

5.5. Componentes web con acceso a datos (Servlet, JSP)



Índice

Objetivos	3
Descargar e instalar Apache Tomcat.....	4
Introducción.....	4
Descarga e instalación	5
Servlets	6
Introducción.....	6
Configuración del proyecto	6
Nuestro primer Java Servlet	15
Generar página de respuesta	18
Ejecución del servlet.....	20
Petición desde un formulario web	24
Ciclo de vida de un servlet.....	28
Acceso a datos desde un servlet	30
Servlet con acceso a base de datos XML.....	31
Java Server Pages (JSP).....	34
Introducción.....	34
Elementos básicos de los documentos JSP	34
Las variables predefinidas request y response	37
Despedida	39
Resumen.....	39

Objetivos

En esta lección perseguimos los siguientes objetivos:

- Desarrollar componentes de servidor (servlets o JSP) que acepten solicitudes de clientes desde sus navegadores web.
- Responder a las solicitudes de los clientes, generando una página HTML dinámica que se enviará como respuesta al servidor.
- Conocer el ciclo de vida de una aplicación web.

Descargar e instalar Apache Tomcat

Introducción

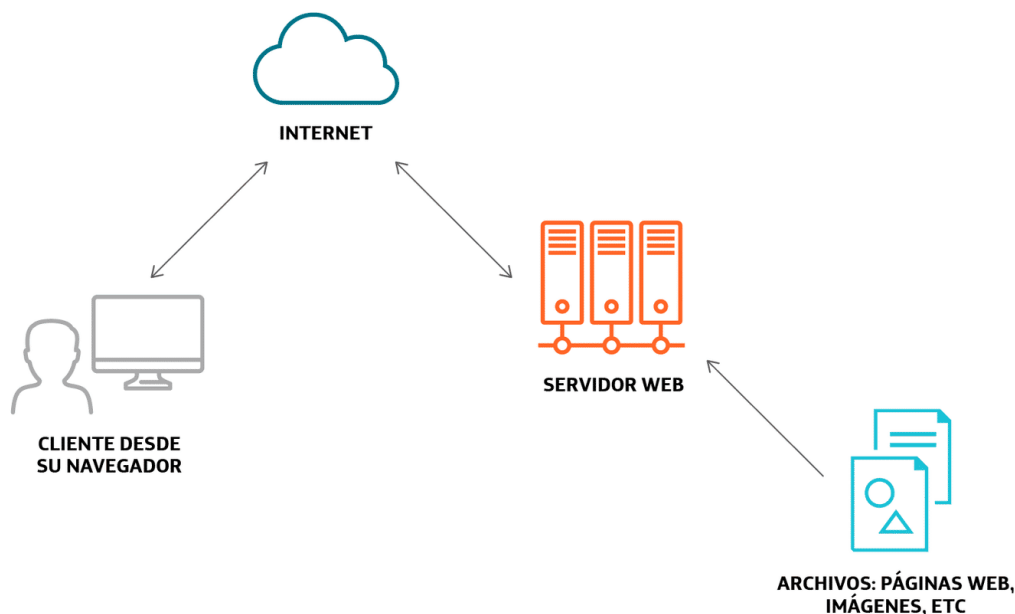
En esta lección, aprenderás a crear aplicaciones que siguen esta lógica:

- El cliente, desde su navegador, hace una petición a través de la URL.
- Un programa de servidor recibe la petición y procesa una respuesta. Aquí es donde debe entrar en juego una de las tecnologías anteriormente mencionadas (servlets, JSP, PHP, etc.). La respuesta finalmente generada tendrá formato HTML.
- El servidor envía la respuesta al cliente, que la visualiza en su navegador.

Pero, ¿para eso necesitamos tener un servidor web en nuestro equipo?

Sí, y por eso vamos a **descargar e instalar el servidor Apache Tomcat**. Así completaremos el ciclo petición-respuesta en nuestro equipo, que actuará como cliente y como servidor.

Recuerda que, para poner a prueba una aplicación web, por pequeña que sea, necesitarás los recursos que ves en la imagen de abajo: un cliente que realice una petición desde su navegador, Internet, un servidor web y el conjunto de recursos que forman el sitio web (páginas, imágenes, programas que forman el *back-end*, etc.).



Estás a punto de descargar e instalar el servidor Apache Tomcat.

¡Adelante!

Descarga e instalación

Puedes descargar Apache Tomcat gratuitamente desde su web oficial.

Accede desde esta URL: <http://tomcat.apache.org/>

Desde ahí, descargaremos la versión 9 ejecutable de Apache Tomcat como servicio web. De este modo se instalará como un servicio más de Windows, y podremos iniciarlo y detenerlo cuando sea necesario.

Un [servicio Windows](#) es un **programa que funciona en segundo plano con la finalidad de prestar algún servicio cuando es requerido**. Los servicios pueden iniciarse o detenerse.

La instalación de Apache Tomcat consta de varios pasos, por los que irás avanzando simplemente pulsando el botón "Next". Pero hay dos de ellos a los que merece la pena prestar más atención.

- **Donde se nos solicita la ubicación de la instalación de Java:** Apache Tomcat requiere que esté instalado Java. En la gran mayoría de los casos el instalador detecta la ubicación de la instalación de Java sin mayor problema, pero en algunas ocasiones puede ser necesario especificarla.
- **Donde aparecen los valores de configuración del servidor:** presta atención al valor de "HTTP/1.1 Connector Port", que será 8080. Recuerda que una petición a un servidor se realiza a través del nombre del servidor y un puerto. Pues bien, el nombre del servidor, al tratarse de nuestro propio equipo, será **localhost** y el puerto el **8080**, que es el valor que aparece por defecto.

El siguiente vídeo muestra cómo descargar e instalar Apache Tomcat en tu equipo.

<https://vimeo.com/telefonicaed/review/277073188/38719569a1>

Servlets

Introducción

Los **servlets** son programas que se ejecutan en un servidor web, en una capa intermedia entre una petición proveniente de un navegador web, o cliente, y las bases de datos o aplicaciones del servidor HTTP.

Un servlet puede realizar las **siguientes tareas**:

Leer los datos enviados por el cliente

Que por lo general han sido introducidos en un formulario HTML.

Buscar información respecto a la petición del cliente

Como el *host*, facultades del navegador, las *cookies*, etc.

Generar los resultados

Que se enviarán al cliente, proceso que podría requerir consultar en base de datos o ejecutar otros procesos de la lógica de negocio.

Dar formato a los resultados

Generando dinámicamente un documento, que en la mayoría de los casos tendrá formato HTML.

Establecer los parámetros adecuados para la respuesta HTTP

Esto se traduce en decirle al navegador el tipo de documento que será devuelto (por ejemplo, HTML), entre otras cosas.

Devolver el documento al cliente

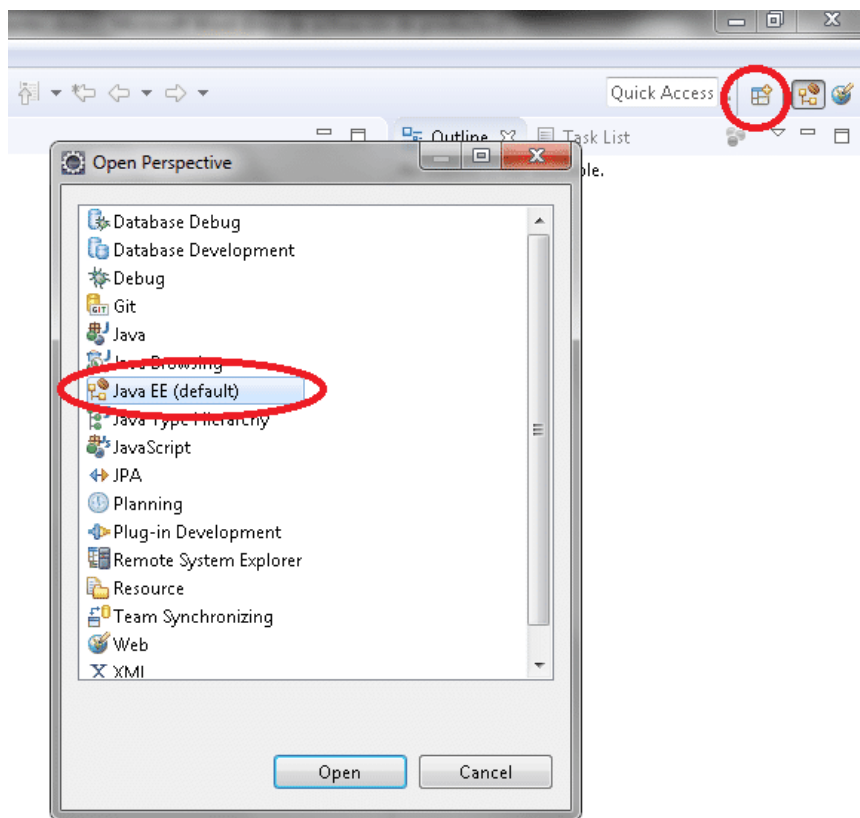
Este documento podría ser enviado en un formato de texto (HTML), binario (imágenes GIF), o incluso en cualquier otro formato comprimido.

Configuración del proyecto

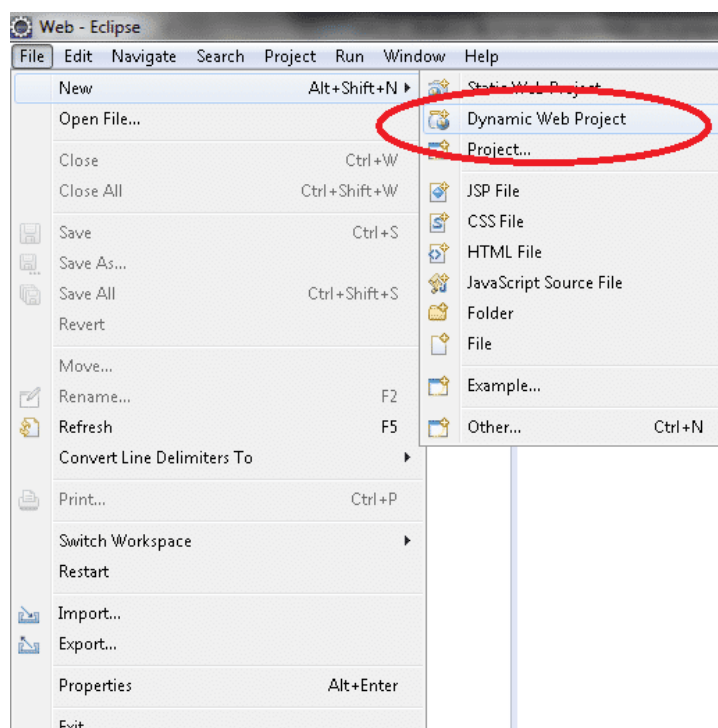
Antes de crear nuestro primer *servlet* necesitamos un **proyecto Eclipse configurado para que sirva como proyecto web dinámico, capaz de ejecutarse en un servidor web Apache Tomcat**.

