Crear un proyecto cliente del servicio Web.

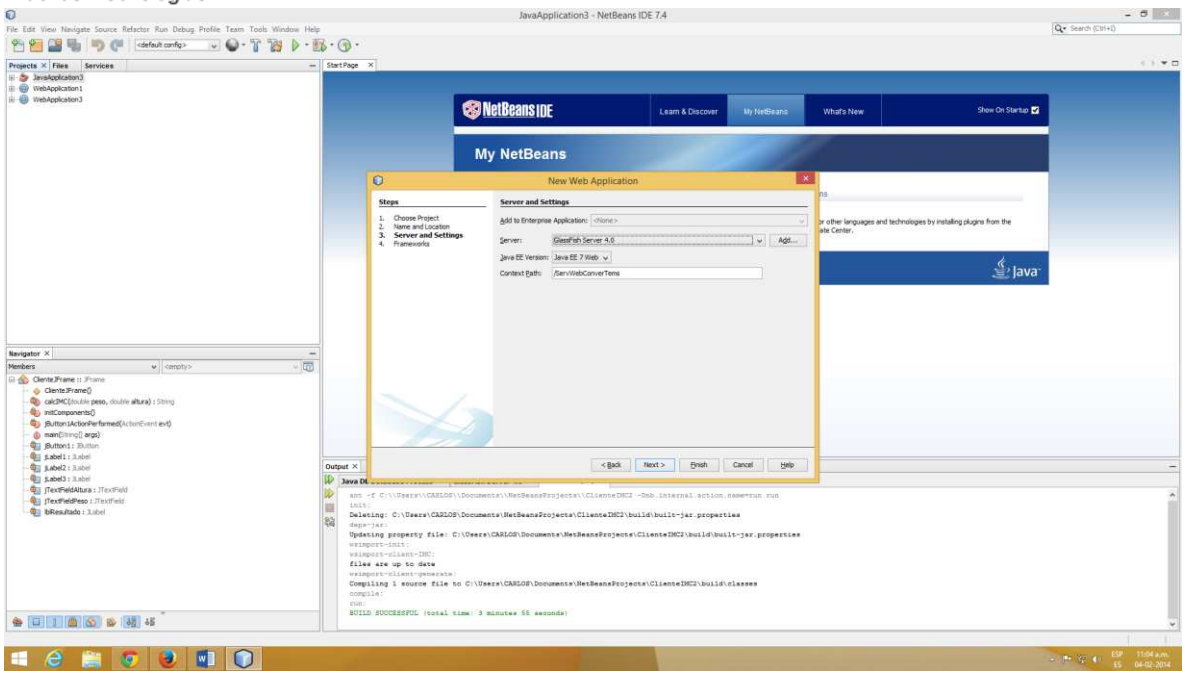
Crear un nuevo servicio Web XML

Crear un proyecto y luego crear una aplicación web.



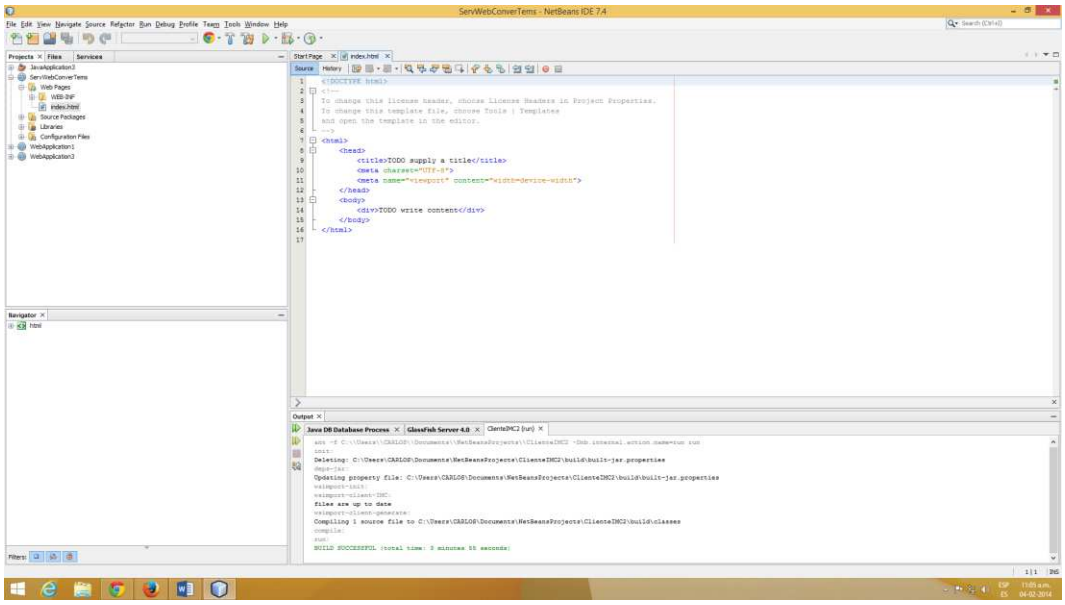


Colocar nombre del proyecto.

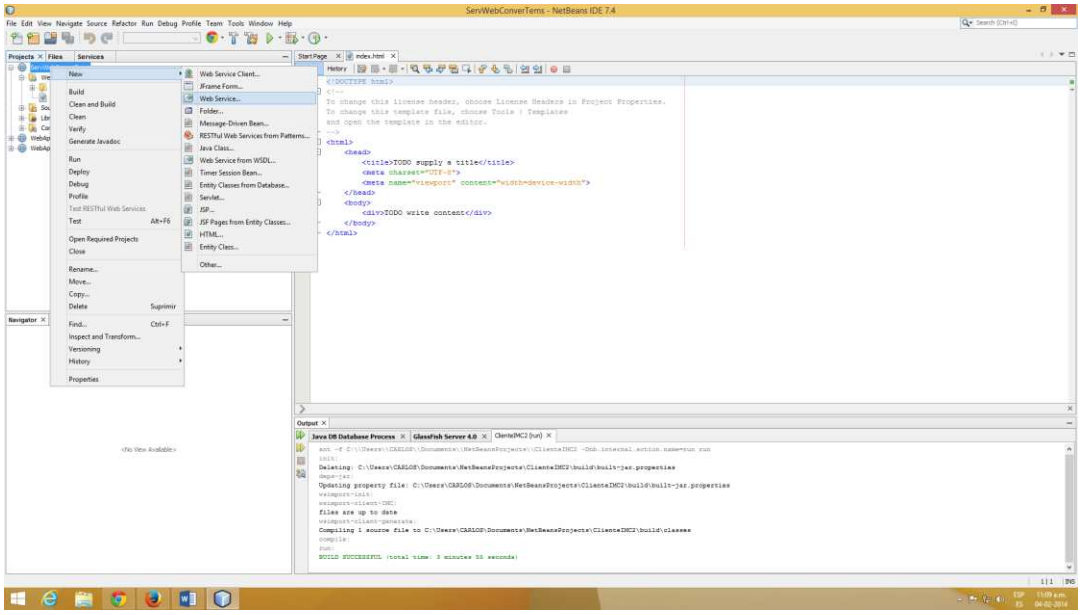


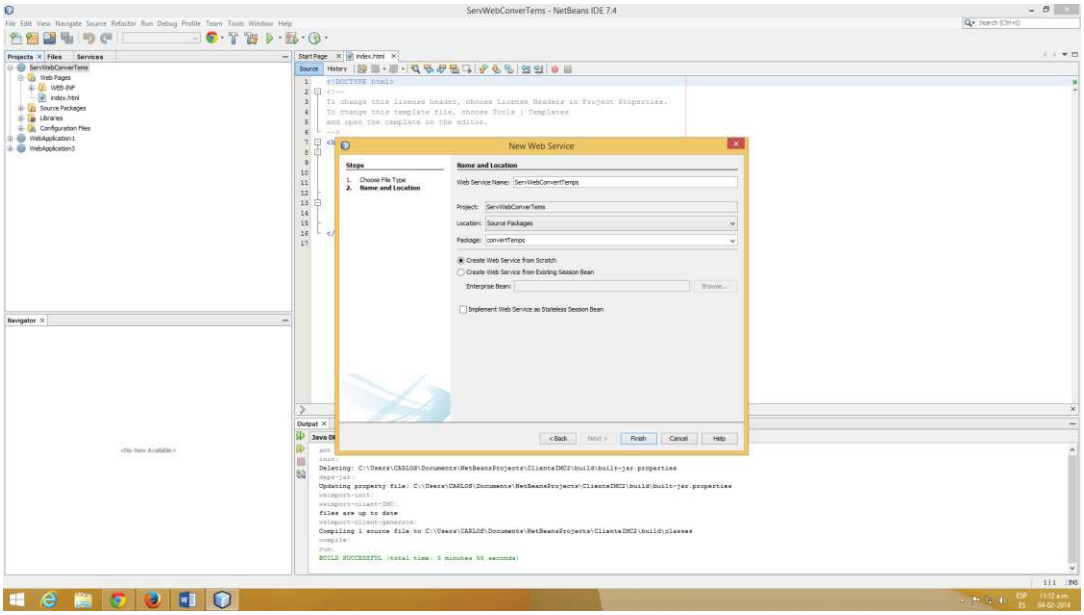
Seleccionar servidor y versión de java y luego presionar “Finish”.

