

PROGRAMACION N-CAPAS LABORATORIO #1



Introducción al entorno de desarrollo, configuración TomCat, uso de Servlet.

Catedrático:

Lic. Juan Lozano

Instructores:

Karla Beatriz Morales Alfaro 00022516@uca.edu.sv

Sara Noemy Romero Menjivar 00030716@uca.edu.sv

Salvador Edgardo Campos Gómez 00117716@uca.edu.sv

Para el desarrollo del curso de programación N-Capas, dos de las opciones a utilizar respecto al entorno de desarrollo son: **Eclipse o Spring Tools 4.**

Antes de realizar cualquier instalación ya sea de Eclipse o de STS es importante que tener instalado el Java Development Kit JDK, puedes descargarlo del siguiente link, en este caso se sugiere el 11 pero no hay problema si se prefiere una versión anterior, es posible que solicite una cuenta de Oracle, se puede crear una de estudiante con el correo institucional.

<https://www.oracle.com/java/technologies/javase-jdk11-downloads.html>

Eclipse



Eclipse es un entorno de desarrollo integrado (IDE) que proporciona herramientas para facilitar el trabajo de los desarrolladores de software, facilitando, pruebas unitarias con JUnit, control de versiones con VCS, asistentes (wizards) para creación de proyectos, clases, tests, etc. También es una plataforma útil para la creación de aplicaciones web con el patrón Modelo Vista Controlador (MVC) en la que está enfocada la materia.

Para descargar Eclipse es necesario bajar el instalador desde la página oficial usando el siguiente enlace:

<https://www.eclipse.org/downloads/>

Al momento de bajar y correr el instalador nos preguntará qué versión de Eclipse se desea instalar, lo más recomendado es seleccionar IDE for Java Developers. A lo largo de los laboratorios se trabajará con Spring Tools Suite pero se puede trabajar de manera similar con Eclipse.

Spring Tools 4.



STS está construido sobre Eclipse y ofrece muchas herramientas en torno a las características de Spring y simplifica en gran medida la instalación del entorno de desarrollo.

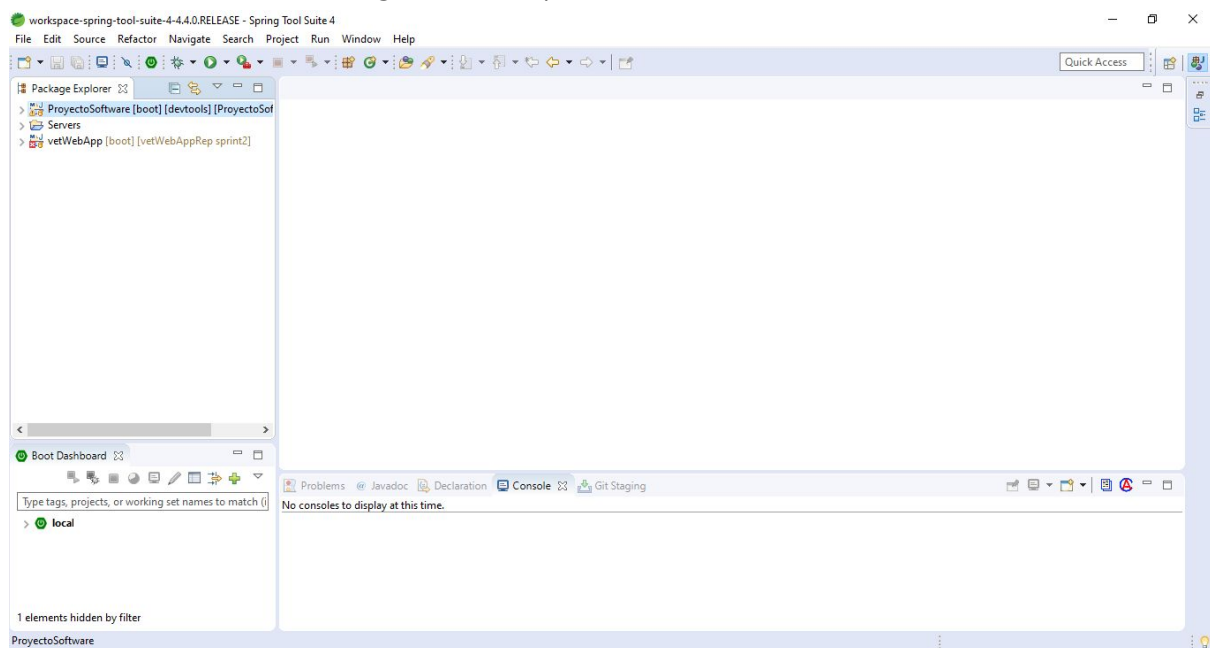
Básicamente se puede configurar Eclipse para tener todas las herramientas del Spring Tools.

Pasos par la instalación de STS:

NOTA: Se debe tener instalado el JDK

1. Ir a <https://spring.io/tools> y descargarlo para el SO que se esté utilizando.
 - a. STS ofrece integración con VSCode y Theia por si no se desea usar el IDE de Eclipse.
2. Una vez descargado iniciar el ejecutable esto extraera los archivos necesarios, una vez la barra del ejecutable termine, buscar una carpeta llamada sts dentro del fichero en el que se ejecutó (Para windows), para Linux seguir las instrucciones de <https://sdacademics.com/instalar-spring-tool-suite-en-ubuntu-18-04/> u otra guía para la distribución de Linux específica que se este utilizando.
3. Si no se había utilizado un JDK antes es posible que tengamos que configurar el PATH, para esto seguir el siguiente tutorial <https://javatutorial.net/set-java-home-windows-10>
4. Mover la carpeta donde se desee iniciar el SpringToolsSuite4.exe
5. Lo primero que pedirá es el Workspace, lo mejor es dejarlo en default y hacer clic en Launch.

Si todo fue correcto se llegará a esta pantalla:



Configuración de TomCat

¿Qué es TomCat?



Es un software de código abierto que implementa Java Servlet, JavaServer Pages, Java Expression Language y Java WebSocket.

En otras palabras es un Servidor donde código del lado del servidor será ejecutado, lo que permite desplegar una aplicación Web dinámica fácilmente sin necesidad de hacer configuraciones astrales.

Para descargar TomCat:

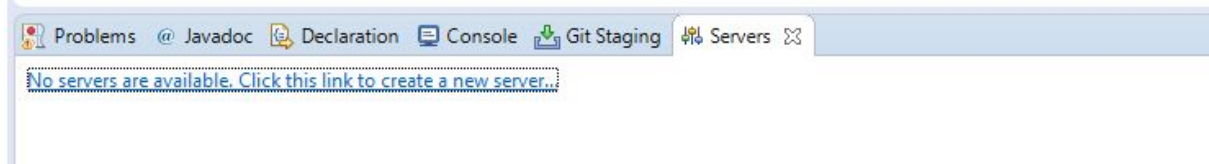
1. Ir a <https://tomcat.apache.org/download-90.cgi> (En este caso es para la versión 9.0 de no indicar lo contrario se trabajará con esta versión).
2. Del core escoger el indicado para nuestro Sistema Operativo. Para windows el 64-bit Windows zip (o de 32-bit si el SO lo requiere). Y para Linux el tar.gz.