Sigue estos pasos:

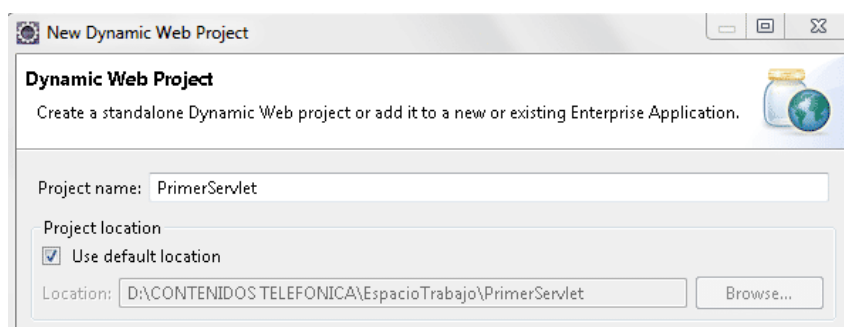
1. Cambia a la perspectiva Java EE, si es que no estás en ella. Es la más adecuada para la tarea que nos ocupa. Pulsa el botón **Open Perspective** y selecciona **Java EE** en el cuadro de diálogo que te aparecerá.



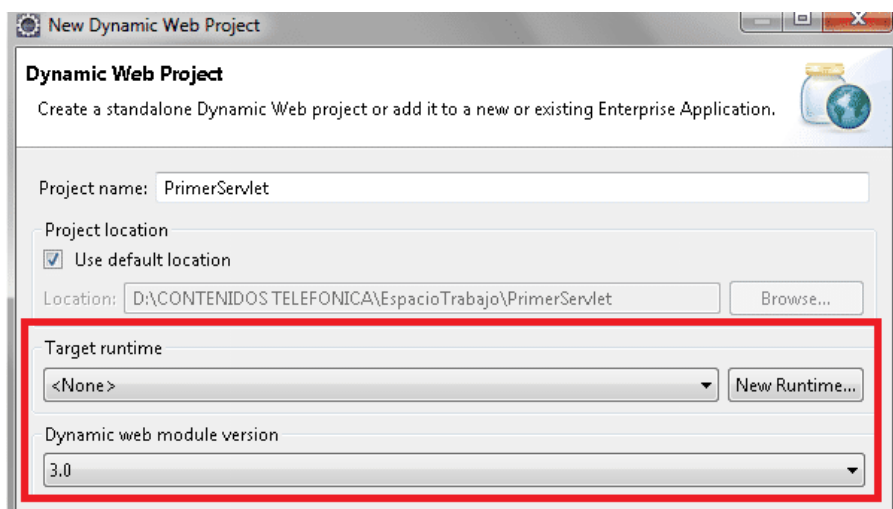
2. Selecciona en la barra de menú **File / New / Dynamic Web Project**.



3. En el cuadro de diálogo *Dynamic Web Project* debes comenzar por **escribir el nombre del nuevo proyecto**. Observa que el cuadro *Location* te está mostrando la ubicación en la que se creará el proyecto, que será a partir de la ubicación del espacio de trabajo de Eclipse.

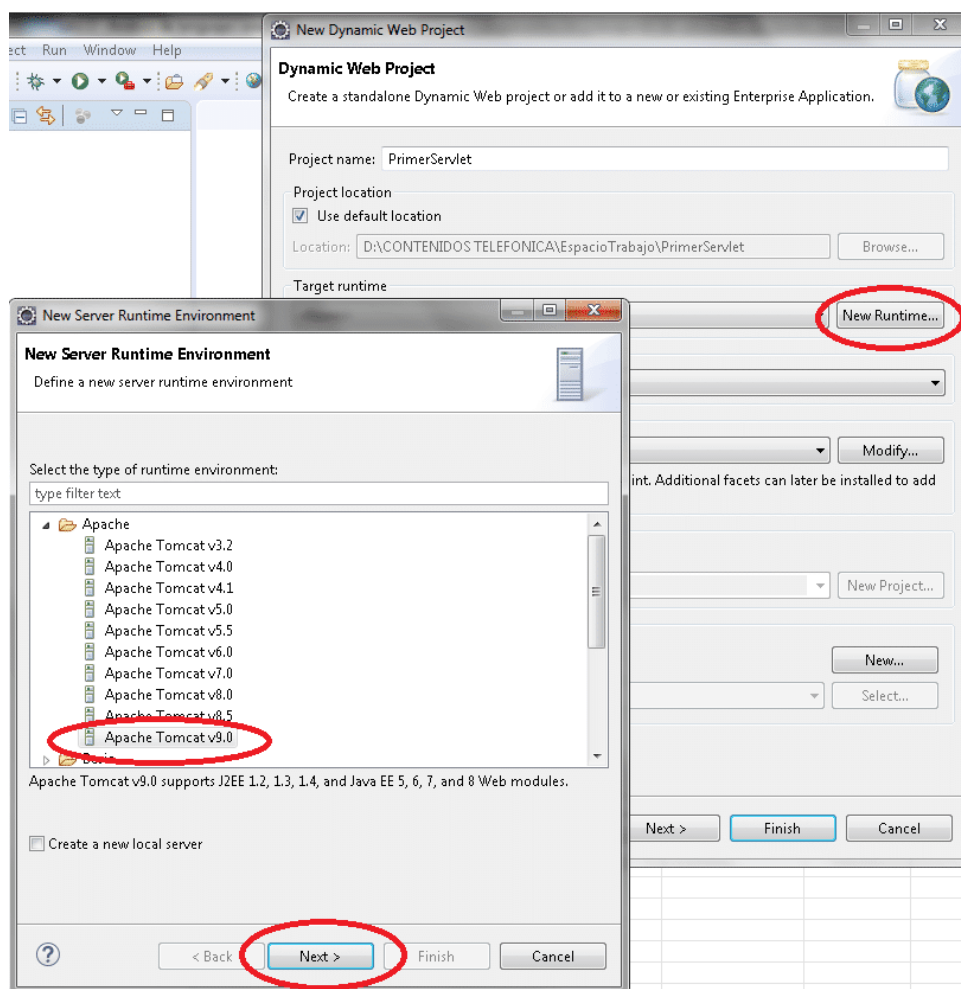


4. Ahora, vamos a prestar atención a las configuraciones del **Target Runtime** y **Web Module Version**.



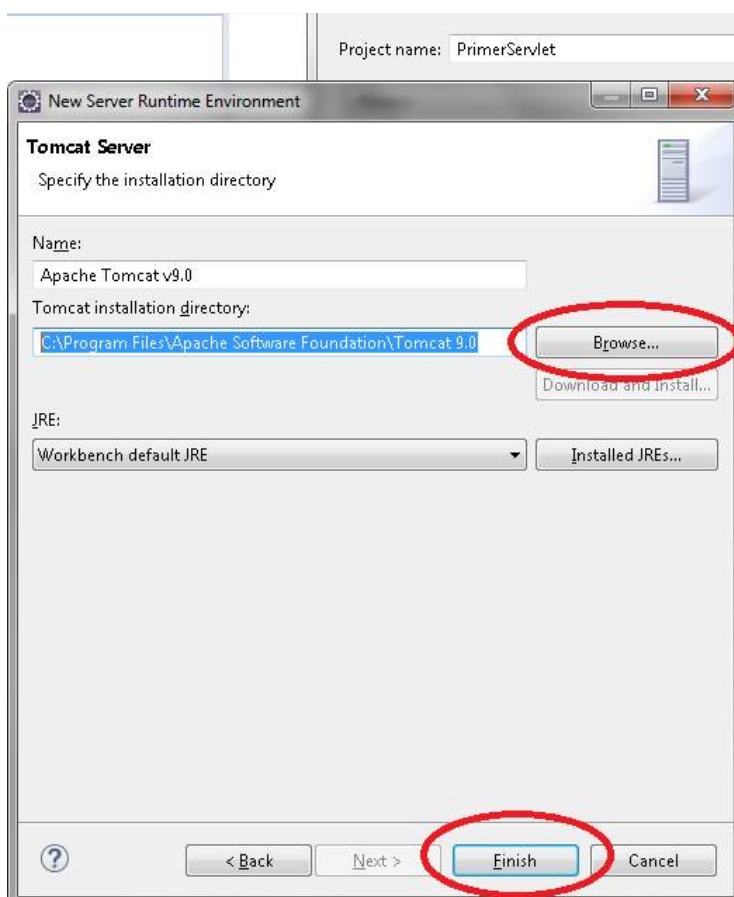
- El **target runtime** hace referencia al responsable de la ejecución del código Java que generaremos para este proyecto. Puesto que en esta ocasión vamos a generar código de servidor o *back-end*, el responsable de la ejecución será el servidor, es decir, Apache Tomcat.
- El **Dynamic web module versión** hace referencia a cómo se comportará Eclipse a la hora de generar código automático. Nosotros no tendremos que escribir todo el código de nuestros servlets, sino que gran parte del código será generado automáticamente por Eclipse, y dicho código variará ligeramente en función del número de versión.

Al tratarse de nuestro primer proyecto web dinámico, no tenemos nada en la lista desplegable **Target runtime**. **Pulsa el botón *New Runtime*** para obtener el cuadro de diálogo *New Server Runtime Environment*.