Obtenemos:



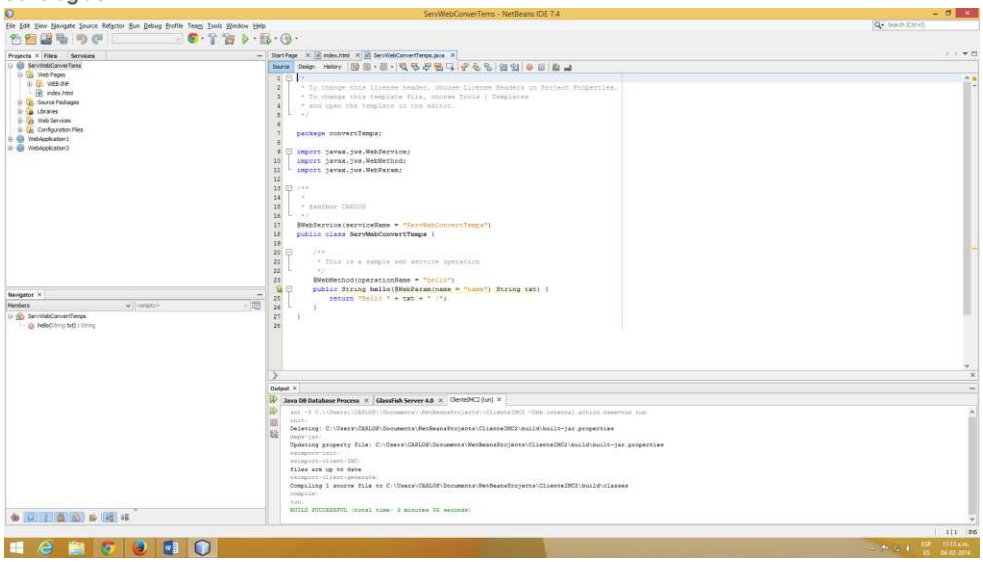
Añadir a la aplicación web un Servicio Web:



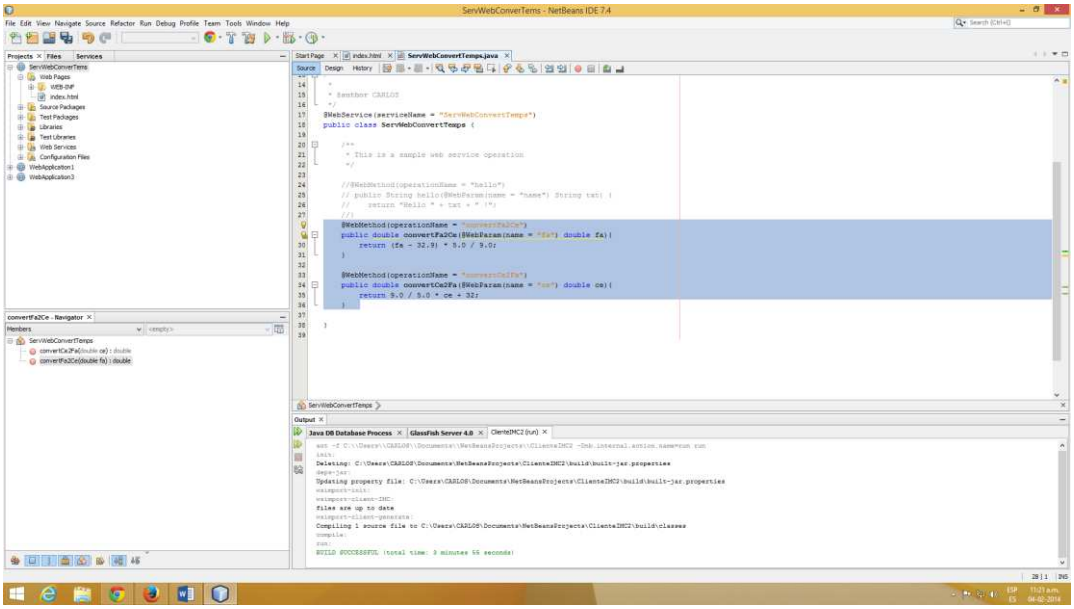


Presionar “Finish”

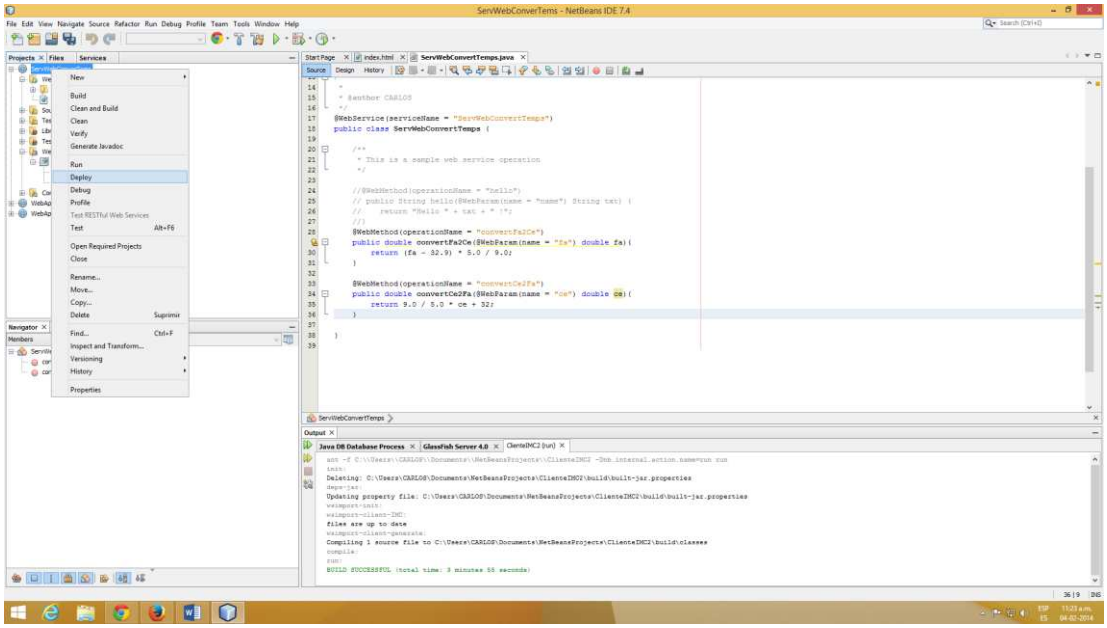
Y obtenemos:



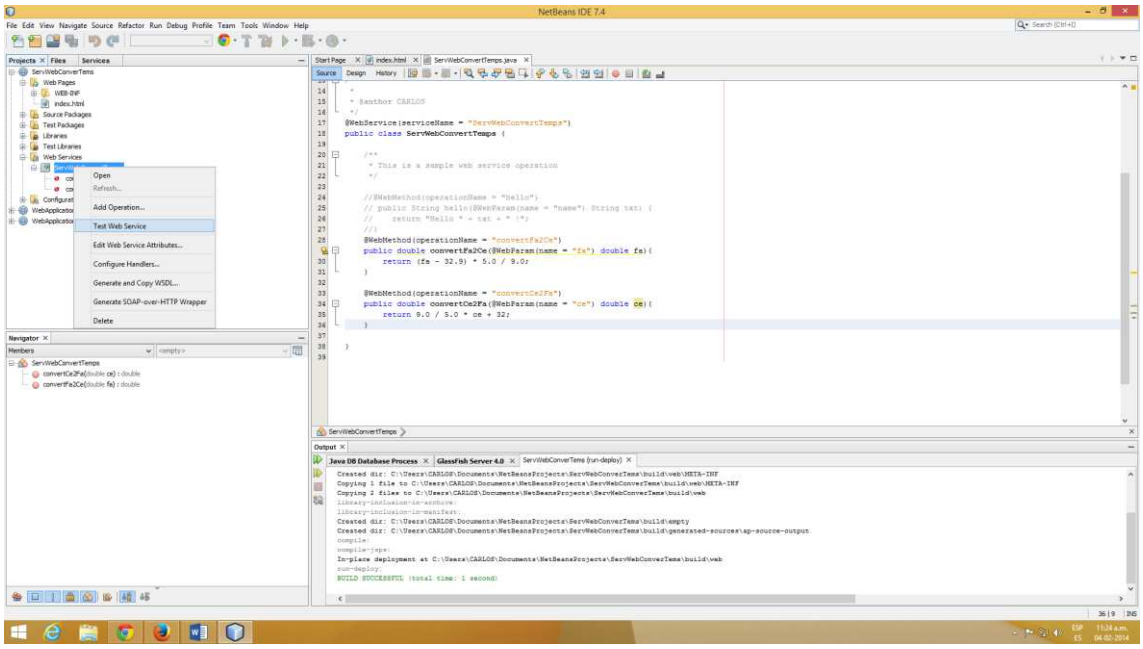
Agregar los métodos necesarios que cumplirán con las funciones requeridas:



Deploy la aplicación:



Probar el Web Service:



Y obtenemos:

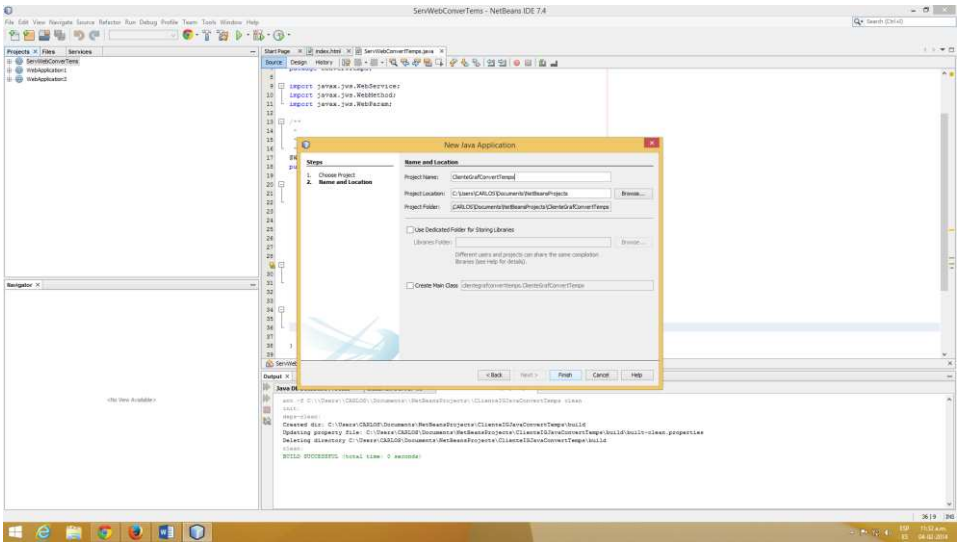
Aquí podemos comprobar el funcionamiento de nuestro WS.

Crear un cliente del servicio Web

1. Crear la interfaz gráfica con Swing para un cliente Web XML.
2. Crear la interfaz de código que facilite la comunicación entre el cliente y el servicio Web XML. Esta interfaz recibe el nombre de proxy.
3. Incluir una referencia al proxy en el cliente que permita acceder a la interfaz mostrada por el servicio Web.
4. Crear una instancia del proxy en el cliente que permita acceder a la interfaz mostrada por el servicio Web.
5. Llamar al método del proxy correspondiente al método del servicio Web XML que desea ejecutar.

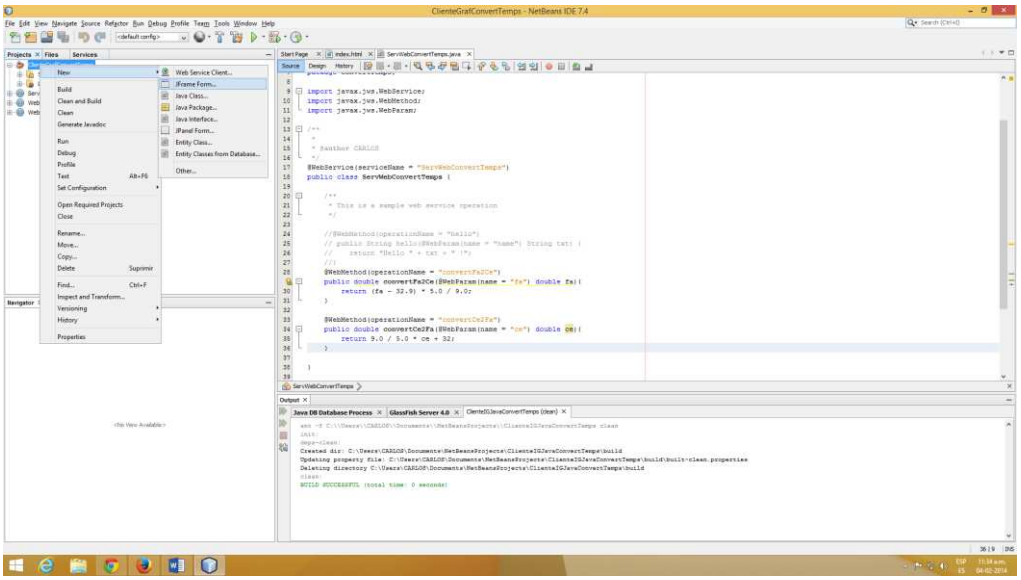
Aplicación Java gráfica como cliente de un servicio Web

1. Crear “Nuevo proyecto”.
2. Seleccionar categories -> Java, Projects -> Java Applications -> Next. Se muestra “New Java Application”
3. Escribir nombre del proyecto; En nuestro caso Cliente-JavaConverTemps; y a continuación seleccione donde quiere guardarlo. No marque la opción “Create Main Class” y así crear un proyecto vacío.
4. Hacer click en el botón “Finish”.

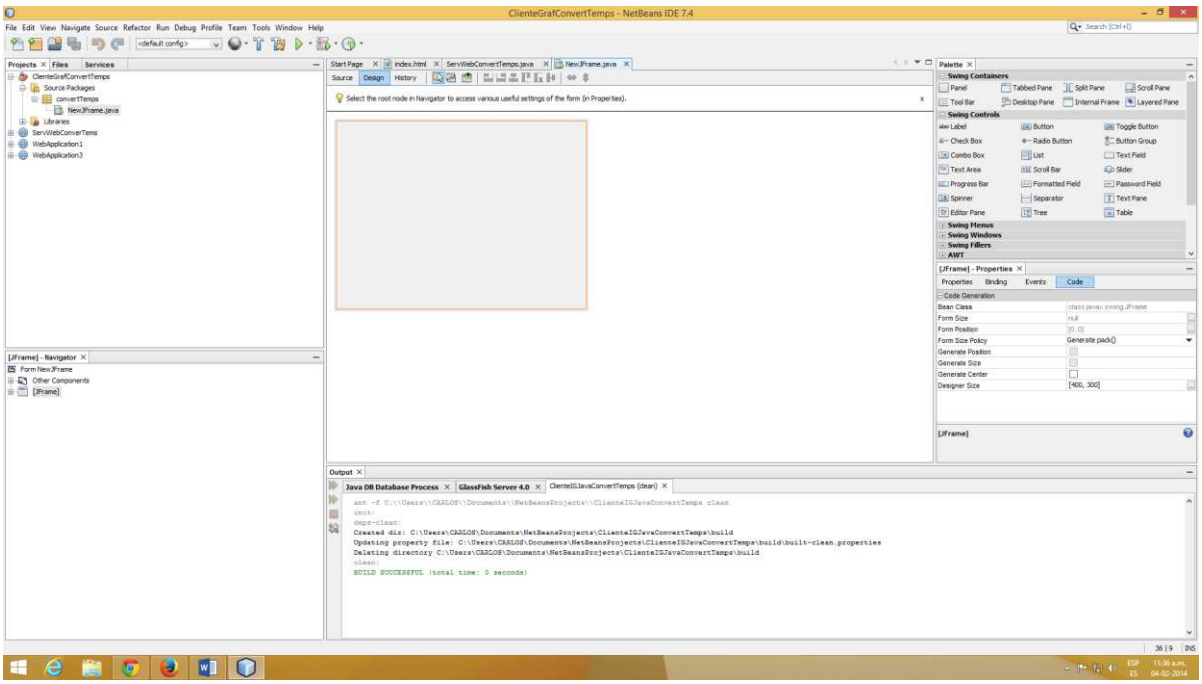


Pulsamos “Finish” y obtenemos:

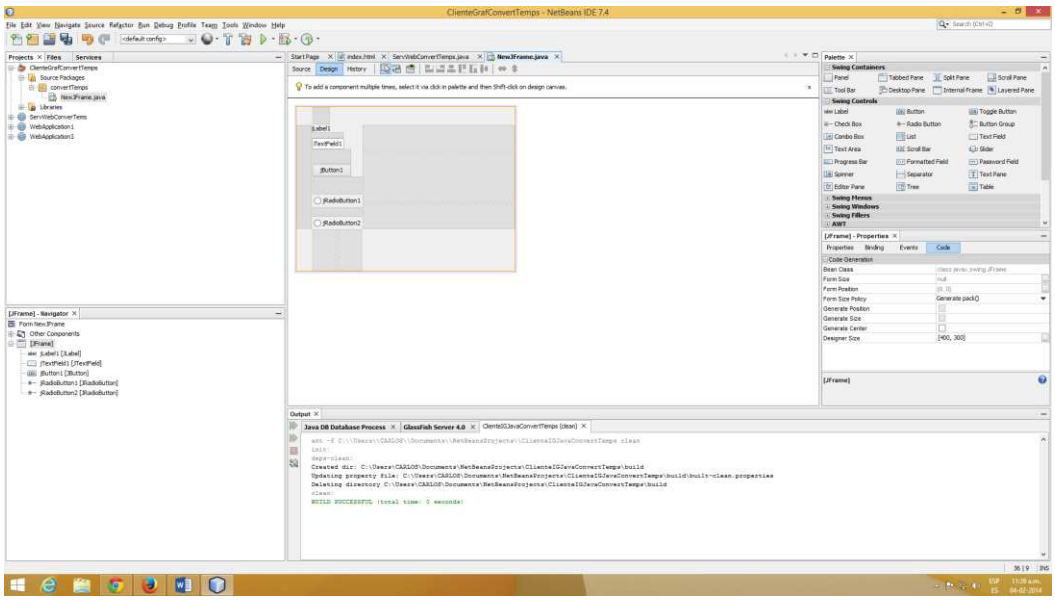
Añadir un formulario jFrame al proyecto:



Y obtenemos:



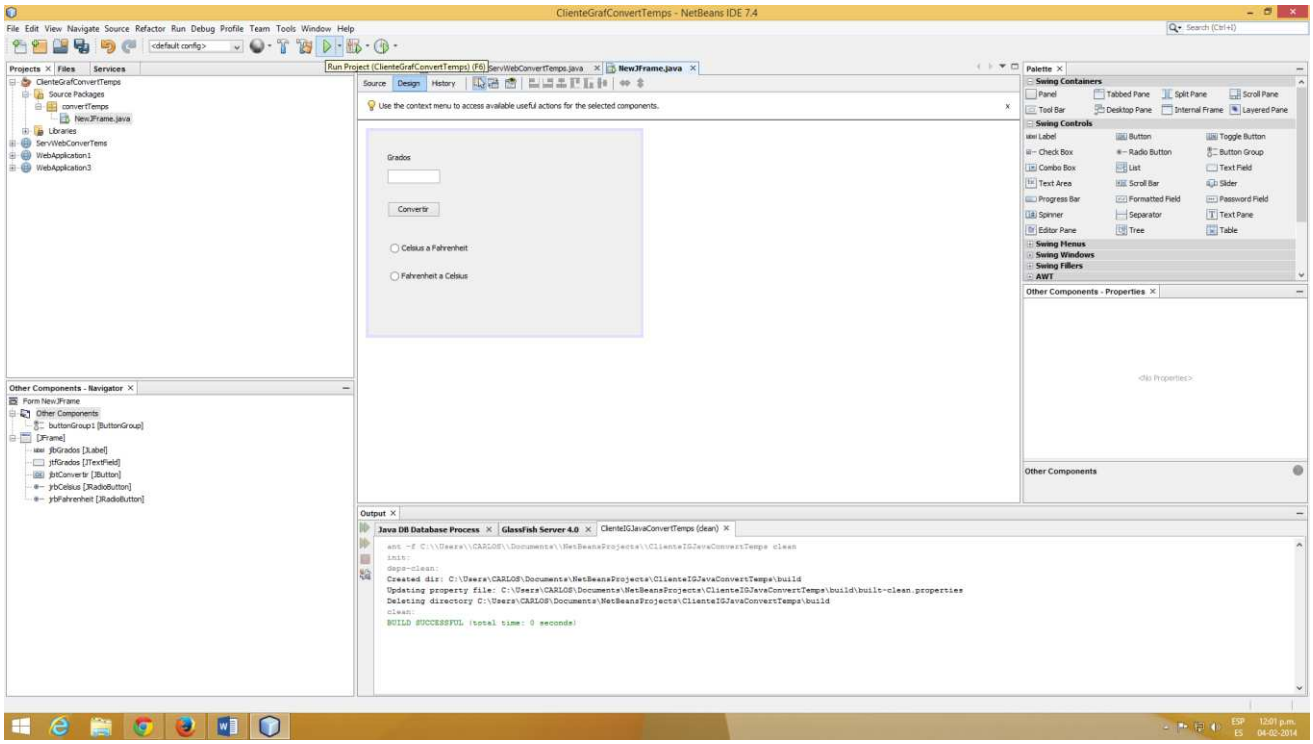
Agregamos los elementos necesarios: una etiqueta, una caja de texto, un botón y dos botones de selección.



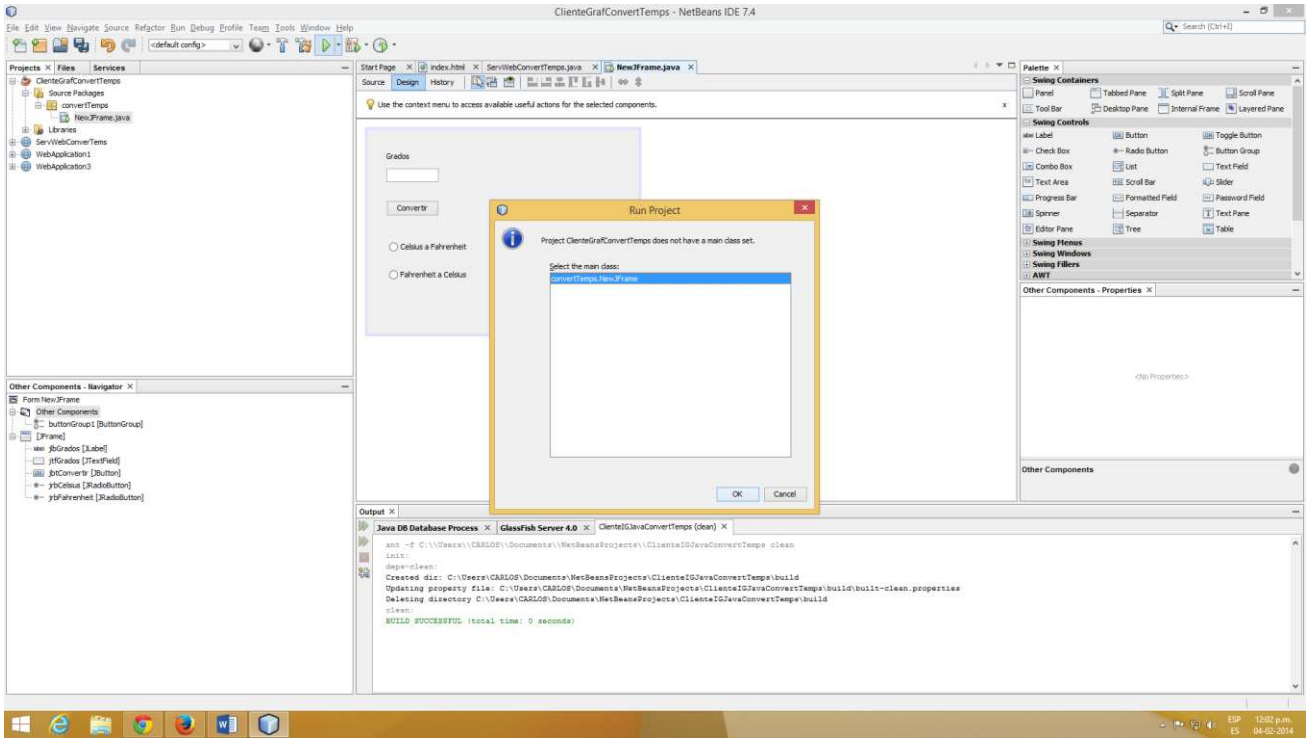
A continuación, añada al proyecto un formulario, objeto JFrame, y coloque sobre él los controles indicados en la tabla siguiente con las propiedades especificadas:

Objeto	Propiedad	Valor
Etiqueta	Variable text	jlbGrados Grados:
Caja de texto	Variable Text Alineación horizontal	jtfGrados 0.00 RIGTH
Botón de pulsación	Variable text	jbtConvertir Convertir
Grupo de botones	Variable	jbgGrados
Botón de opción	Variable Text Selected ButtonGroup	jrbCentAFahr Centigrados a Fahrengeit True jbgGrados
Botón de opción	Variable Text Selected ButtonGroup	jrbFahrACent Fahrengeit a Centigrados False jbgGrados

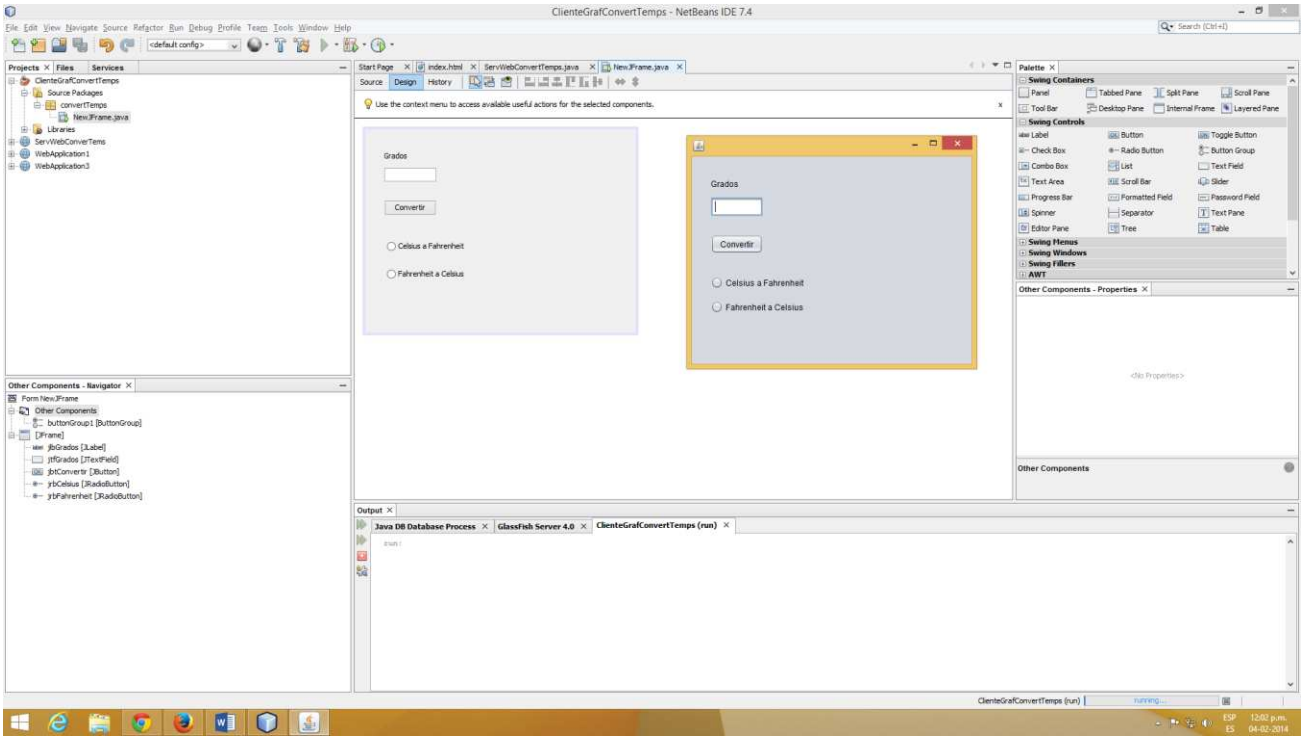
Ahora tenemos:



Probamos nuestra interfaz gráfica.

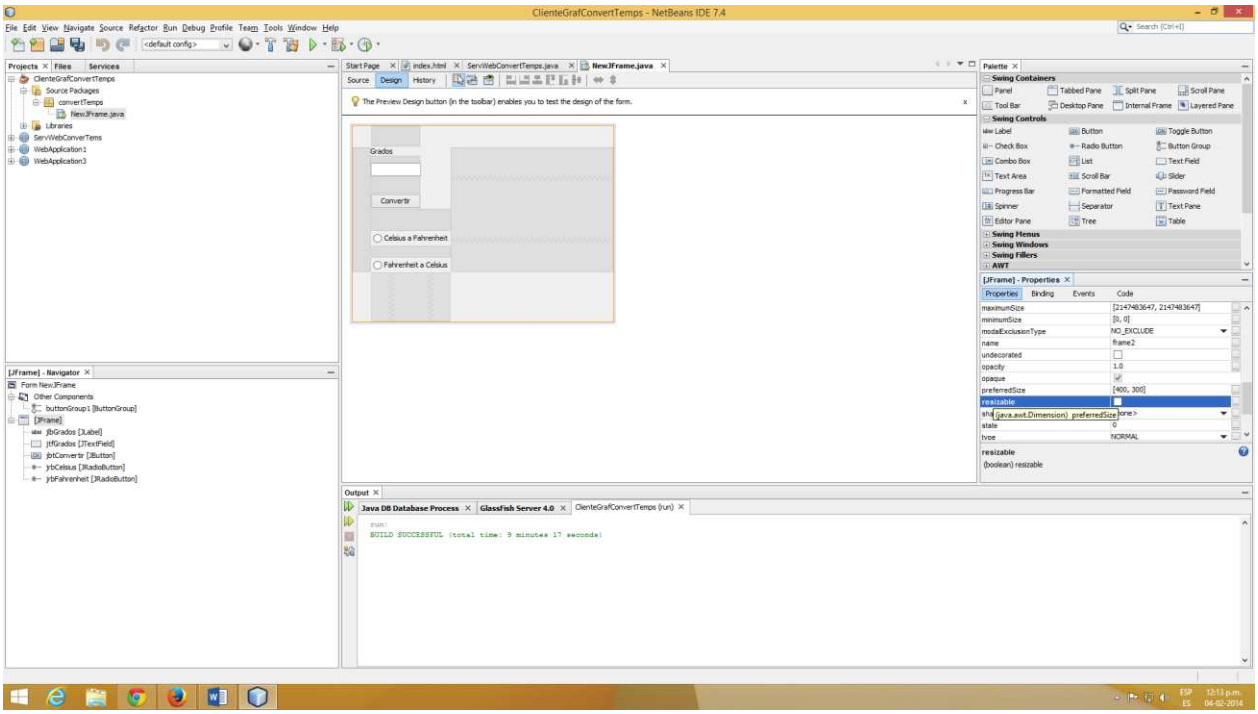


Y obtenemos:



Esta interfaz no tiene ninguna funcionalidad, así que debemos agregarle el código necesario.

Finalmente haga click sobre el formulario y, desde la ventana de propiedades, cambie su propiedad “title” al valor “Conversion Centigrados ←→ Fahrengeit y su propiedad “resizable” a valor “false”.