Binary Distributions

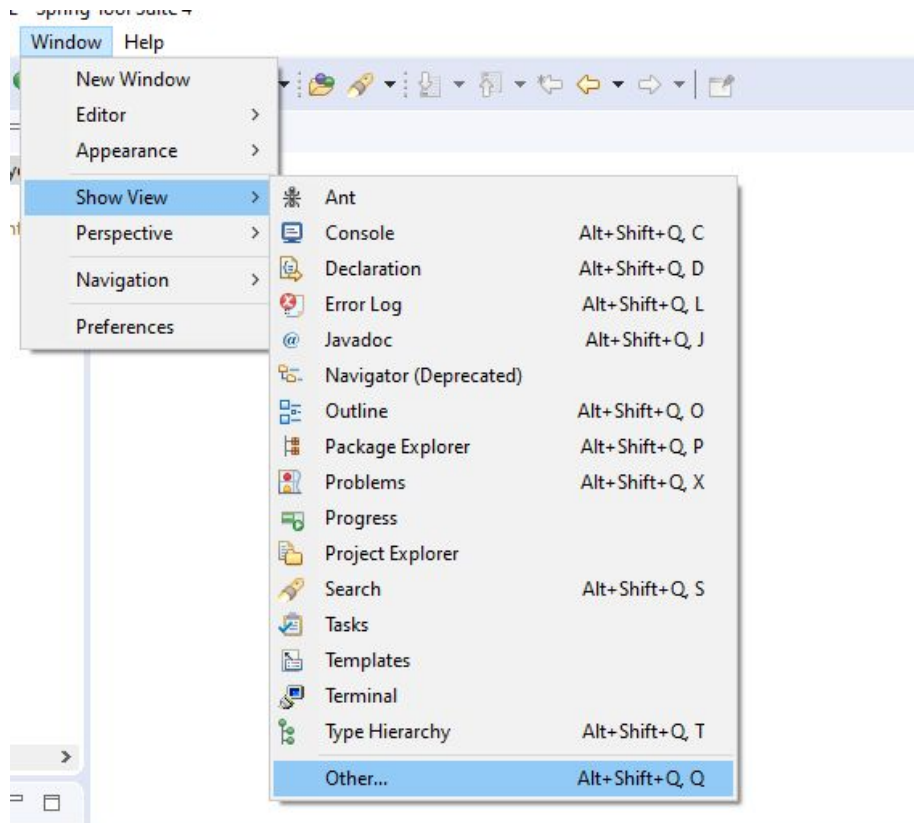
- Core:
 - [zip](#) ([pgp](#), [sha512](#))
 - [tar.gz](#) ([pgp](#), [sha512](#))
 - [32-bit Windows zip](#) ([pgp](#), [sha512](#))
 - [64-bit Windows zip](#) ([pgp](#), [sha512](#))
 - [32-bit/64-bit Windows Service Installer](#) ([pgp](#), [sha512](#))
 - Full documentation:
 - [tar.gz](#) ([pgp](#), [sha512](#))
 - Deployer:
 - [zip](#) ([pgp](#), [sha512](#))
 - [tar.gz](#) ([pgp](#), [sha512](#))
 - Embedded:
 - [tar.gz](#) ([pgp](#), [sha512](#))
 - [zip](#) ([pgp](#), [sha512](#))
3. Luego de descargarlo, descomprimirlo y mover la carpeta a donde se desee (De preferencia en Mis Documentos o un lugar de fácil acceso).

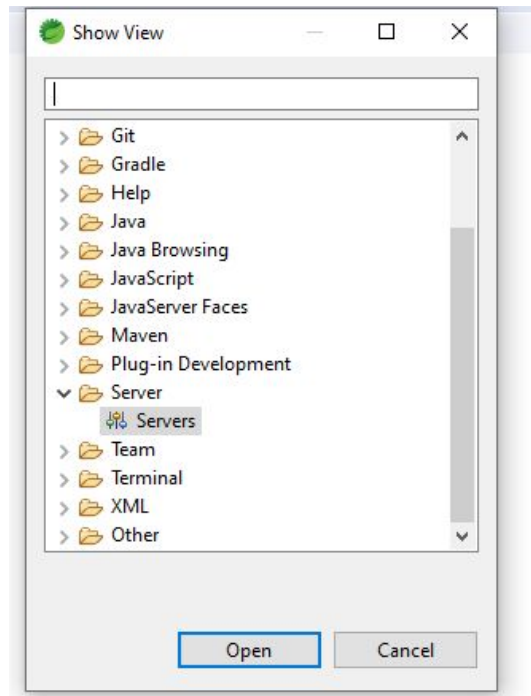
Configurando el servidor TomCat en Spring Tools Suite 4.

En la parte de abajo buscar la pestaña “Servers”.

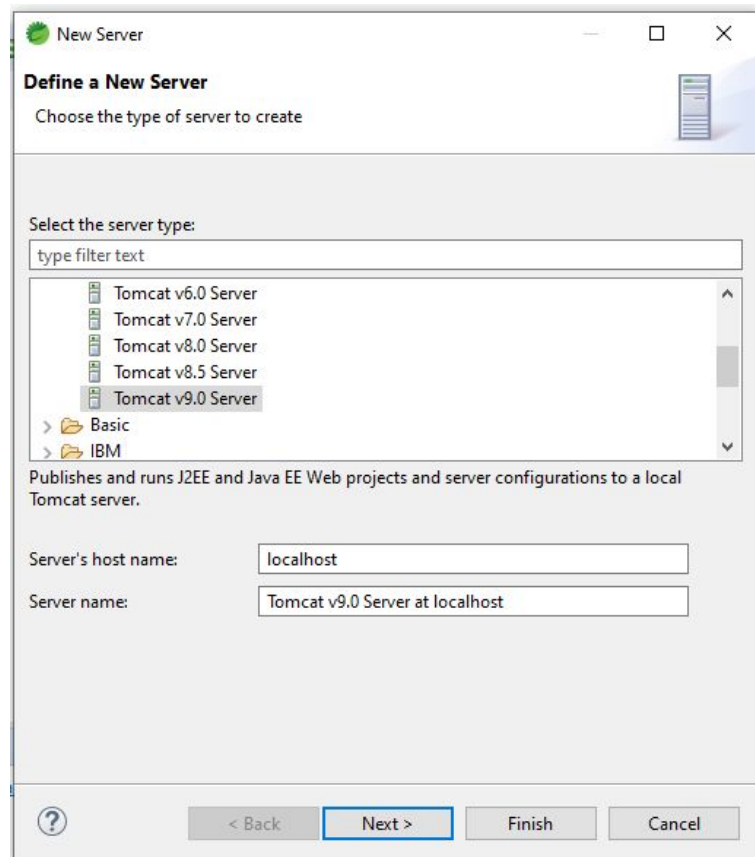


De no salir la pestaña ir a:

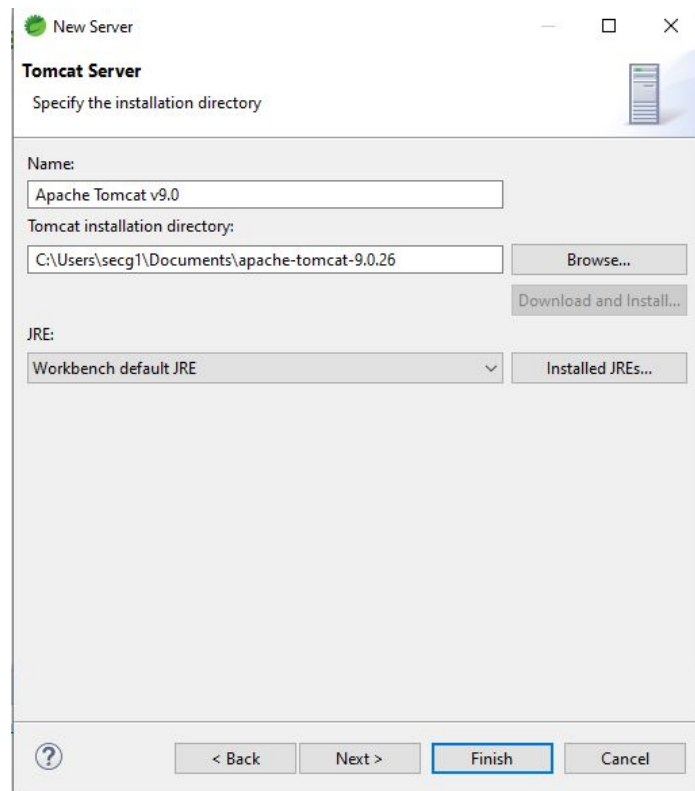




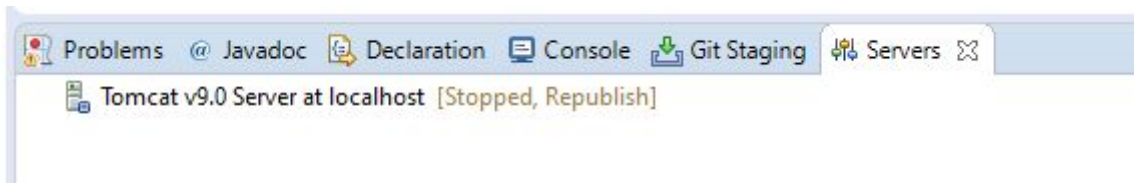
Luego dar clic en “Click this link to create a new server”.
Buscar la versión del TomCat que se va a usar y luego dar Next.



En el siguiente paso hay que buscar la carpeta donde se extrajo el TomCat Server, en el JRE dejar Workbench Default y dar clic en Finish.



Listo se ha configurado correctamente el TomCat en Spring Tools 4.



Más adelante en la guía veremos cómo ejecutar un proyecto en nuestro recién configurado Server. Antes se retomarán unos conceptos básicos.

¿Qué son Servlets?

Un servlet es un objeto de java que pertenece a una clase que extiende `javax.servlet.http.HttpServlet`, este nos permite crear aplicaciones web dinámicas. Los servlets reciben peticiones desde un navegador web, las procesan y devuelven una respuesta al navegador.

¿Qué es un contenedor Servlet?

Un contenedor Servlet es un programa capaz de recibir peticiones de páginas web y redireccionar estas peticiones a un objeto Servlet específico, el que usaremos es Apache Tomcat.

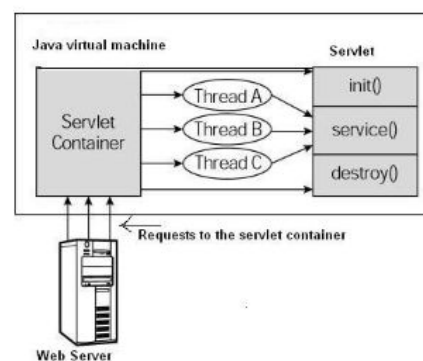
¿Cómo funciona?

El navegador pide una página al servidor HTTP (contenedor Servlet), el contenedor delega la petición a un Servlet en particular, el Servlet (objeto Java) se encarga de procesar y regresar una respuesta.

Esquema de funcionamiento:

1. Cuando el contenedor de Servlet inicia, despliega y carga todas las aplicaciones web que contiene, se crea el objeto ServletContext y se almacena en memoria.
2. Luego se parsea el archivo **web.xml** o las clases anotadas con **@WebServlet** y se instancian una vez (Patrón Singleton).
3. Mediante Multithread y pooling el servlet con una sola instancia puede atender muchas peticiones de varios clientes.
4. Cuando el contenedor de servlets se apaga, se invoca el método `destroy()` de todos los servlets inicializados.

Servlet-LifeCycle



La clase `HttpServlet` posee los siguientes métodos principales:

- **`doGet()`**: Invocado por el método `service` del servlet para manejar los requests GET del cliente.
- **`doPost()`**: Similar a `doGet`, pero maneja los request POST.
- **`doPut()`**: Método que se utiliza cuando el cliente envía archivos al servidor.
- **`doDelete()`**: Permite al cliente eliminar del servidor.
- **`init()` y `destroy()`**: Se utilizan para manejar recursos durante el ciclo de vida del servlet.

Para el laboratorio se trabajarán los métodos **`doGet()`** y **`doPost()`**.

Mapeo de Servlets:

El mapeo de servlets especifica el contenedor Web del cual un servlet de java debe ser invocado para una URL brindada por un cliente, hay dos métodos para el mapeo de Servlets por medio de **XML** y por medio de **Anotaciones de Spring**.

Definir servlets mediante XML luce así... se imaginan una aplicación Web con miles de Servlets y tener que definirlos todos en el `web.xml`... no suena genial ¿Verdad? por eso es mejor utilizar anotaciones:


```

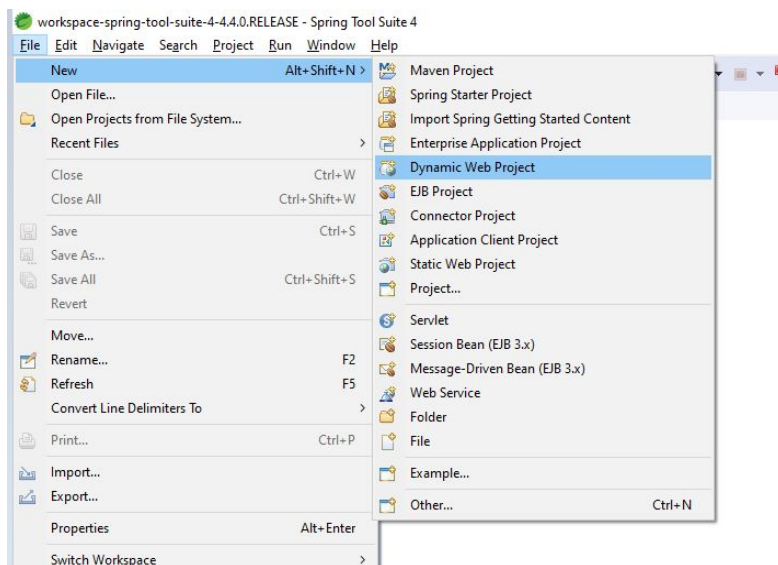
1 <?xml version="1.0" encoding="UTF-8"?>
2 <web-app>
3   <welcome-file-list>
4     <welcome-file>index.jsp</welcome-file>
5   </welcome-file-list>
6   <!-- Mapeando servlets con XML -->
7   <!--<servlet>
8     <servlet-name>MainServlet</servlet-name>
9     <servlet-class>com.uca.capas.servlets.MainServlet</servlet-class>
10  </servlet>
11  <servlet-mapping>
12    <servlet-name>MainServlet</servlet-name>
13    <url-pattern>/main</url-pattern>
14  </servlet-mapping> -->
15 </web-app>

```

Ejemplo demostrativo:

El ejemplo lo pueden descargar de: <https://github.com/salvadorc94/laboCapas1.git>

Para comenzar se tiene que abrir el STS 4, luego ir a Archivo, nuevo proyecto Web Dinámico en caso no salga entre las opciones buscarla en "Project...":



Luego se debe nombrar el proyecto, en Target Runtime se coloca el servidor que se utilizará, en nuestro caso es el servidor que ya habíamos configurado previamente, se puede dejar vacío y posteriormente añadirle el servidor pero resulta más fácil hacerlo desde la creación del proyecto.

New Dynamic Web Project

Dynamic Web Project

Create a standalone Dynamic Web project or add it to a new or existing Enterprise Application.