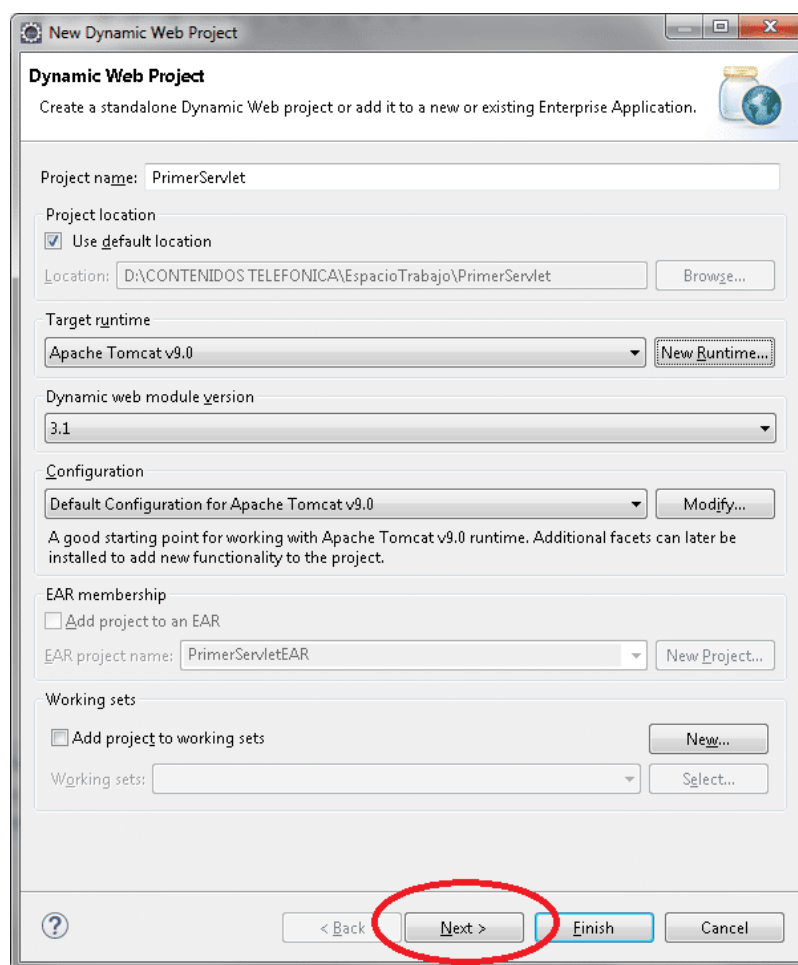
5. Selecciona **Apache Tomcat v9.0**. Si trabajas con una versión antigua de Eclipse, podría ser que no reconozca que existe la versión 9. En ese caso, no te quedaría más remedio que descargar y descomprimir una versión de Eclipse más moderna. Una vez seleccionada la versión 9 de Apache Tomcat haz clic en el botón **Next**.

6. Ahora, Eclipse te está pidiendo que especifiques la ubicación de la instalación de Apache Tomcat y debes pulsar el botón **Browse ...** para buscar dicha ubicación. Si utilizas Windows, lo más habitual es que se encuentre en "C:\Program Files\Apache Software Foundation\Tomcat 9.0".



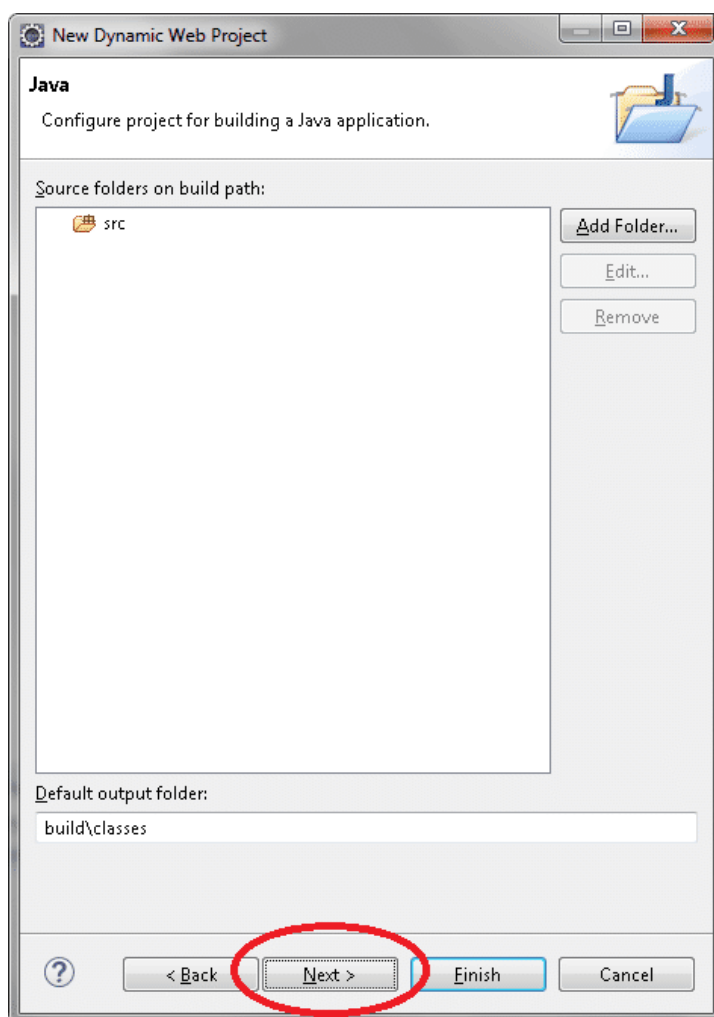
Una vez especificada la ubicación, pulsa el botón **Finish**.

7. Ahora, el cuadro de diálogo *New Dynamic Web Project* muestra "Apache Tomcat v9.0" como *Target Runtime*.



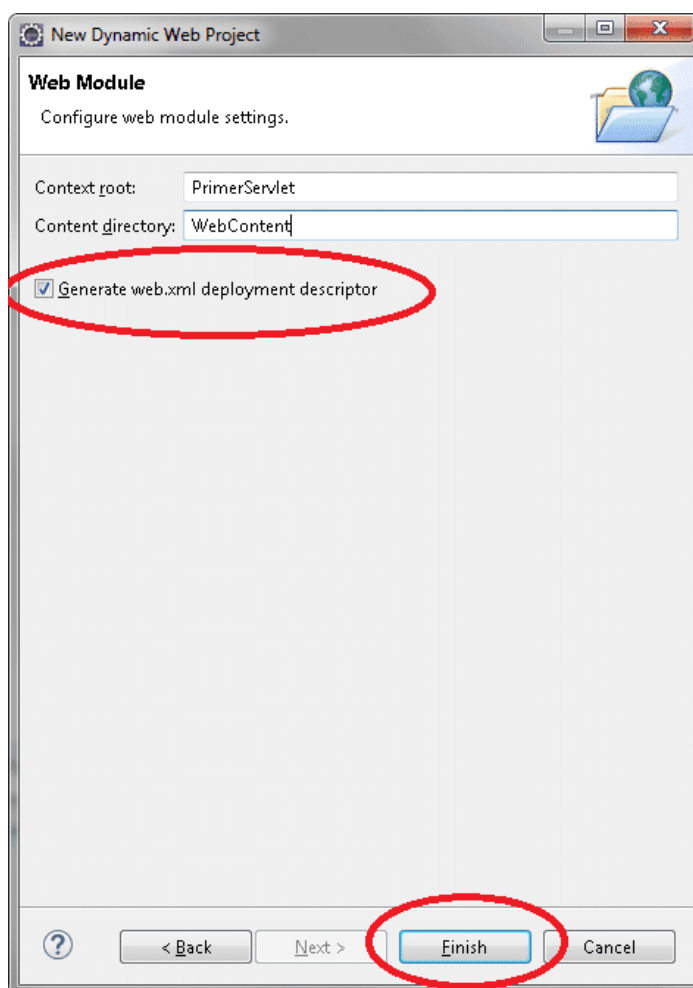
Pulsa el botón **Next** para ir al siguiente paso.

8. En esta ocasión, Eclipse simplemente te está informando sobre la ubicación del código fuente de los programas Java, que es en la carpeta **src**; no tienes que hacer nada especial.



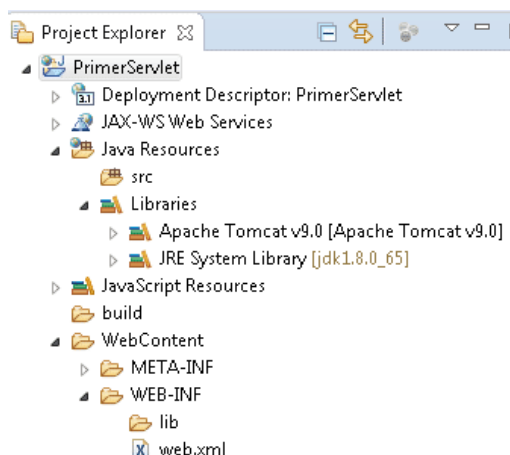
Sólo pulsa el botón **Next** para avanzar al siguiente paso.

9. Ahora, es importante que actives la casilla de verificación **Generate web.xml deployment descriptor**. De esta forma, Eclipse generará por nosotros el archivo web.xml, que contendrá valores de configuración del sitio web.



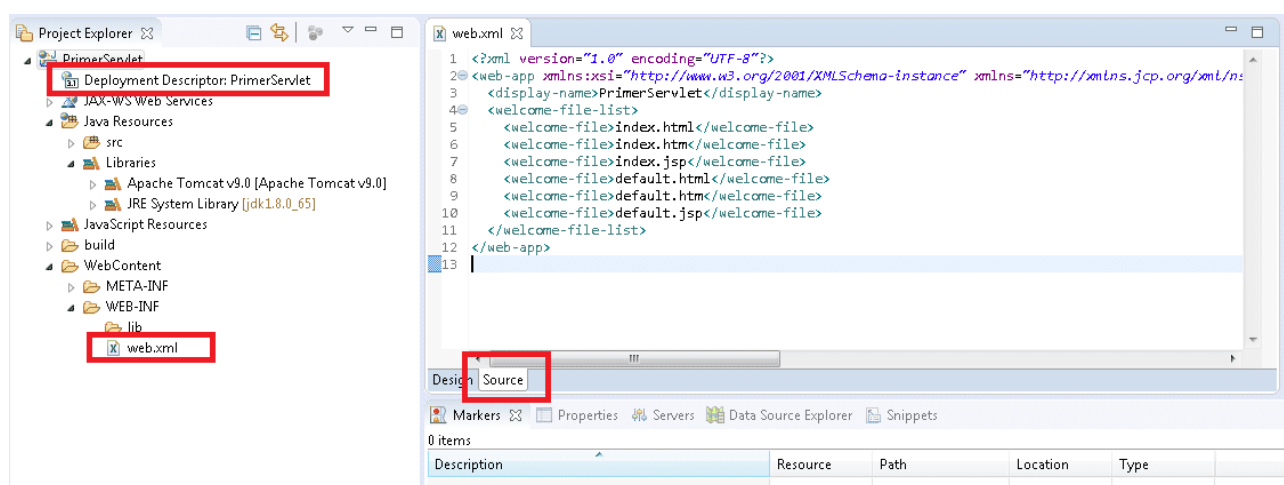
Por último, haz clic en el botón **Finish** y tendrás creado un proyecto Java, configurado para crear una aplicación web dinámica que se ejecutará en Apache Tomcat.