Pensemos ahora como sucederán los hechos cuando un usuario solicite que se ejecute nuestra aplicación Java:

1. Se visualiza el formulario Java.
2. El usuario selecciona el tipo de conversión que desea realizar haciendo clic sobre el botón de opción correspondiente.
3. Escribe en la caja de texto el valor en grados que se desea convertir.
4. Hacer clic en el botón “Convertir”. El controlador de este botón, en función del tipo de conversión solicitado, invocará al método ConvCentAFahr o ConvFahrACent del servicio web pasando como argumento el valor de la caja de texto.

Para llevar a cabo el punto 4. El cliente debe tener un medio de comunicar con el servicio Web XML y de encontrarlo durante la ejecución. Este lo hace a través de una referencia al servicio WebXML agregada al proyecto. Al agregar esta referencia, como veremos a continuación, se generan varias clases (proxy) que proporcionan al cliente una representación local del servicio para interactuar con él.

Descubrimiento del servicio Web XML.

El descubrimiento de servicios Web XML, es el proceso por el que un cliente encuentra un servicio Web XML, y obtiene su descripción: documento XML, que utiliza el lenguaje de descripción de servicios Web (WSDL: Web Services Description Language). ¿Cómo se realiza este proceso? Conociendo la dirección URL de un documento de descubrimiento residente en un servidor Web, el programador de una aplicación cliente puede generar en el equipo local un conjunto de ficheros, que contienen información acerca de los servicios Web XML, de sus capacidades y de la forma correcta de interactuar con ellos. Este proceso resulta muy sencillo utilizando los Web Service Client. Un cliente

de servicios Web Java tiene la capacidad de proporcionar los artefactos necesarios para que una aplicación cliente pueda comunicarse con un servicio Web. Basta con conocer la URL del archivo WSDL, o del servicio Web. Por ejemplo para el servicio que hemos desarrollado esta URL sería así:

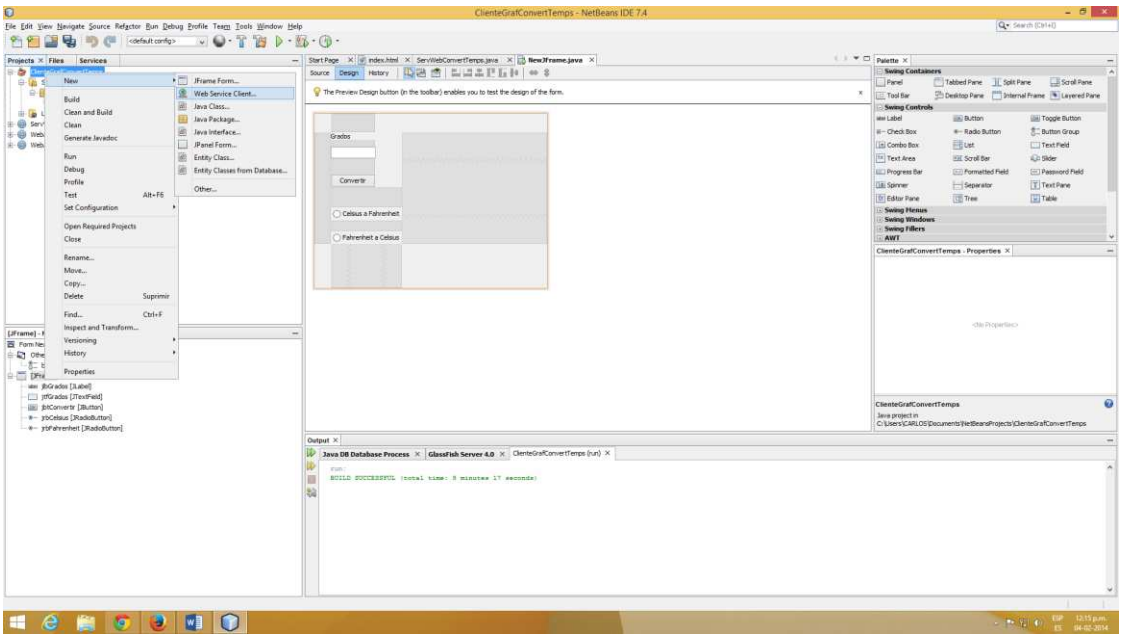
http://localhost:8080/ServWebConverTempsService?wsdl

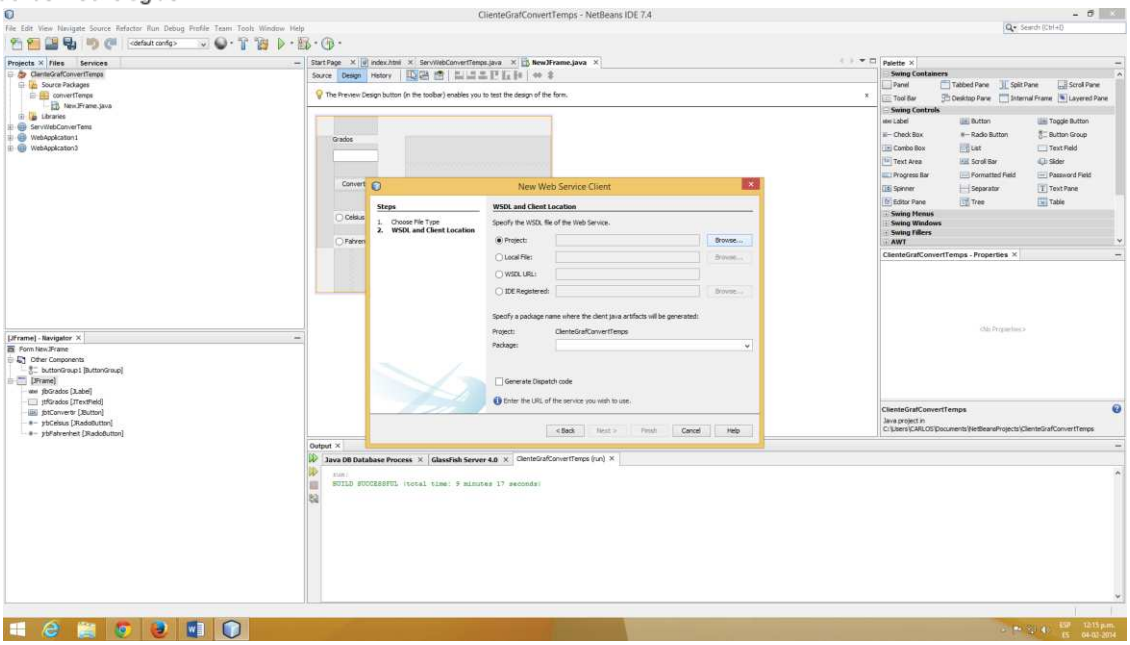
Entonces, crearemos un cliente de servicios Web para consumir un servicio Web. Obsérvese en la figura siguiente las opciones mostradas para ello; esto es, la forma en que un cliente de servicios Web consume un servicio Web depende de cómo el proveedor del servicio Web lo distribuye:

1. El proveedor puede publicar el archivo WSDL de un servicio Web en funcionamiento (desplegado en un servidor ejecutándose).
2. El proveedor distribuye en archivo WSDL; por lo tanto, podemos tenerlo disponible en nuestro sistema de archivos local.
3. El proveedor distribuye un proyecto NetBeans que define el servicio Web, para su despliegue en un contenedor.