Project name:

Project location

☒ Use default location

Location:

Target runtime

Dynamic web module version

Configuration

A good starting point for working with Apache Tomcat v9.0 runtime. Additional facets can later be installed to add new functionality to the project.

EAR membership

☐ Add project to an EAR

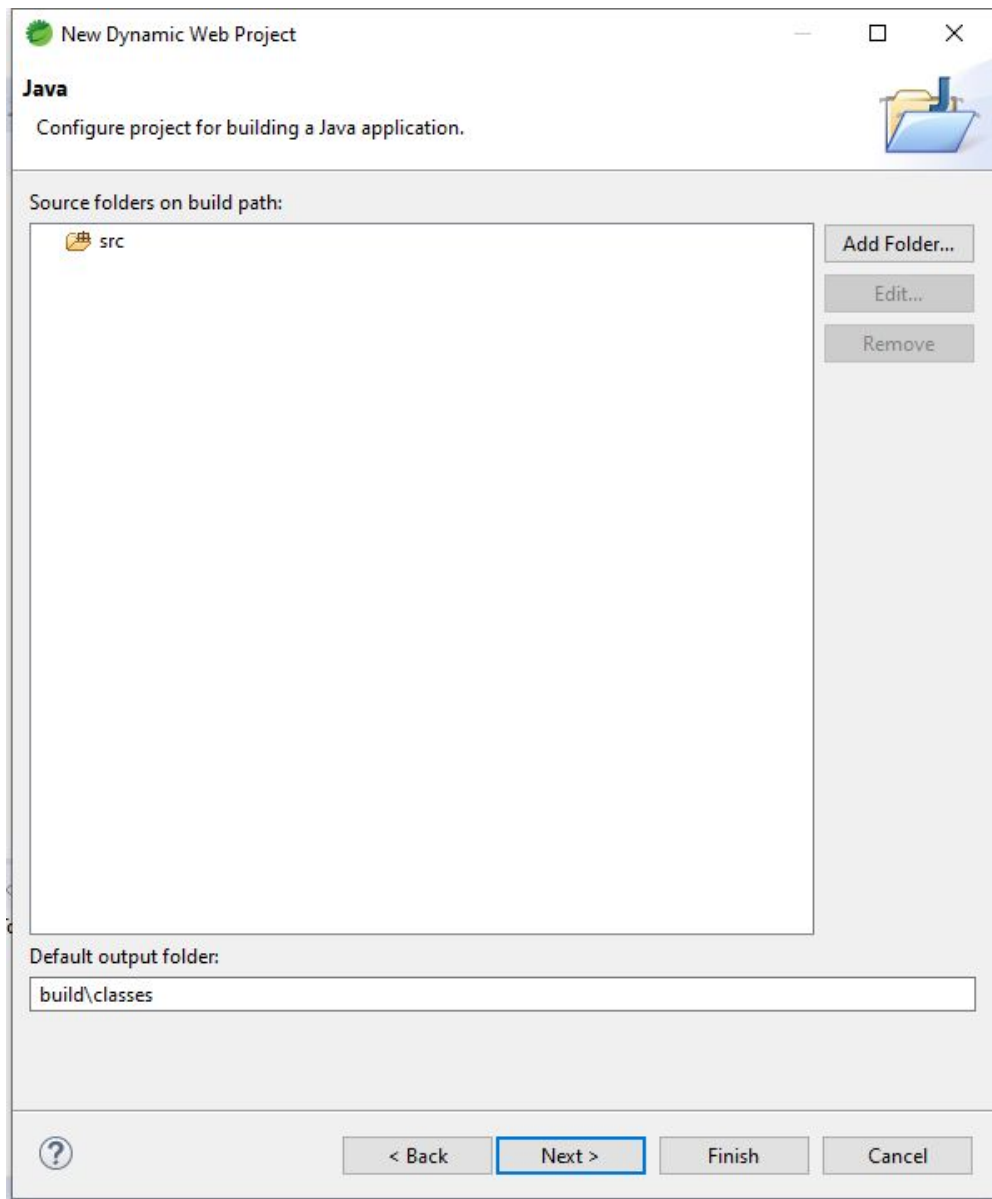
EAR project name:

Working sets

☐ Add project to working sets

Working sets:

Luego se da Next.

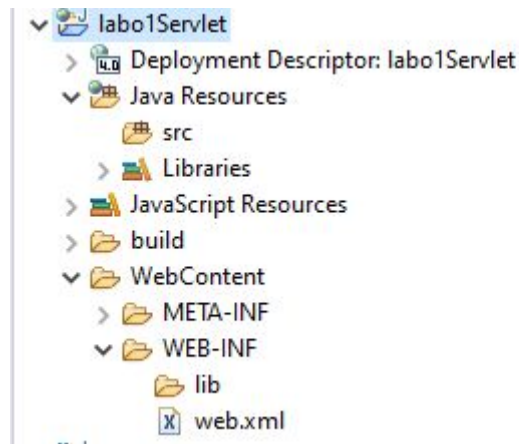


Luego Next.



En esta última vamos a marcar la casilla Generate web.xml deployment descriptor, esto para definir la página de inicio de nuestro proyecto (En este archivo también se pueden especificar los servlets haciendo uso de **XML** como lo vimos en la imagen de mapeo de servlets). Luego de marcar la casilla se da Finalizar.

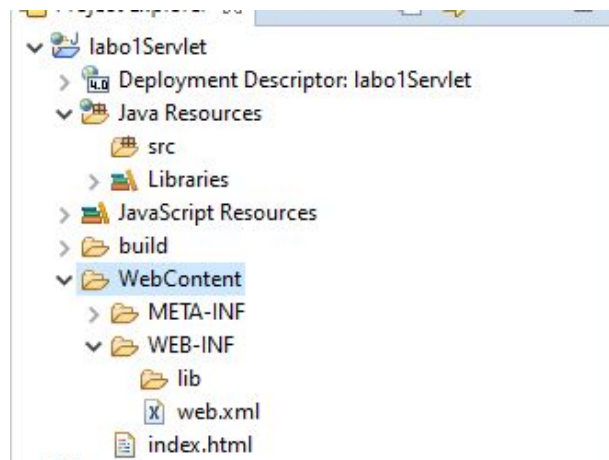
Se crea el directorio de nuestro proyecto:



Como se puede observar el archivo web.xml fue creado automáticamente.