10. Fíjate ahora en el explorador de proyectos para familiarizarte con la estructura que tiene un proyecto web dinámico. Expande las carpetas que necesites hasta que el proyecto aparezca como en la siguiente imagen:



5.5. Componentes web con acceso a datos (Servlet, JSP)

- La carpeta **Java Resources**, como su nombre indica, contiene todos los recursos Java. Está compuesta por las subcarpetas **src**, donde se guardarán tus códigos fuentes, y **Libraries**, donde se encuentran las librerías necesarias. En este caso, además de las librerías del JRE, se encuentran otras librerías suministradas por el servidor Apache Tomcat.
- La carpeta **Web Content** es donde se irán ubicando todos los archivos del sitio web, que pueden ser ficheros HTML, imágenes, archivos CSS, etc. como sabes por lecciones anteriores.
- Observa que dentro de **Web Content** hay una carpeta llamada **WEB-INF** que contiene el archivo **web.xml**. Como hemos mencionado en otras ocasiones, este archivo contiene configuración del sitio web que será relevante para el servidor. Ábrelo para ver su contenido, pero tendrás que situarte en la pestaña **Source** para verlo igual que en la siguiente imagen:



Otra forma de abrir el archivo web.xml es haciendo clic en el enlace *Deployment Descriptor: PrimerServlet*. Por ahora solamente vamos a analizar esta línea:

`<welcome-file>index.html</welcome-file>`

Estamos indicando el nombre de la página de inicio, o *home page*, de nuestra web. El archivo web.xml, generado automáticamente, ofrece varias alternativas de nombres. El servidor HTTP actúa de la siguiente manera: si un cliente accede desde su navegador a la dirección `www.unaurl.com`, está indicando el nombre de dominio, pero no el recurso al que desea acceder. En este caso, si el proyecto alojado en dicho dominio tiene un archivo web.xml, como el de la anterior imagen, devolvería al cliente el contenido del archivo `index.html` en caso de que exista, y, si no existiese, devolvería el contenido del archivo `index.htm`. Y así sucesivamente.

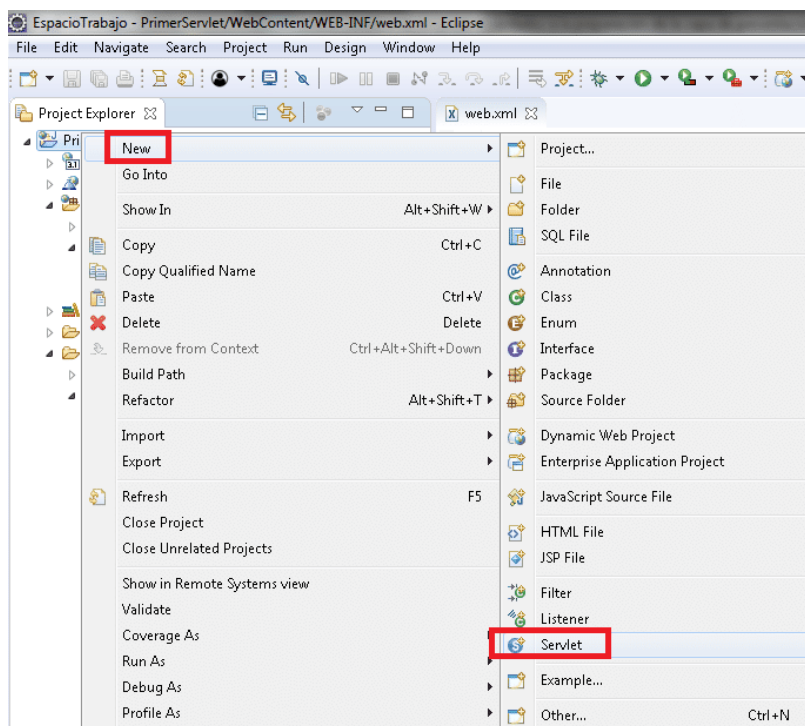
Ya tienes el proyecto perfectamente configurado. Sólo falta crear el *servlet* para ponerlo a funcionar.

Nuestro primer Java Servlet

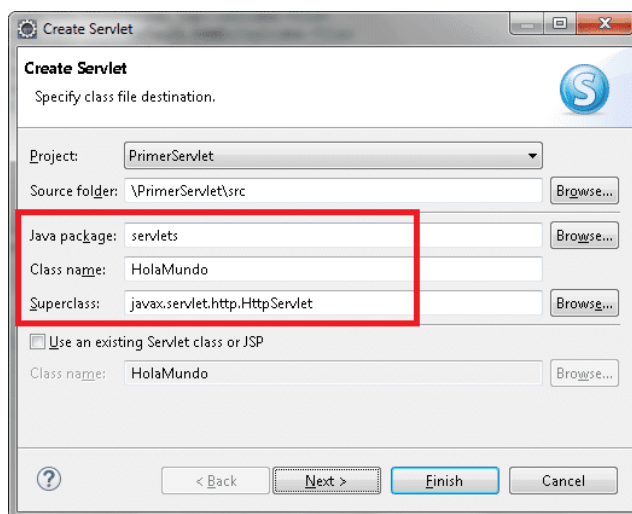
Ha llegado el momento de crear el primer *servlet* que nos sirva para atender peticiones desde un navegador web. Lo llamaremos **HolaMundo**.

Sigue estos pasos:

1. Con el puntero del ratón sobre el nombre del proyecto, haz **clic derecho** y selecciona **New / Servlet** en el menú contextual.



2. Debes tener abierto el cuadro de diálogo **Create Servlet**, en el que necesitarás establecer algunos valores. Observa la imagen:



Los valores de configuración del *servlet* que te interesan en este momento son los siguientes:

Java package

Paquete Java donde se ubicará el *servlet*. Al fin y al cabo, un *servlet* es una clase Java que podrá estar ubicada en un paquete. **Escribe *servlets* como nombre de paquete.**

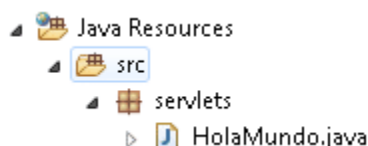
Class name

Será el nombre de la clase que representará al *servlet*. Escribe aquí ***HolaMundo***.

Superclass

Aunque aquí no tienes que tocar nada, es interesante que prestes atención. Un *servlet* es una clase Java que deriva de la superclase ***HttpServlet***, que está ubicada en el paquete o librería ***javax.servlet.http***.

3. Pulsa ***Finish*** para terminar y despliega la carpeta *Java Resources* de tu proyecto. Gran parte de tu *servlet* ya estará creado. En la carpeta *src* se ha creado la estructura, conforme a la configuración que especificaste: **una clase Java llamada *HolaMundo* en un paquete llamado *servlets*.**



Tu nuevo *servlet* tendrá el siguiente aspecto:

```
package servlets;

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
    response.getWriter().append("Served at:
").append(request.getContextPath());
}

/**
 * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse
 response)
 */
protected void doPost(HttpServletRequest request, HttpServletResponse
 response) throws ServletException, IOException {
    // TODO Auto-generated method stub
    doGet(request, response);
}
}
```

A continuación, vamos a analizar el código que ha generado Eclipse de manera automática.

```
package servlets;
```

La nueva clase Java estará ubicada en el paquete *servlets*, tal como indicamos en el momento de su creación.

```
import java.io.IOException;
```

Ya que el objetivo de cualquier *servlet* es atender peticiones de clientes y responder a dichas peticiones con una página de respuesta, es fácil predecir que, en estas tareas, entran en juego operaciones de entrada y salida de datos para la comunicación entre cliente y servidor. Estas operaciones de entrada/salida podrían ocasionar excepciones que podrían ser capturadas por un objeto de la clase *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;
```

Todas las clases que forman parte de la jerarquía de clases relacionadas con los *servlets* se encuentran en la librería *javax.servlet*. Poco a poco irás aprendiendo la función de todas estas clases.

```
@WebServlet("/HolaMundo")
public class HolaMundo extends HttpServlet {
}
```

Lo que hace que nuestra clase pueda considerarse un *servlet* es que extiende la **superclase *HttpServlet***, lo que la provee de la funcionalidad necesaria para poder considerarse un *servlet*.

La **anotación *@WebServlet*** se utiliza para registrar el *servlet* en el servidor. En el *argumento*, especificamos el nombre que utilizará el cliente para hacer referencia al *servlet* desde el navegador. Para acceder al *servlet* desde el navegador, utilizaríamos el siguiente formato:

protocolo://servidor:puerto/proyecto/Servlet

Que para nuestro ejemplo sería:

http://localhost:8080/PrimerServlet/HolaMundo

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {

}

protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {

    doGet(request, response);
}
```

Recuerda que **la misión principal de un servlet es atender la petición de un cliente desde su navegador web** a través del protocolo HTTP y esta petición suele llegar en la mayoría de los casos desde la URL.

Pues bien, **cada servlet está siempre en ejecución en el servidor a la espera de atender peticiones**. Cada vez que recibe una petición, ésta es atendida en los métodos ***doGet()*** o ***doPost()***. El método ***doGet()*** atiende peticiones de tipo GET, y ***doPost()*** atiende peticiones de tipo POST.

Más adelante aprenderás la diferencia entre una petición GET y una petición POST. Pero en nuestro ejemplo trataremos de la misma forma una petición GET que una petición POST, ya que, si te fijas, desde el método ***doPost()*** invocamos al método ***doGet()***, donde falta completar el código para generar la página de respuesta que se enviará al cliente.

Generar página de respuesta

En este apartado **completaremos el código del método *doGet()* de nuestro *servlet*** para que responda a la petición del cliente.

Comienza por sustituir el método *doGet()* actual por el siguiente:

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {

    response.setContentType("text/html");
    PrintWriter flujoEscritura=response.getWriter();

    flujoEscritura.append("<!DOCTYPE html>");
    flujoEscritura.append("<html><head><meta charset='UTF-8'>");
    flujoEscritura.append("<title>Página dinámica</title>");
    flujoEscritura.append("</head><body>");
    flujoEscritura.append("<h1>Hola Mundo</h1>");
    flujoEscritura.append("</body></html>");
    flujoEscritura.close();

}
```

Ahora, vamos a analizar detenidamente la implementación del método:

1.

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
```

En primer lugar, el método *doGet()* recibe dos argumentos: ***request*** de tipo *HttpServletRequest* y ***response*** de tipo *HttpServletResponse*.

El objeto ***request*** hace referencia a la petición realizada por el usuario. A través de este objeto podemos obtener información sobre el cliente que realizó la solicitud: navegador, parámetros de entrada, etc.

El objeto ***response*** hace referencia a la respuesta que enviará el servidor y nos servirá para preparar la página HTML de respuesta.

```
response.setContentType("text/html");
```

Con esta línea estamos indicando el formato de la página de respuesta que vamos a enviar al cliente. El texto especificado en el argumento es lo que se denomina un tipo [mime](#).