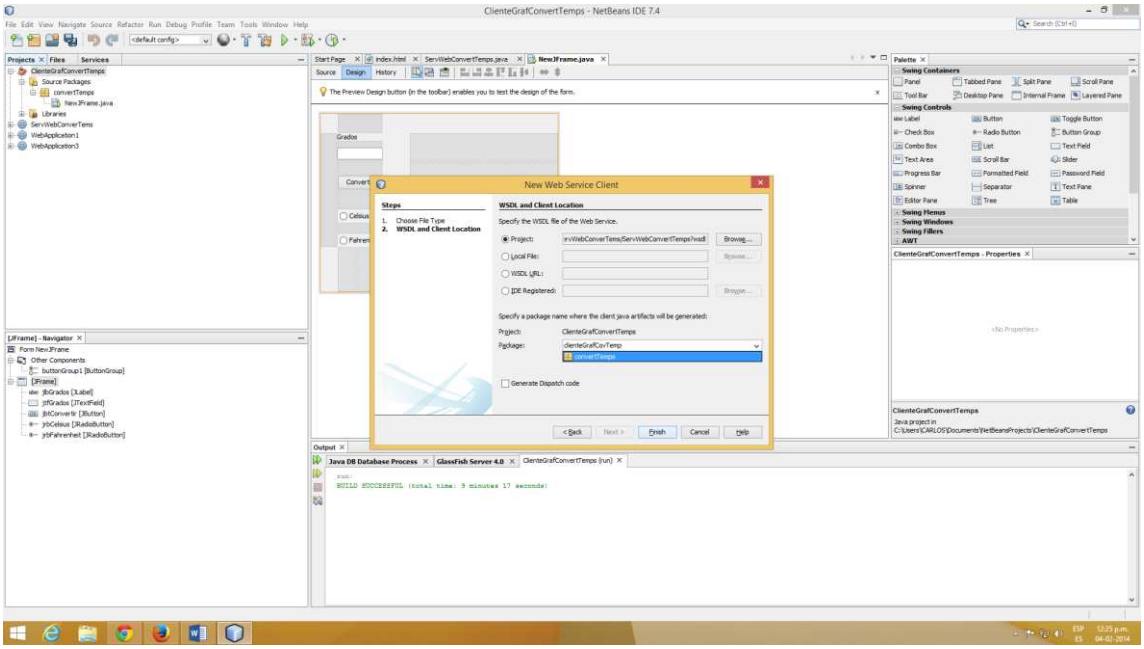
Según lo expuesto y siguiendo con el desarrollo de nuestra aplicación cliente, añada al proyecto un Web Service Client y especifique la localización del archivo WSDL, según muestran las dos figuras siguientes. En nuestro caso, hemos optado por la opción primera porque el servicio Web que queremos consumir lo tenemos desplegado en el servidor de aplicaciones GlassFish (debe estar ejecutándose).

Convirtiendo mi aplicación gráfica en un cliente del servicio web. Agregando una referencia del servicio web. Descubriendo el Web Service.

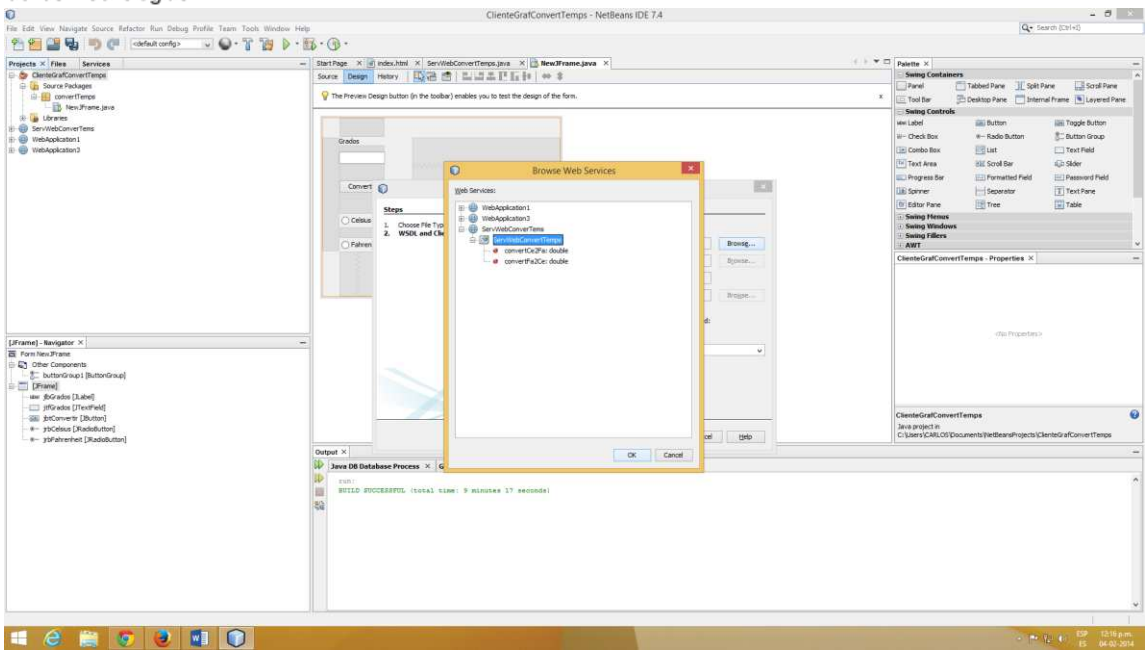




Seleccionamos “Project” y “browse” para ubicar al archivo WSDL.



Seleccionando el paquete.



Seleccionando el web service.

Una vez descubierto el servicio Web SWConverTemps, se agregó en el cliente Java de forma automática una referencia al servicio Web que desamos utilizar. Puede comprobarlo en el explorador de proyectos; observará que se ha añadido un nuevo nodo Web Service References:

El hecho de añadir la referencia Web al proyecto desencadenó cosas interesantes. En el nodo de referencias a los servicios Web del proyecto (Web Services References) se puede observar que se han añadido una serie de archivos (se obtienen a través de la utilidad wsimport y se pueden inspeccionar desde el explorador de archivos: panel Files) que dan lugar a lo que hemos denominado proxy, esto es, una representación local del servicio para interactuar con él. Entre todos estos archivos nos vamos a fijar en dos: SWConverTemps y SWConverTempsServices. El 1ro es la interfaz de Java correspondiente a la clase que implementa el servicio Web generada anteriormente; un objeto de este tipo es conocido también como service endpoint: representación local del servicio Web. Y el 2do es una clase derivada de la clase service del paquete javax.xml.ws que implementa una función (getSWConverTempsPort que invoca a getPort de Service) que proporciona el objeto service endpoint.

La generación automática de y la inclusión de estos ficheros por NetBeans hacen que la integración de un servicio Web en un cliente sea sencilla.

De forma predeterminada, el proxy utiliza SOAP sobre HTTP para comunicarse con el servicio Web XML. Esto es, un cliente y un servicio Web XML se comunican mediante mensajes SOAP, que encapsulan los parámetros in y out como XML. Afortunadamente, el proxy controla el trabajo de asignar parámetros a elementos XML y, después, de enviar el mensaje SOAP a través de la red.

Obtener acceso al servicio Web XML

Una vez agregada en el cliente la referencia al servicio Web XML, el siguiente paso es crear un objeto de la clase que define el servicio Web en el lado del cliente (service endpoint). Con este objeto podrá acceder a los métodos del servicio Web XML en el lado del servidor de la misma forma que con cualquier otro objeto, es decir, invocando a sus métodos. Cuando la aplicación cliente llama a esos métodos, es el proxy el que gestiona las comunicaciones entre la aplicación cliente y el servicio Web XML.

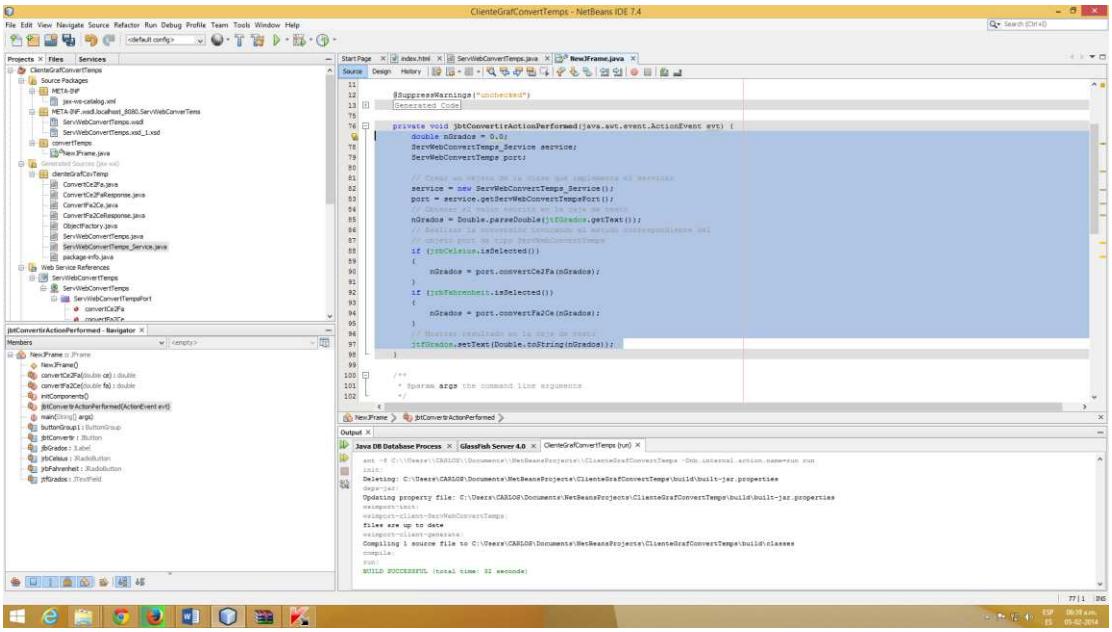
Por lo tanto, haga doble clic en el botón Convertir del formulario para crear el manejador de eventos que responderá al evento clic de ese botón.