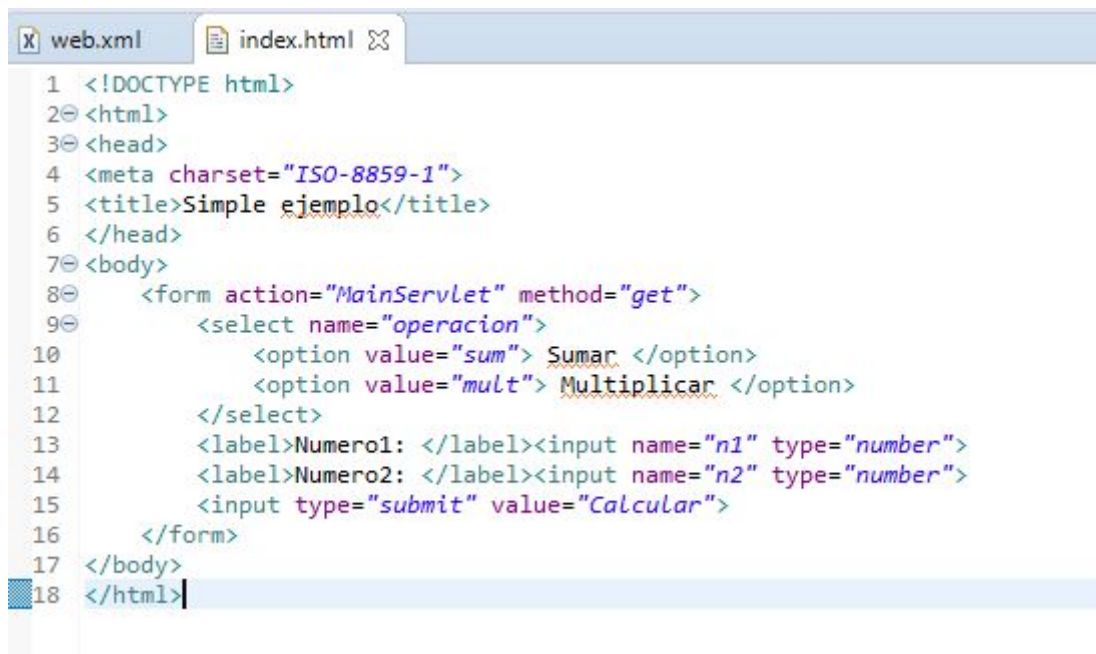


El siguiente paso será crear en la carpeta WebContent nuestro index.html, como pueden ver el index.html se encuentra en el archivo web.xml como "Welcome-File" es decir al iniciar el proyecto en el Servidor mostrará esa página de no encontrarla buscará las demás.



NOTA: tener cuidado de no posicionar el index.html en WEB-INF o META-INF debe estar en WebContent, de lo contrario no se encontrará el archivo y el servidor dará error 404.

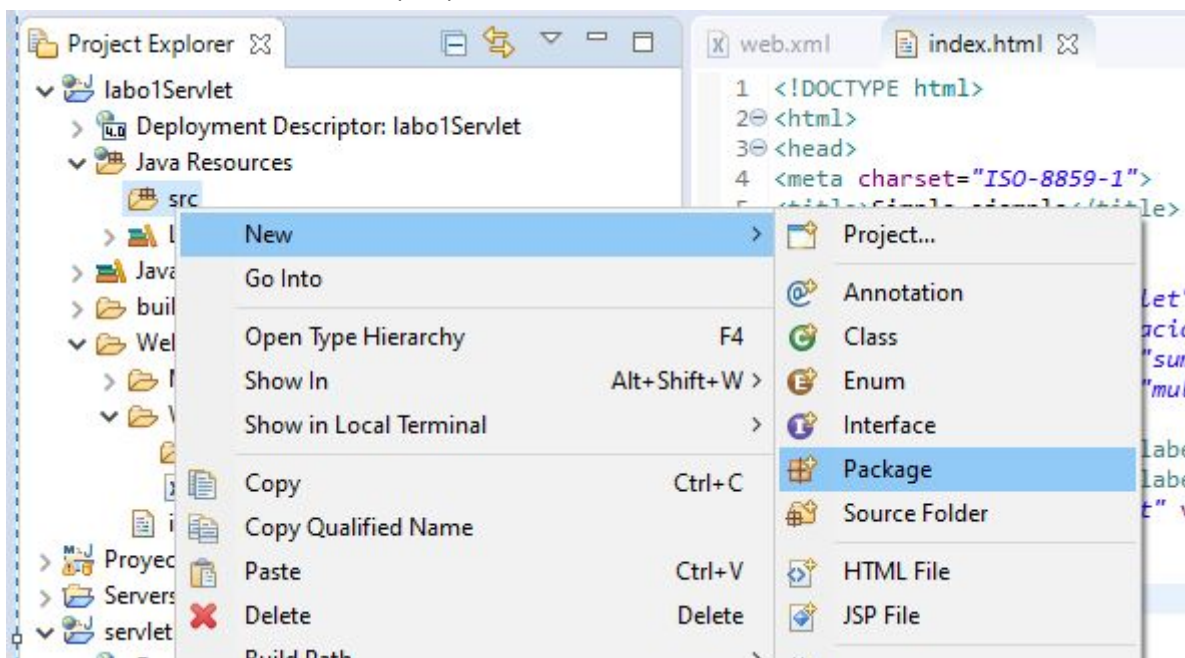
Dentro del index.html colocaremos lo siguiente:



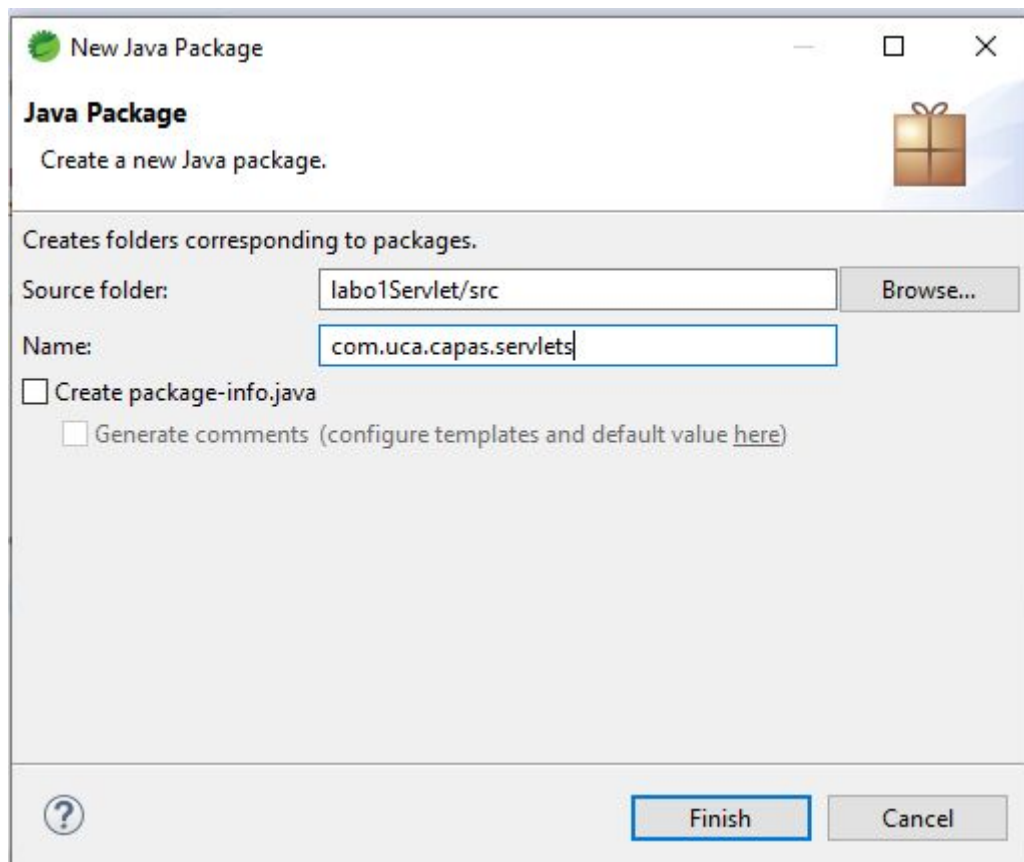
```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset="ISO-8859-1">
5 <title>Simple ejemplo</title>
6 </head>
7 <body>
8 <form action="MainServlet" method="get">
9 <select name="operacion">
10 <option value="sum"> Sumar </option>
11 <option value="mult"> Multiplicar </option>
12 </select>
13 <label>Numero1: </label><input name="n1" type="number">
14 <label>Numero2: </label><input name="n2" type="number">
15 <input type="submit" value="Calcular">
16 </form>
17 </body>
18 </html>
```

Es un simple formulario, es importante poner el atributo “name” ya que con este se hará referencia al valor que cada campo tenga. También se puede observar que el action del form lleva a “MainServlet”.