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

Enviar la página de resultado al cliente implica una operación de entrada y salida de datos, y eso requiere del uso de un **flujo o stream**. Toda información que se transmite entre ordenadores fluye desde una entrada hacia una salida, y esta transmisión ha de ser gestionada por un objeto especial que represente dicho flujo o *stream*. Será el objeto *response* quien nos provea de dicho flujo, ejecutando el método ***getWriter()***, que retornará un objeto ***PrintWriter***.

```
flujoEscritura.append("<!DOCTYPE html>");  
flujoEscritura.append("<html><head><meta charset='UTF-8'>");  
flujoEscritura.append("<title>Página dinámica</title>");  
flujoEscritura.append("</head><body>");  
flujoEscritura.append("<h1>Hola Mundo</h1>");  
flujoEscritura.append("</body></html>");
```

A partir de aquí, la tarea consiste en utilizar el método ***append(...)*** para ir enviando cadenas de texto que formarán un documento HTML completo.

```
flujoEscritura.close();
```

Para finalizar, cerramos el flujo de datos.

Sigue adelante para ejecutar tu primer *servlet*.
¡Ya queda muy poco!

Ejecución del servlet

En este apartado podrás ver el resultado final de la ejecución del *servlet*.

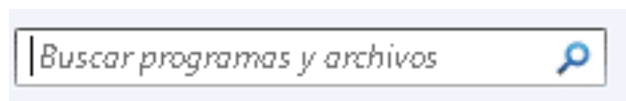
Recuerda que nuestro *servlet* será ejecutado en el servidor, es decir, será responsabilidad de Apache Tomcat ejecutarlo.

En primer lugar, debes tener en cuenta que Apache Tomcat actúa como servicio y, en consecuencia, dicho servicio puede iniciarse o detenerse. Pues bien, Eclipse se encargará de iniciar y detener el servicio cuando sea necesario, pero requiere que no esté iniciado ya desde fuera. Si trabajas con Windows, puedes comprobar si Apache Tomcat está iniciado abriendo la herramienta *Monitor Tomcat*.

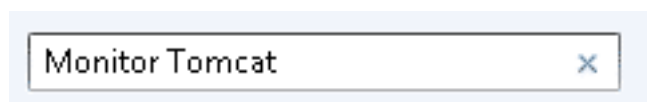
Sigue los pasos que te indicamos para ejecutar el servlet. Lo primero, será detener Apache Tomcat.

1. Detener Apache Tomcat

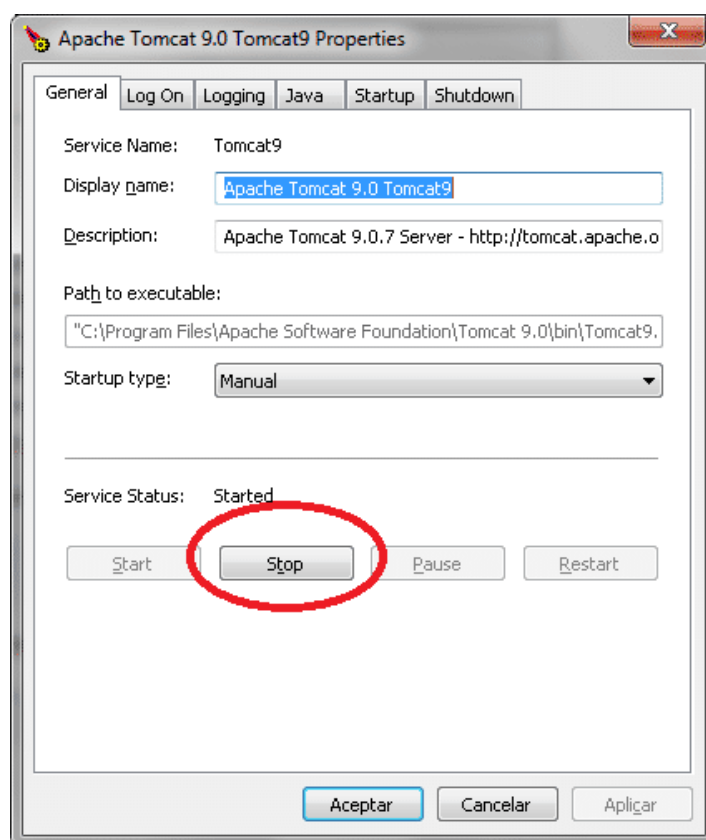
Utiliza el cuadro de texto de ***Buscar programas y archivos***.



Escribe ***Monitor Tomcat*** y pulsa enter.

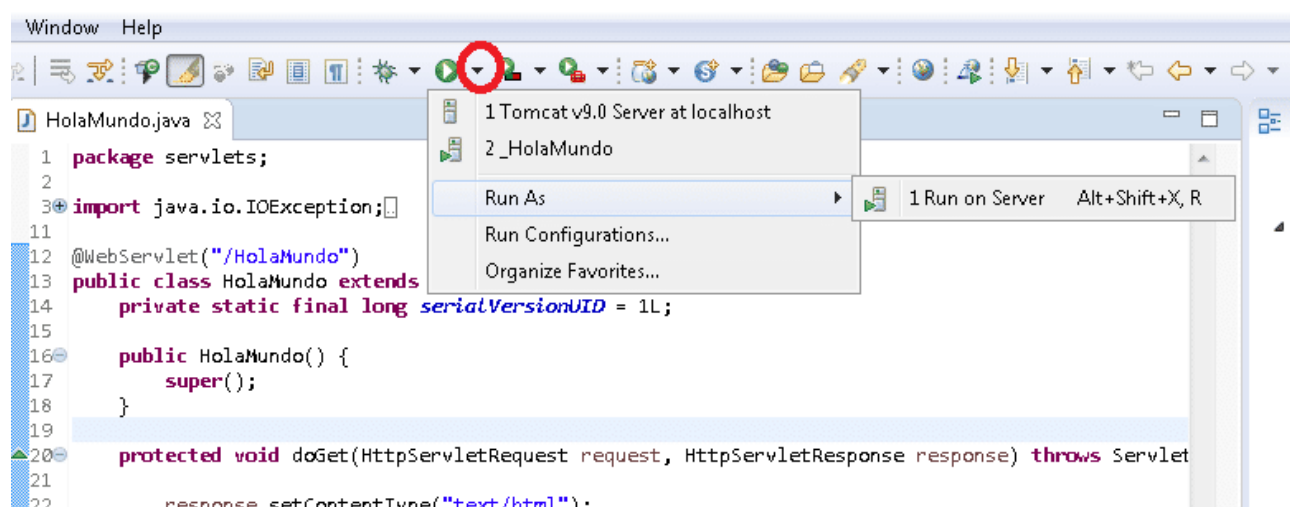


Ahora, pulsa el botón **Stop** del cuadro de diálogo *Apache Tomcat 9.0 Properties*.

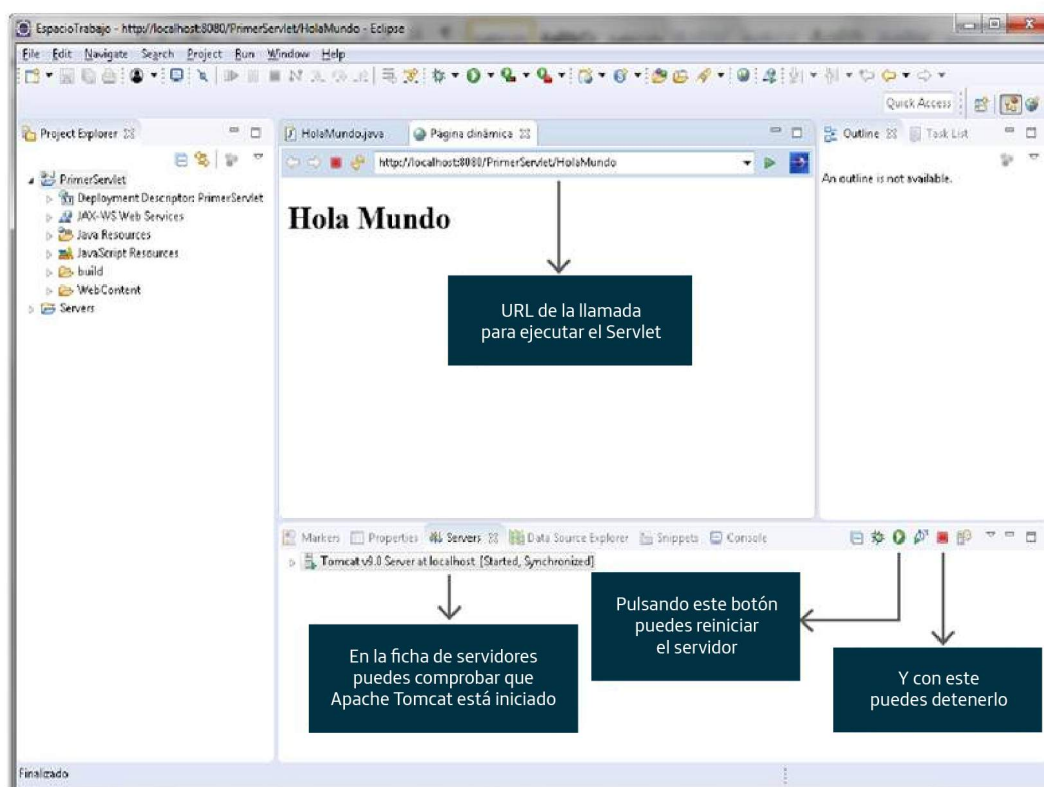


2. Ejecutar el proyecto

Haz clic en el icono del triángulo negro, en el botón de *ejecutar*, y selecciona **Run As / Run on Server** en el menú contextual.

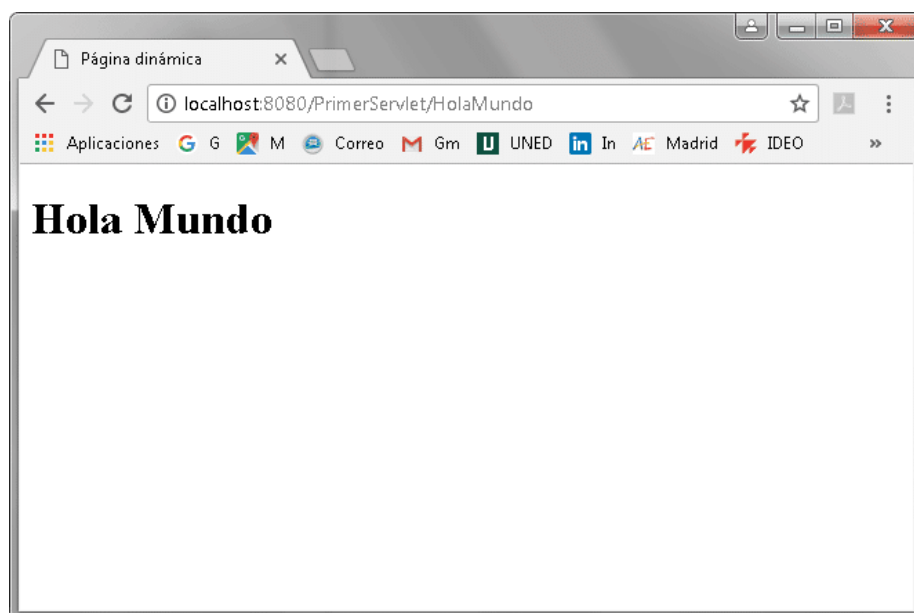


Si has seguido bien los pasos, Eclipse te estará mostrando el resultado en el explorador que tiene integrado.