A continuación, para acceder a una operación de las proporcionadas por el servicio Web, diríjase al manejador de eventos, en el panel que muestra el código fuente del cliente, y proceda de una de las dos formas siguientes:

- Expanda el nodo “Web Service References” hasta llegar al nodo que representa la operación. Utilizando el ratón, arrastre y suelte el nodo en el lugar donde desea que se añada el código.
- O bien, haga clic utilizando el botón secundario del ratón en el lugar donde desea que se añada el código, elija la orden “Call Web Service Operation” del menú contextual que se muestra y seleccione en el dialogo que se visualiza la operación del servicio Web a la que se desea acceder.

Después complete el manejador de eventos como se indica a continuación.

Obsérvese que, en 1er lugar, se crea un objeto de la clase que define el servicio y, a continuación, se toma el valor proporcionado en la caja de texto “jtfGrados” y se llama al método adecuado del servicio Web XML utilizando el objeto que encapsula el servicio. El resultado, valor devuelto por el servicio web XML, se muestra en la misma caja de texto.

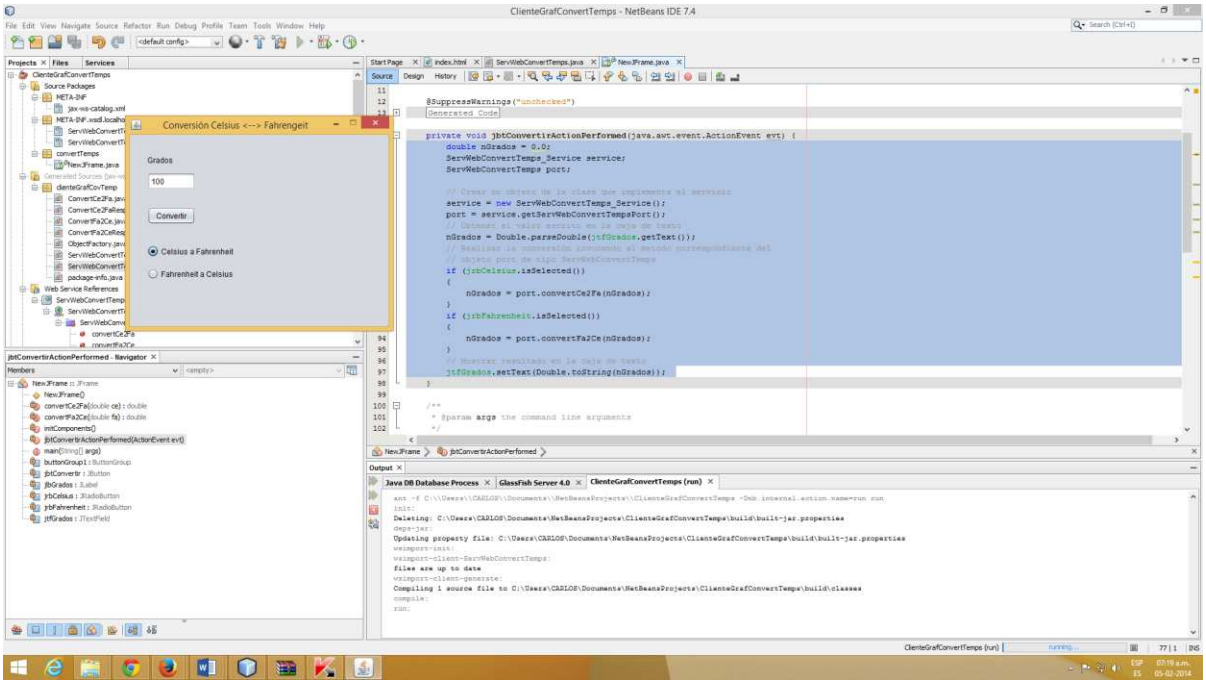


Abajo estamos usando el mismo código, pero, con excepciones.

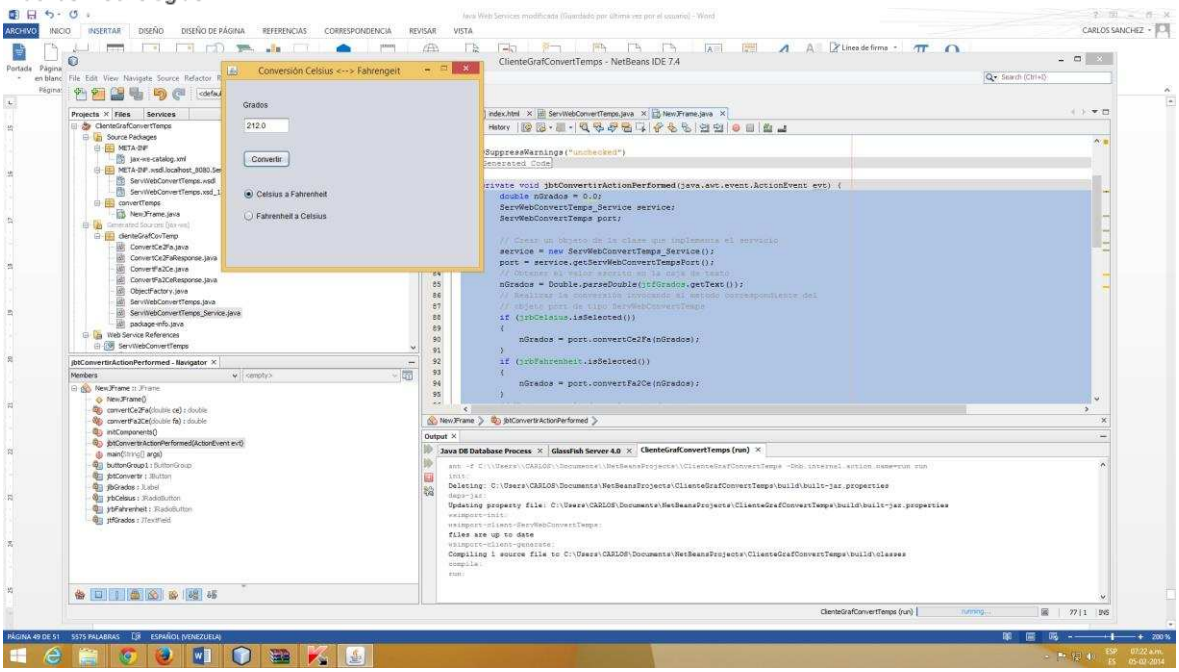
```
private void jbtConvertirActionPerformed(java.awt.event.ActionEvent evt)
{
  double nGrados = 0.0;
  SWConverTemp_Service service;
  SWConverTemps port;

  try
  {
    //Cear un objeto de la clase que implementa el servicio
    service = new SWConverTemps_Service();
    port = service.getSWConverTempsPort();
    //Obtener el valor escrito en la caja de texto
    nGrados = Double.parseDouble(jtfGrados.getText());
   }
   catch(NumberFormatException ex)
   {
     System.out.println("No se puede acceder al servicio\n);
     return;
    // Realizar la conversión invocando al método correspondiente del objeto port de tipo
    // SWConverTemps
    if(jrbCentAFahr.isSelected())
    {
       nGrados = port.convCentAFahr(nGrados);
    }
    if(jrbFahrACent.isSelected())
    {
       nGrados = port.convFahrACent(nGrados);
    }
    // Mostrar el resultado en la caja de texto
    jtfGrados.setText(Double.toString(nGrados));
}
```

Utilizando nuestra aplicación:



Luego de haber colocado el valor 100 y haber seleccionado "Celsius a Fahrengeit", presionamos "Convertir" y obtenemos:


Conversión Celsius <--> Fahrengeit
Grados
212.0
Convertir
Celsius a Fahrenheit
Fahrenheit a Celsius
ClienteGrafConvertTemps - NetBeans IDE 7.4