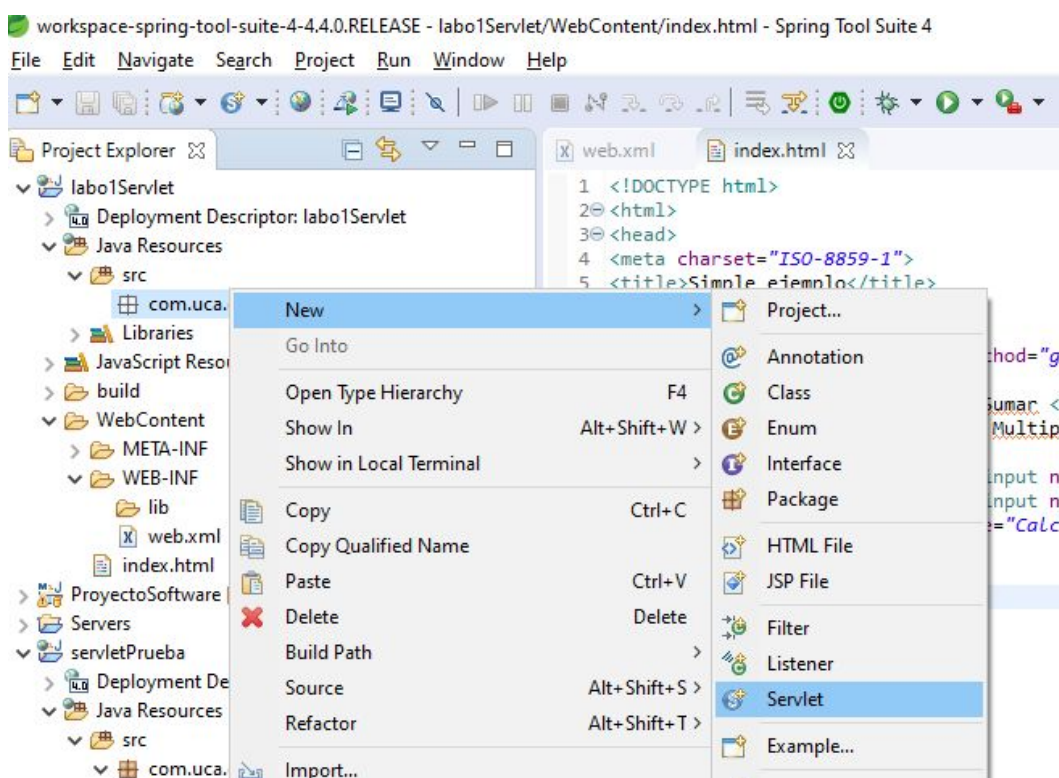
El siguiente paso es crear nuestro Servlet para esto nos vamos a Java Resources y en SRC se creará un nuevo paquete:

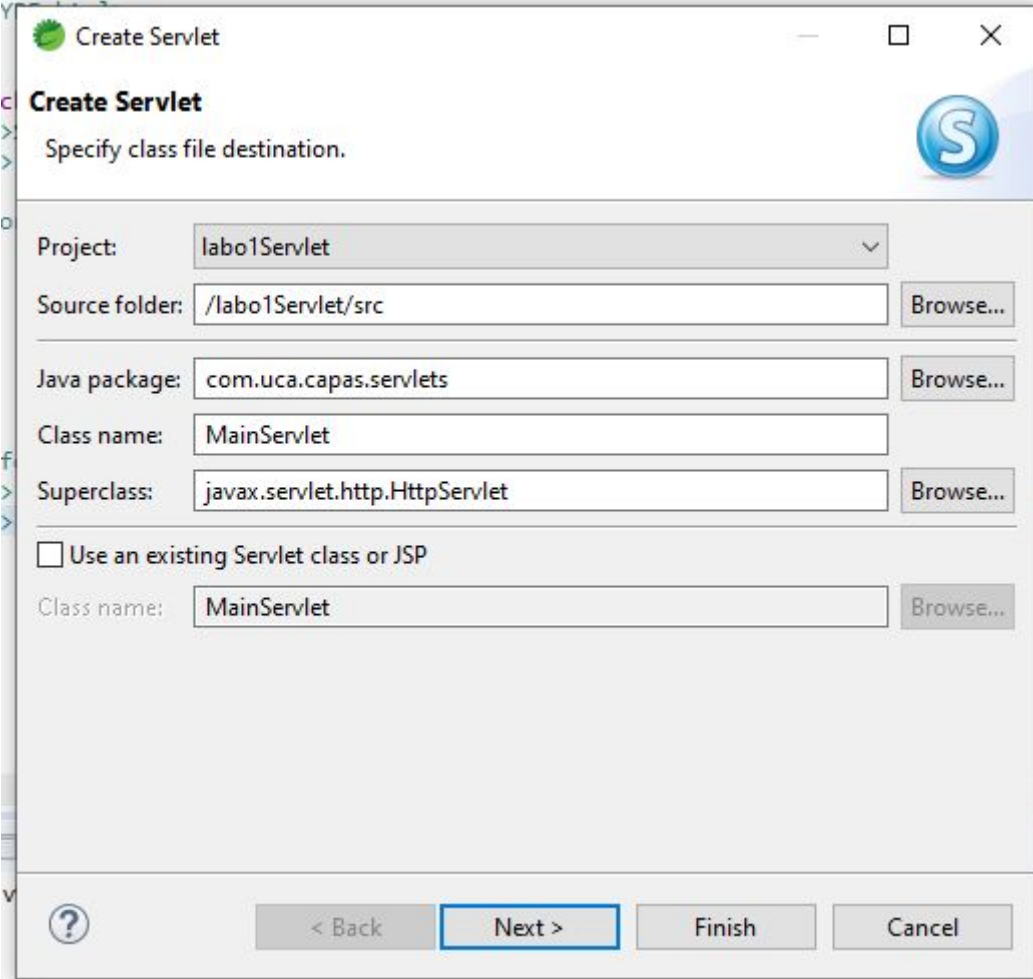


El paquete lo nombraremos “com.uca.capas.servlets” y en este paquete guardaremos todos los Servlets que se vayan a hacer.



Luego dar en Finish, dentro de nuestro paquete crearemos el Servlet que debe tener el mismo nombre del action del form:





Create Servlet

Specify class file destination.

Project: labo1Servlet

Source folder: /labo1Servlet/src Browse...

Java package: com.uca.capas.servlets Browse...

Class name: MainServlet

Superclass: javax.servlet.http.HttpServlet Browse...

☐ Use an existing Servlet class or JSP

Class name: MainServlet Browse...

< Back Next > Finish Cancel

Luego se hace clic en Finish.

Dentro de nuestro Servlet tendremos lo siguiente:

```

1 package com.uca.capas.servlets;
2
3 import java.io.IOException;
4
5
6 /**
7  * Servlet implementation class MainServlet
8  */
9 @WebServlet("/MainServlet")
10 public class MainServlet extends HttpServlet {
11     private static final long serialVersionUID = 1L;
12
13     /**
14      * @see HttpServlet#HttpServlet()
15      */
16     public MainServlet() {
17         super();
18         // TODO Auto-generated constructor stub
19     }
20
21     /**
22      * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
23      */
24     protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
25         // TODO Auto-generated method stub
26         response.getWriter().append("Served at: ").append(request.getContextPath());
27     }
28
29     /**
30      * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
31      */
32     protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
33         // TODO Auto-generated method stub
34         doGet(request, response);
35     }
36 }

```

1. Como se puede observar la clase está con la anotación **@WebServlet** y un mapeo a `"/MainServlet"` de tal manera que cuando en el form se haga clic en el botón el action lleva a `"MainServlet"` lo que hace que el Servlet tome esa petición si el mapeo está mal escrito tanto en la anotación como en el action del form la aplicación Web no encontrará el recurso solicitado.
2. La clase `MainServlet` extiende de `HttpServlet` que incluye los métodos que ya habíamos discutido antes.
3. Se encuentran ya implementados los dos métodos **`doGet()`** y **`doPost()`**.