Desde Eclipse puedes interactuar con el servidor Apache mediante la ficha Servers: puedes comprobar que está iniciado, puedes detenerlo, reiniciarlo, etc.

El *servlet* ha quedado registrado en el servidor y está preparado para recibir solicitudes de múltiples clientes desde sus navegadores. Incluso puedes probar a escribir la URL desde Google Chrome o cualquier otro navegador.



Ejecución desde Google Chrome.

Si ahora quisieras modificar tu *servlet* para que, por ejemplo, escriba varias veces “Hola Mundo” en varios tamaños, sería necesario detener el servidor y volver a iniciarlo para que el nuevo *servlet* con las modificaciones quede registrado correctamente.

Modifica de nuevo el método *doGet()* de la siguiente manera:

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {

    response.setContentType("text/html");
    PrintWriter flujoEscritura=response.getWriter();

    flujoEscritura.append("<!DOCTYPE html>");
    flujoEscritura.append("<html><head><meta charset='UTF-8'>");
    flujoEscritura.append("<title>Página dinámica</title>");
    flujoEscritura.append("</head><body>");
    for (int i=1; i<7; i++) {
        flujoEscritura.append("<h"+i+">Hola Mundo</h"+i+">");
    }
    flujoEscritura.append("</body></html>");
    flujoEscritura.close();

}
```

Una vez que hayas terminado los cambios, haz clic en el botón **Stop Server**.



Vuelve a ejecutar el *servlet* de la misma forma que lo hiciste anteriormente y el resultado será:

Hola Mundo

Hola Mundo

Hola Mundo

Hola Mundo

Hola Mundo

Hola Mundo

Petición desde un formulario web

Vamos a completar el proyecto anterior creando el archivo *index.html*, que incluirá un enlace al *servlet* anterior y un formulario que realizará la petición a un segundo *servlet*.

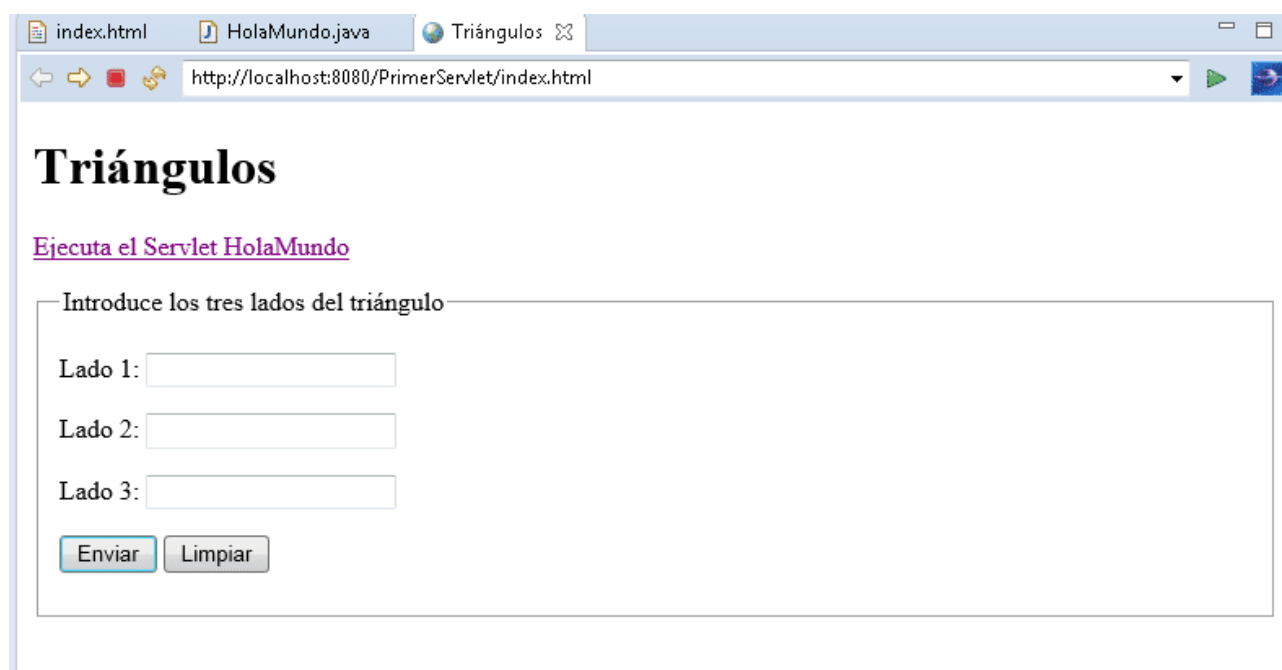
El archivo *index.html* incluirá los siguientes elementos:

- Un vínculo, que realizará la petición al *servlet* *HolaMundo*.
- Un formulario, que permitirá al usuario introducir los tres lados de un triángulo. Al enviar dicho formulario se ejecutará un *servlet*, que recibirá los tres parámetros y generará una página de respuesta.

Recuerda que debes hacer clic derecho sobre el nombre del proyecto y seleccionar **New / HTML file** en el menú contextual. Luego, puedes simplemente copiar y pegar este código:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Triángulos</title>
</head>
<body>
<h1>Triángulos</h1>
<p>
    <a href="HolaMundo">Ejecuta el Servlet HolaMundo</a>
</p>
<form action="Triang" method="get">
<fieldset>
<legend>Introduce los tres lados del triángulo</legend>
    <p>
        <label for="lado1">Lado 1:</label>
        <input type="number" name="lado1" />
    </p>
    <p>
        <label for="lado2">Lado 2:</label>
        <input type="number" name="lado2" />
    </p>
    <p>
        <label for="lado3">Lado 3:</label>
        <input type="number" name="lado3" />
    </p>
    <p>
        <input type="submit" value="Enviar" />
        <input type="reset" value="Limpiar" />
    </p>
</fieldset>
</form>
</body>
</html>
```

Tu nuevo documento HTML se visualizará de la siguiente manera:



The screenshot shows a web browser window with the address bar displaying `http://localhost:8080/PrimerServlet/index.html`. The page has a title bar with tabs for `index.html`, `HolaMundo.java`, and `Triángulos`. The main content area features a heading

Triángulos

 followed by a subheading

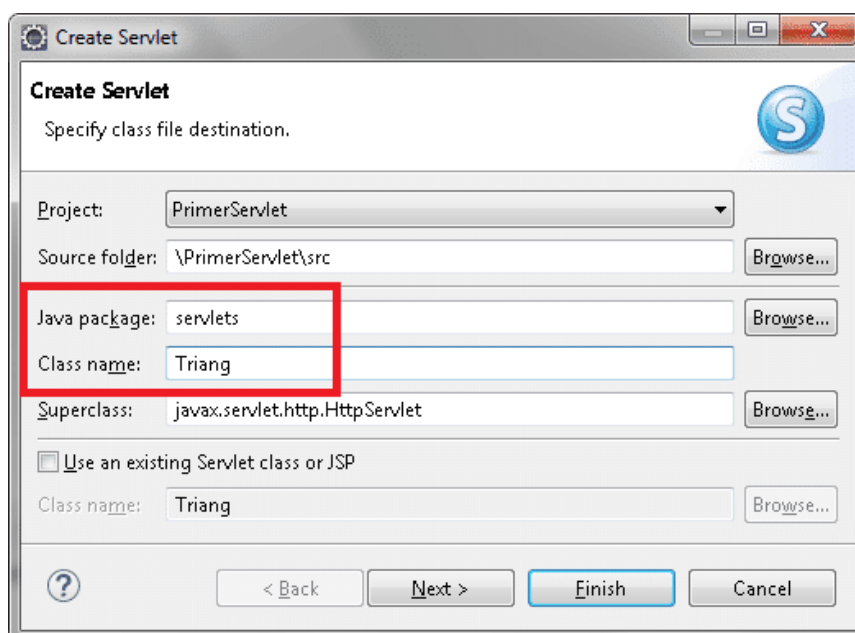
Ejecuta el Servlet HolaMundo

. Below this is a form titled 'Introduce los tres lados del triángulo' containing three input fields labeled 'Lado 1:', 'Lado 2:', and 'Lado 3:'. At the bottom of the form are two buttons: 'Enviar' and 'Limpiar'.

El vínculo ya está preparado para ejecutar el anterior *servlet*.

¡Ahora te toca crear el segundo!

Crea tu *servlet* igual que lo hiciste la vez anterior. Recuerda crearlo dentro del paquete *servlets* y llamarlo ***Triang***, tal como nos hemos referido a él en el documento HTML.



The screenshot shows the 'Create Servlet' dialog box. The 'Project' field is set to 'PrimerServlet'. The 'Source folder' is '\PrimerServlet\src'. The 'Java package' field is 'servlets', and the 'Class name' field is 'Triang'. The 'Superclass' is 'javax.servlet.http.HttpServlet'. The 'Use an existing Servlet class or JSP' checkbox is unchecked. The 'Class name' field at the bottom is also 'Triang'. The 'Finish' button is highlighted.

Puedes copiar y pegar el código desde aquí:

```
package servlets;

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;

@WebServlet("/Triang")
public class Triang extends HttpServlet {
    private static final long serialVersionUID = 1L;

    public Triang() {
        super();
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        int lado1, lado2, lado3;

        response.setContentType("text/html");
        PrintWriter flujoEscritura=response.getWriter();

        flujoEscritura.append("<!DOCTYPE html>");
        flujoEscritura.append("<html><head><meta charset='UTF-8'>");
        flujoEscritura.append("<title>Página dinámica</title>");
        flujoEscritura.append("</head><body>");

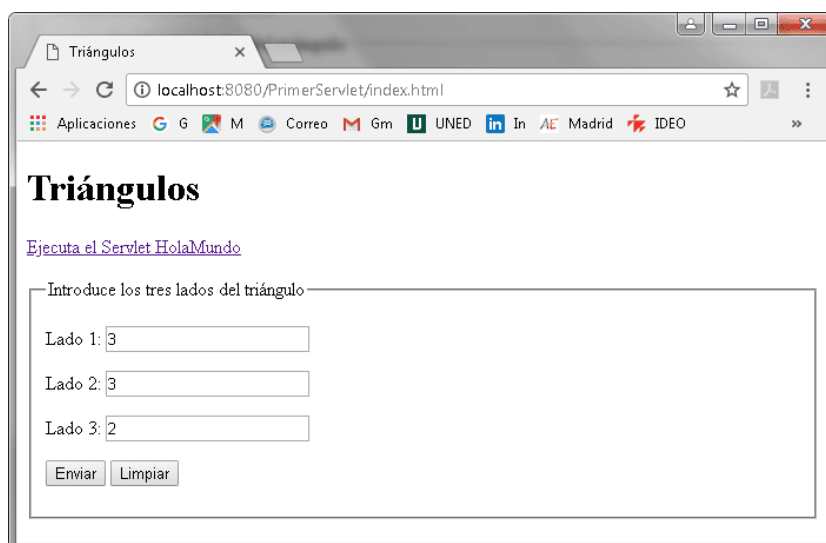
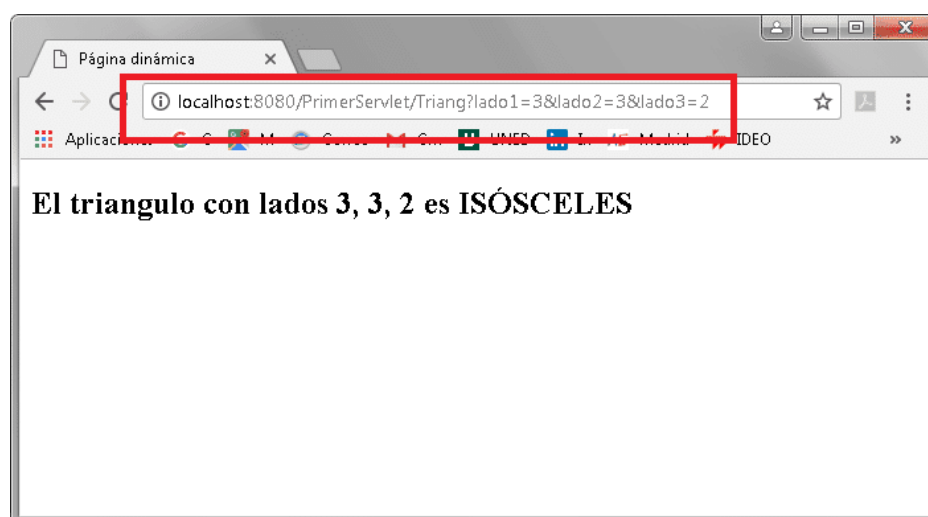
        try {
            lado1 = Integer.parseInt(request.getParameter("lado1"));
            lado2 = Integer.parseInt(request.getParameter("lado2"));
            lado3 = Integer.parseInt(request.getParameter("lado3"));
            flujoEscritura.append("<h2>El triangulo con lados " + lado1 + ", " +
lado2 + ", " + lado3 + " es ");
            if (lado1==lado2 && lado2==lado3)
                flujoEscritura.append("EQUILÁTERO</h2>");
            else if (lado1==lado2 || lado2==lado3 || lado1==lado3)
                flujoEscritura.append("ISÓSCELES</h2>");
            else
                flujoEscritura.append("ESCALENO</h2>");

        } catch (NumberFormatException e) {
            flujoEscritura.append("Alguno de los lados que has introducido no es
correcto");
        }

        flujoEscritura.append("</body></html>");
        flujoEscritura.close();
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        doGet(request, response);
    }
}
```

Ahora, es importante que prestes atención a lo que está ocurriendo cuando el usuario hace clic en el botón *Enviar* del formulario. Puede resumirse en estos puntos:

1. El usuario escribe la longitud de los tres lados:**2. Después, el usuario hace clic en el botón *Enviar*, lo que provoca la **redirección al servlet *Triang***.**

En la nueva URL, no sólo aparece el nombre del servlet, sino que además se pasan como parámetros los valores que ha introducido el usuario y el *servlet* es capaz de recoger dichos parámetros para responder al usuario. Los parámetros son recogidos dentro del *servlet* por medio del objeto ***request*** y el método ***getParameter()***.

```
lado1 = Integer.parseInt(request.getParameter("lado1"));  
lado2 = Integer.parseInt(request.getParameter("lado2"));  
lado3 = Integer.parseInt(request.getParameter("lado3"));
```

Vamos a repasar el sistema que utilizamos desde el formulario HTML para provocar la ejecución del *servlet*.

```
<form action="Triang" method="get">
```

En el parámetro **action** indicaste el nombre del *servlet*, y en el parámetro **method**, el método de envío. El método *get* hace que los parámetros que envía el usuario se pasen a través de la URL. Otra posibilidad es usar el método *post*, que hace que los parámetros se envíen en el cuerpo de la petición y, por lo tanto, no se muestren en la URL. **Este sistema sería más conveniente para datos que no nos interesa que se vean por temas de seguridad y privacidad.**

Ciclo de vida de un servlet

Cuando no estás familiarizado con el desarrollo de aplicaciones web, puede resultar complejo comprender su ciclo de vida.

Puede que estés acostumbrado a que un programa comienza, realiza las tareas que tiene encomendadas y termina. El **ciclo de vida de un *servlet*** es bien distinto.

Veamos las características más importantes de la vida de un *servlet*.

- El *servlet* queda alojado en la memoria del servidor la primera vez que un usuario accede a él. A partir de ese momento, queda disponible para todos los usuarios que accedan a él.
- Miles de usuarios al mismo tiempo podrían estar solicitando la ejecución del *servlet* y, por lo tanto, provocando la ejecución de los métodos *doGet()* o *doPost()*, pero el *servlet* se ha tenido que registrar en el servidor una sola vez.
- El *servlet* seguirá estando activo y esperando solicitudes hasta que la persona encargada de la administración del servidor decida detenerlo por labores de mantenimiento.

En ocasiones, es necesario ejecutar una tarea solamente en el momento que se registra el *servlet* en el servidor; por ejemplo, para abrir conexión con una base de datos. Pues bien, es posible sobrescribir un método heredado de la clase *HttpServlet* para ese fin. Se trata del método *init()*. Por otro lado, también podemos sobrescribir el método *destroy()* para ejecutar alguna acción cuando el *servlet* pasa a liberarse de la memoria del servidor.

Modifica el código del primer *servlet* de la siguiente forma:

```
package servlets;

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;

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

```
public HolaMundo() {
    super();
}

protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {

    response.setContentType("text/html");
    PrintWriter flujoEscritura=response.getWriter();

    flujoEscritura.append("<!DOCTYPE html>");
    flujoEscritura.append("<html><head><meta charset='UTF-8'>");
    flujoEscritura.append("<title>Página dinámica</title>");
    flujoEscritura.append("</head><body>");
    for (int i=1; i<7; i++) {
        flujoEscritura.append("<h"+i+">Hola Mundo</h"+i+">");
    }
    flujoEscritura.append("</body></html>");

    flujoEscritura.close();

}

protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
    doGet(request, response);
}

@Override
public void init() throws ServletException {
    System.out.println("El Servlet se ha registrado en el servidor");
}

@Override
public void destroy() {
    System.out.println("El Servlet ha sido retirado de la memoria del
servidor");
}

}
```

Hemos sobrescrito los métodos *init()* y *destroy()* mostrando un simple mensaje en la consola de Eclipse. Sólo verás el mensaje “El *servlet* se ha registrado en el servidor” la primera vez que se ejecute el *servlet* después de haber reiniciado el servidor.



Puede que te cueste más ver el mensaje mostrado por el método `destroy()`, ya que puede pasar tiempo hasta que el recolector de basura de Java libere realmente el objeto.

Acceso a datos desde un servlet

Ya has comprobado que **un *servletes* en realidad una clase Java cualquiera, pero que extiende de `HttpServlet`.**

Recuerda que un *servlet* es una clase, y que desde él puedes acceder a todos los recursos de acceso a datos que has aprendido en este módulo.

Dentro de un servlet puedes:

- Acceder a **ficheros**, utilizando las clases que representan flujos de datos y que aprendiste en la unidad 1 de este módulo.
- Acceder a una **base de datos** relacional, utilizando el **API JDBC** tal y como aprendiste en la unidad 2 de este módulo.
- Acceder a una base de datos relacional utilizando un **framework ORM**, tal y como aprendiste en la unidad 3 de este módulo.
- Acceder a documentos **XML**, utilizando el **parser DOM** o solicitando los servicios de acceso a bases de datos XML que proporciona la aplicación **BaseX**.

Servlet con acceso a base de datos XML

Vamos a crear un *servlet* que genere una página dinámica con el inventario de artículos de la base de datos XML de ALMACEN que creamos en la unidad anterior.

Comienza por crear un proyecto web dinámico que se ejecutará en el servidor Apache Tomcat, tal y como has aprendido.

Dentro del proyecto, accederemos al servicio que presta BaseX para acceso a las bases de datos que administra, por lo que es importante que prepares el proyecto para dicho objetivo. Debes realizar un par de tareas para que tu proyecto funcione:

- Importa dentro del proyecto los tres archivos que conforman la librería BaseX XQJ API.
- **IMPORTANTE:** puesto que el proyecto será ejecutado por Apache Tomcat, copia también los tres archivos de la librería BaseX XQJ API dentro de la carpeta *lib* de la instalación de Apache Tomcat.

Dentro de tu nuevo proyecto, crea un *servlet* llamado *Inventario* con el siguiente código:

```
package servlets;

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;
import javax.xml.xquery.XQConnection;
import javax.xml.xquery.XQDataSource;
import javax.xml.xquery.XQException;
import javax.xml.xquery.XQExpression;
import javax.xml.xquery.XQResultSequence;

import org.w3c.dom.Node;
import org.w3c.dom.NodeList;

import net.xqj.baseX.BaseXXQDataSource;

@WebServlet("/Inventario")
public class Inventario extends HttpServlet {
    private static final long serialVersionUID = 1L;

    public Inventario() {
        super();
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        XQDataSource xds = new BaseXXQDataSource();
        XQConnection con;
        XQExpression expr;
        XQResultSequence result;
        String sentencia;
        String textoHTML = "";
    }
}
```

```

        try {
            xds.setProperty("serverName", "localhost");
            xds.setProperty("port", "1984");
            con = xds.getConnection("admin", "admin");
            expr = con.createExpression();
            sentencia = "for $pro in fn:collection('almacen')//productos return
$pro/producto";

            result = expr.executeQuery(sentencia);
            while (result.next()) {
                Node nodoProducto = result.getNode();
                textoHTML = mostrarProducto(nodoProducto, textoHTML);
            }
            con.close();
        } catch (XQException e) {
            textoHTML = "Error al obtener los datos XML <br />";
            textoHTML = textoHTML + e.getMessage();
        }

        response.setContentType("text/html");
        PrintWriter flujoEscritura=response.getWriter();

        flujoEscritura.append("<!DOCTYPE html>");
        flujoEscritura.append("<html><head><meta charset='UTF-8'>");
        flujoEscritura.append("<title>Página dinámica</title>");
        flujoEscritura.append("</head><body>");
        flujoEscritura.append("<h1>Articulos de alimentación</h1>");
        // Creamos una tabla HTML
        flujoEscritura.append("<table>");
        // Creamos la cabecera
        flujoEscritura.append("<tr>");
        flujoEscritura.append("<th>Artículo</th>");
        flujoEscritura.append("<th>Cantidad por cada unidad</th>");
        flujoEscritura.append("<th>Precio Unidad</th>");
        flujoEscritura.append("<th>Stock</th>");
        flujoEscritura.append("</tr>");
        flujoEscritura.append(textoHTML);
        flujoEscritura.append("</table>");
        flujoEscritura.append("</body></html>");
        flujoEscritura.close();
    }

    private static String mostrarProducto(Node nodo, String texto) {
        NodeList nodos = nodo.getChildNodes();
        // Creamos una fila HTML (Table Row)
        texto = texto + "<tr>";
        for (int i=0; i<nodos.getLength();i++) {
            Node nodoHijo = nodos.item(i);
            if (nodoHijo.getNodeType() == Node.ELEMENT_NODE) {
                // Creamos una celda de datos HTML (Table data)
                texto = texto + "<td>";
                texto = texto + nodoHijo.getTextContent();
                texto = texto + "</td>";
            }
        }
        texto = texto + "<tr>";
        return texto;
    }
}

```

Si te fijas detenidamente en el código del *servlet*, verás que no hemos introducido nada nuevo, sino que hemos generado una página dinámica que **accede a BaseX para obtener el inventario de productos del almacén, itera los elementos *producto* obtenido y genera una tabla HTML como respuesta al cliente.**

El resultado en el navegador quedará así:



Artículos de alimentación

Artículo	Cantidad por cada unidad	Precio	Unidad	Stock
Té Dharamsala	10 cajas x 20 bolsas	20.16		39
Cerveza tibetana Barley	24 - bot. 12 l	21.28		17
Refresco Guaraná Fantástica	12 - latas 355 ml	5.04		20
Cerveza Sasquatch	24 - bot. 12 l	15.68		111
Cerveza negra Steeleye	24 - bot. 12 l	20.16		20
Vino Côte de Blaye	12 - bot. 75 cl	295.12		17
Licor verde Chartreuse	750 cc por bot.	20.16		69
Café de Malasia	16 - latas 500 g	51.52		17
Cerveza Laughing Lumberjack	24 - bot. 12 l	15.68		52
Cerveza Outback	24 - bot. 355 ml	16.8		15
Cerveza Klosterbier Rhönbräu	24 - bot. 0,5 l	8.68		125
Licor Cloudberry	500 ml	20.16		57
Sirope de regaliz	12 - bot. 550 ml	11.2		13
Espicias Cajun del chef Anton	48 - frascos 6 l	24.64		53
Mezcla Gumbo del chef Anton	36 cajas	23.912		0
Mermelada de grosellas de la abuela	12 - frascos 8 l	28		120
Salsa de arándanos Northwoods	12 - frascos 12 l	44.8		6
Salsa de soja baja en sodio	24 - bot. 250 ml	17.36		39

Java Server Pages (JSP)

Introducción

Ahora aprenderás a crear **páginas dinámicas**, utilizando la tecnología Java Server Pages.

Java Server Pages es una tecnología que permite mezclar el código HTML, que se ejecuta en el *front-end*, con *script* de código Java que se ejecuta en el servidor y que forma parte del *back-end*, todo esto en un único archivo con extensión *.jsp*. Tan sólo se debe escribir el código HTML de la manera habitual y luego encerrar el código Java de las partes dinámicas en etiquetas especiales, muchas de las cuales comienzan con `<%` y finalizan con `%>`.

Los **elementos básicos de cualquier página JSP** son:

- **Expresiones** `<%= expresión %>`
- **Scriptlets** `<% código java %>`
- **Declaraciones** `<%! código java %>`

Sigue adelante para profundizar más sobre estos elementos.

Elementos básicos de los documentos JSP

En este apartado vamos a conocer algunos **elementos básicos de los documentos JSP**.

1. Expresiones `<%= expresión %>`

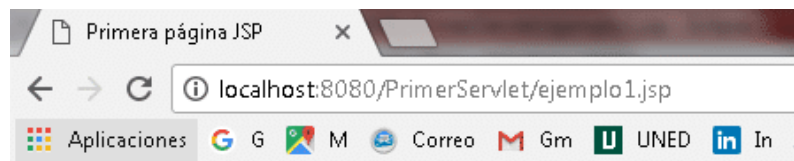
La expresión especificada se resuelve y su resultado se integra en el contenido HTML. Puedes ponerlo en práctica creando un archivo llamado *ejemplo1.jpg* con el siguiente contenido:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Primera página JSP </title>
</head>
<body>
    <p>La longitud de la circunferencia de radio 30 es
    <%= 2*30*java.lang.Math.PI%> </p>
    <p>La fecha y hora actual:
    <%= new java.util.Date() %> </p>
</html>
```

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
```

Esta cabecera es una **directiva común a todas las páginas JSP**.

Cuando veas el resultado en el navegador, comprobarás que las expresiones se han resuelto: lo que ve el usuario es el resultado.



La longitud de la circunferencia de radio 30 es 188.49555921538757

La fecha y hora actual: Thu Apr 19 18:35:12 CEST 2018

Prueba a hacer clic derecho desde el navegador y selecciona *Ver código fuente* para ver el código HTML. Comprobarás que el único contenido que aparece es HTML. Es algo así:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Primera página JSP </title>
</head>
<body>
    <p>La longitud de la circunferencia de radio 30 es
    188.49555921538757 </p>
    <p>La fecha y hora actual:
    Thu Apr 19 18:27:12 CEST 2018 </p>
</body>
</html>
```

La página JSP ha sido ejecutada en el servidor, que ha convertido las partes de código Java en código HTML estático. Este resultado estático es el que devuelve al cliente para que sea visualizado en el navegador.

2. Scriptlets <% código java %>

Un **scriptlet** es un bloque de código Java que es ejecutado en el servidor. Veamos un ejemplo: Crea otro archivo; esta vez, puedes llamarlo *ejemplo2.jsp*.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Articulo </title>
</head>
<body>
    <h1>INFORMACION SOBRE EL PRODUCTO</h1>
    <%
        double precio=10;
        int cantidad=5;
```

```

String producto="Peras";
double subtotal=precio*cantidad;
if (subtotal>40) {
    subtotal = subtotal - (subtotal * 0.1);
}
double iva=subtotal*0.16;
double total=subtotal+iva;
%>
<p>Nombre artículo:<%= producto %> </p>
<p>Precio artículo:<%= precio %> </p>
<p>Subtotal: <%= subtotal%> </p>
<p>Total IVA incluido: <%= total %> </p>
</body>
</html>

```

El resultado en el navegador es:



3. Declaraciones <%! código java %>

Nos referimos a la declaración de variables o funciones cuyo ámbito y vigencia es global, hasta que el servidor se detiene por labores de mantenimiento.

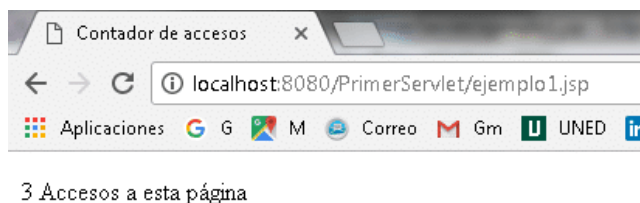
Crea ahora el archivo *ejemplo3.jsp* con el siguiente contenido:

```

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Contador de accesos </title>
</head>
<body>
    <%!
        int contador=0;
    %>
    <% contador++; %>
    <p><%= contador %> Accesos a esta página</p>
</body>
</html>

```

La variable *contador* está declarada dentro de una etiqueta especial JSP de tipo *declaración*, lo que permite que su ámbito sea global y se mantenga durante todo el tiempo que el servidor esté abierto. Puedes abrir la página incluso desde navegadores distintos y comprobar que se mantiene el valor anterior del contador.



Las variables predefinidas request y response

Para el servidor, una página JSP es como un *servlet*, ya que igualmente recibe peticiones de clientes y envía respuestas en formato HTML.

Cada página JSP es registrada en el servidor como un *servlet* y **tiene predefinidas las variables *request* y *response***. Podríamos resolver el problema de los triángulos con una página JSP en sustitución del *servlet* anterior. Veamos cómo:

Crea la página ***triang.jsp*** con el siguiente código:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<title>Contador de accesos </title>
</head>
<body>
<%
    int lado1=0, lado2=0, lado3=0;
    String resultado;

    response.setContentType("text/html");

    try {
        lado1 = Integer.parseInt(request.getParameter("lado1"));
        lado2 = Integer.parseInt(request.getParameter("lado2"));
        lado3 = Integer.parseInt(request.getParameter("lado3"));

        if (lado1==lado2 && lado2==lado3)
            resultado = "EQUILÁTERO";
        else if (lado1==lado2 || lado2==lado3 || lado1==lado3)
            resultado = "ISÓSCELES";
        else
            resultado = "ESCALENO";

    } catch (NumberFormatException e) {
        resultado = "Alguno de los lados que has introducido no es correcto";
    }

%>
<h2>Triangulo con lados <%=lado1 %>, <%=lado2 %>, <%=lado3 %></h2>
<h2><%=resultado %></h2>
</body>
</html>
```

Observa que podemos seguir utilizando el objeto *request* para recoger los parámetros introducidos por el usuario. Los objetos *response* y *request* están predefinidos, y siempre disponibles para las páginas JSP.

Ahora, sólo te falta modificar la cabecera del formulario de la página *index.html* para que utilice la página JSP en lugar del *servlet*.

```
<form action="triang.jsp" method="get">
```

Despedida

Resumen

Has terminado la lección, repasemos los puntos más importantes que hemos tratado.

- Los ***servlets*** son programas que se ejecutan en un servidor web, en una capa intermedia entre una petición proveniente de un navegador web o cliente, y las bases de datos o aplicaciones del servidor HTTP. Su misión principal es **atender peticiones de clientes, leer los parámetros de entrada y enviar una página de respuesta.**
- **Java Server Pages** es una tecnología que permite mezclar el código HTML que se ejecuta en el *front-end*, con *script* de código Java que se ejecuta en el servidor y forma parte del *back-end*, todo esto en un único archivo con extensión *.jsp*. Tan sólo se debe escribir el código HTML de la manera habitual y luego encerrar el código Java de las partes dinámicas en etiquetas especiales, muchas de las cuales **comienzan con `<%` y finalizan con `%>`.**