Luego de repasar esos puntos añadiremos que hará la aplicación cuando del formulario entre al servlet, ya que el formulario se hizo con método GET el código se implementa en el método `doGet()` de la clase.

```
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    // TODO Auto-generated method stub
    //response.getWriter().append("Served at: ").append(request.getContextPath());
    String operator = request.getParameter("operacion");
    int number1 = Integer.parseInt(request.getParameter("n1"));
    int number2 = Integer.parseInt(request.getParameter("n2"));

    PrintWriter out = response.getWriter();

    if(operator.equals("sum")) {
        int res = number1+number2;
        out.println("<html>");
        out.println("<body>");
        out.println("<h3>" + "Resultado de " + number1 + " + " + number2 + " es: " + res + "</h3>");
        out.println("</body>");
        out.println("</html>");
    }else {
        out.println("<html>");
        out.println("<body>");
        out.println("<h3>" + "Resultado de " + number1 + " * " + number2 + " es: " + number1*number2 + "</h3>");
        out.println("</body>");
        out.println("</html>");
    }
}
```

En este caso recuperamos los valores del formulario utilizando el método `getParameter` del request, en donde lo que está entre comillas es el "name" del elemento del form.

1. Recuperamos el value del Select en un String.
2. Recuperamos los dos números del form.

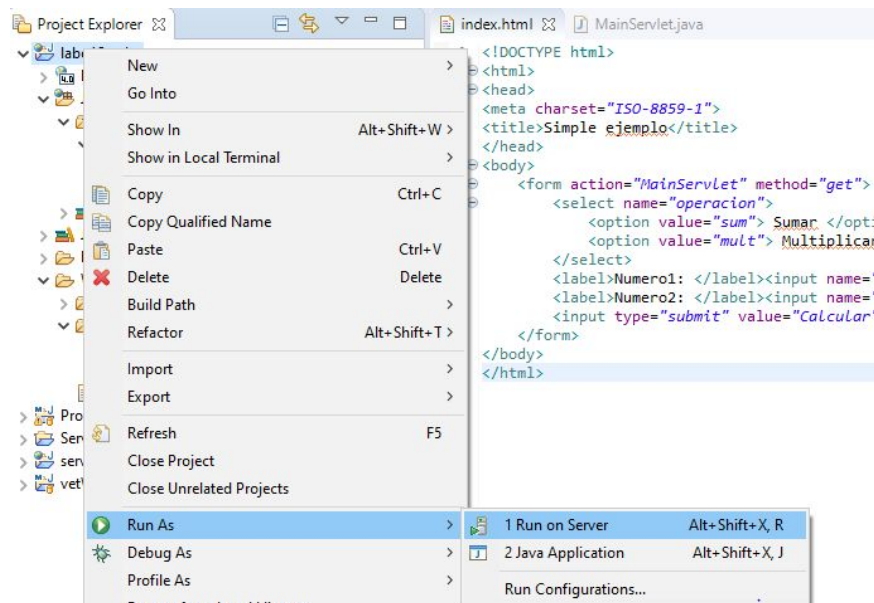
Luego retornamos haciendo uso del método `getWriter()` del response.

Cómo se puede observar hay que imprimir en orden el HTML, y dentro se pueden hacer operaciones y concatenar variables.

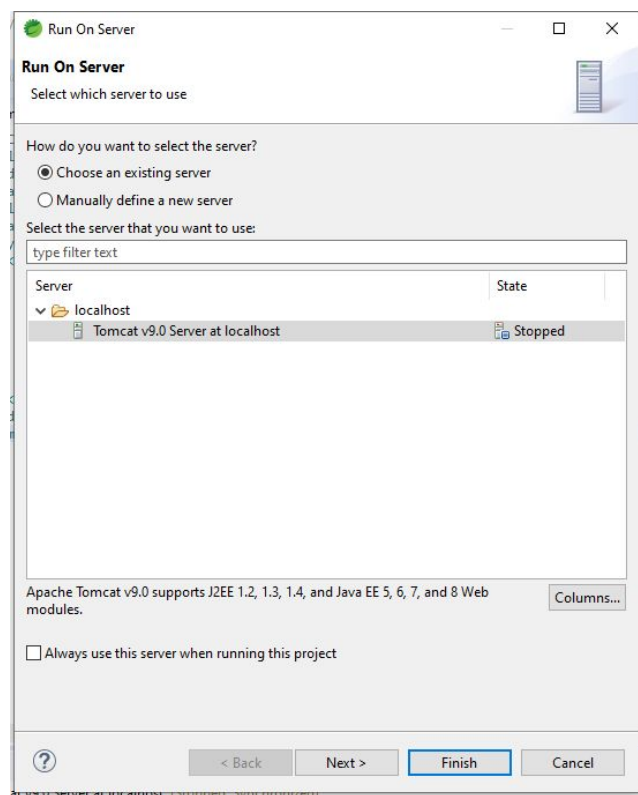
Nota: ¿Se imaginan una página grande y tener que ir imprimiendo línea por línea el HTML?

Para eso se hace uso de otras tecnologías que se verán más adelante.

Ya tenemos nuestro Servlet y tenemos nuestra página HTML, solo queda probar el proyecto en nuestro servidor TomCat:

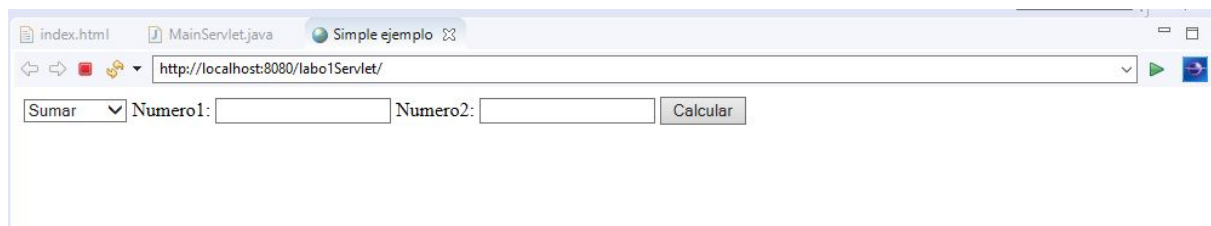


Luego se selecciona el server, como pueden observar podemos añadir un nuevo Servidor, para nuestro caso como ya lo teníamos configurado solo se selecciona y se da en finalizar:

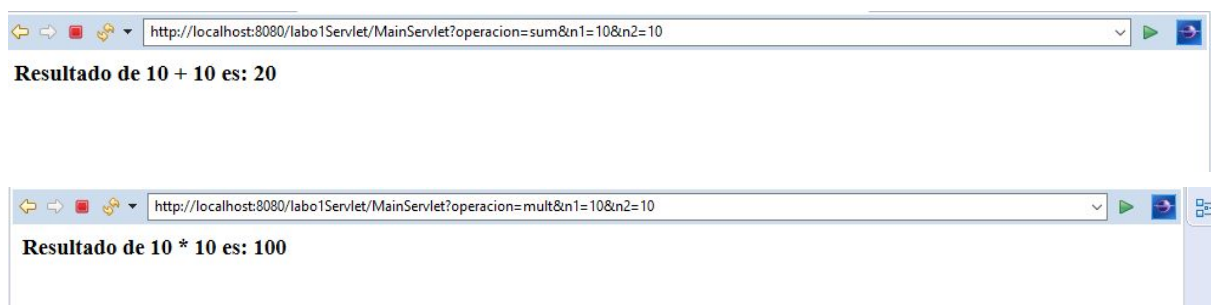


En consola se podrá ver el progreso del servidor inicializando, si se llega a la última línea es que el servidor pudo levantar sin ningún problema y dentro del IDE nos abriría la aplicación WEB.

```
mar. 16, 2020 6:53:32 P. M. org.apache.coyote.AbstractProtocol start
INFO: Starting ProtocolHandler ["ajp-nio-8009"]
mar. 16, 2020 6:53:32 P. M. org.apache.catalina.startup.Catalina start
INFO: Server startup in [861] milliseconds
```



También se puede copiar la URL y probarla en el navegador que más prefieran mientras el server siga ejecutándose, al finalizar pruebas es importante detener el Server.



NOTA: Como el método que se utiliza es GET en la URL se encuentran los parámetros del formulario y sus valores. ¿Para un login sería lo ideal utilizar GET? No, para eso se utiliza el method POST del form y el método doPost() del Servlet.

TAREA (100% DE LA NOTA).

Elaborar un login utilizando el método doPost() de un Servlet, dicho login debe recibir un nombre de usuario y contraseña; y devolver si tiene acceso o no a la plataforma.

Detalles:

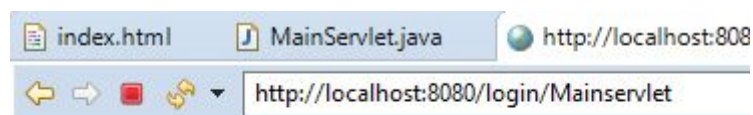
1. El proyecto debe ser nombrado **“tareaLabo1”**.
2. Crear un archivo index.html en el WebContent.
 - a. Dentro del index debe haber un formulario con dos input y un botón.
 - b. No es necesario validar los input.
3. Crear en la carpeta src un paquete “com.uca.capas.servlets”.
4. Dentro del paquete crear un servlet llamado “MainServlet”.

5. Modificar el Servlet de manera adecuada y utilizando el método **doPost()** de tal manera que devuelva si el usuario tiene acceso o no dependiendo de las credenciales ingresadas.
 - a. Las credenciales correctas pueden ir quemadas.
6. Subir el proyecto a GitHub y enviar el link.

Se espera ver lo siguiente:

Login

Username:
pass:



ACCESO INCORRECTO



ACCESO CORRECTO

Fuentes.

<http://tomcat.apache.org/>

<http://tomcat.apache.org/tomcat-9.0-doc/index.html>

<https://spring.io/tools>

<https://spring.io/guides>

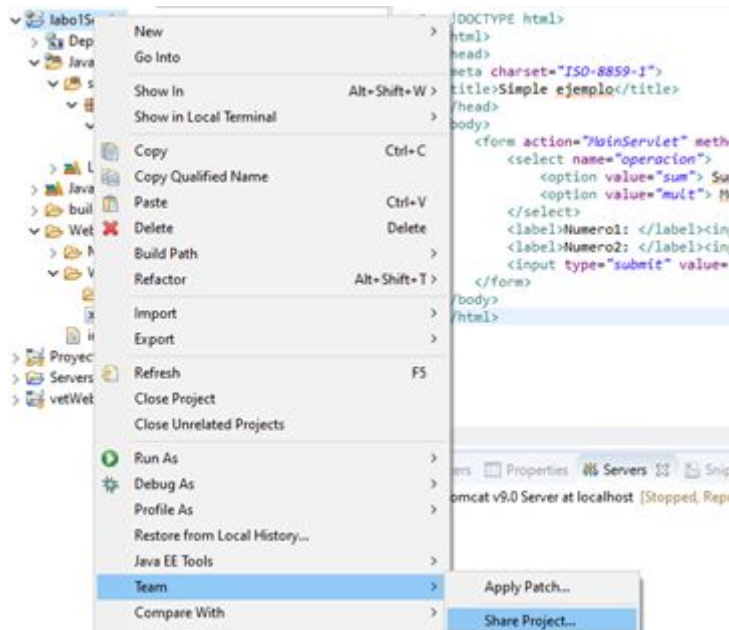
<https://www.techartifact.com/blogs/2012/01/servlet-lifecycle.html/servlet-lifecycle>

<https://www.codejava.net/java-ee/servlet/websocket-annotation-examples>

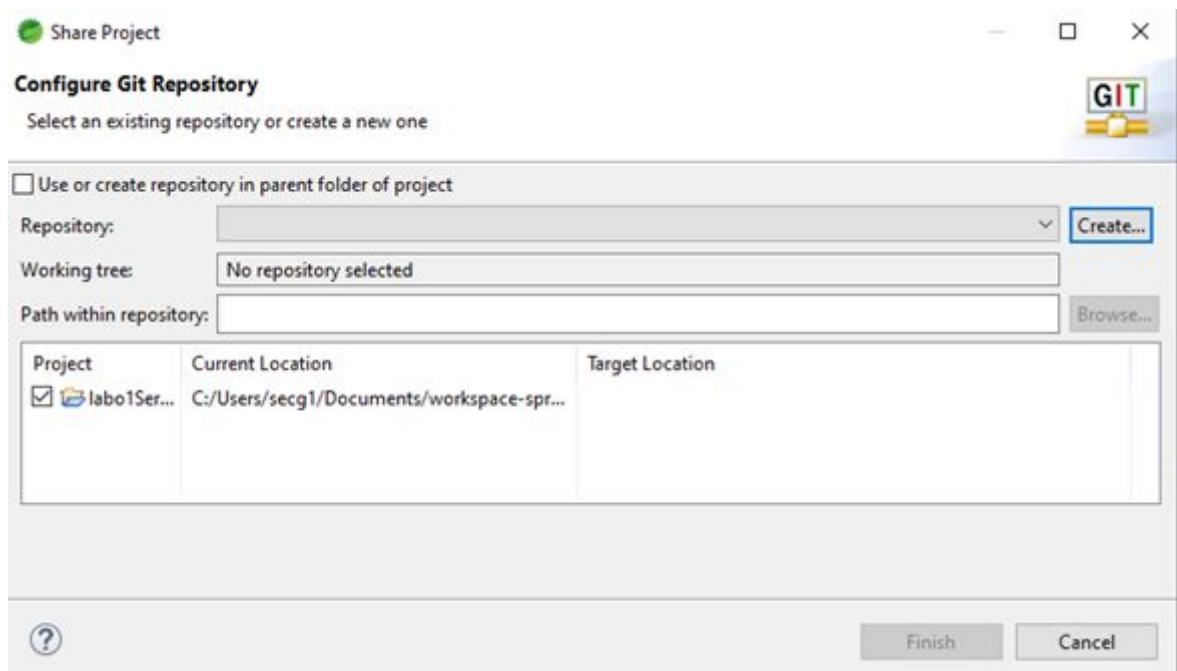
<https://users.dcc.uchile.cl/~jbarrios/servlets/general.html>

Guía de cómo usar Git y Github utilizando el IDE

Cuando se quiera crear un repositorio local, al dar click con el botón derecho sobre el proyecto buscamos la opción Team -> Share Project...



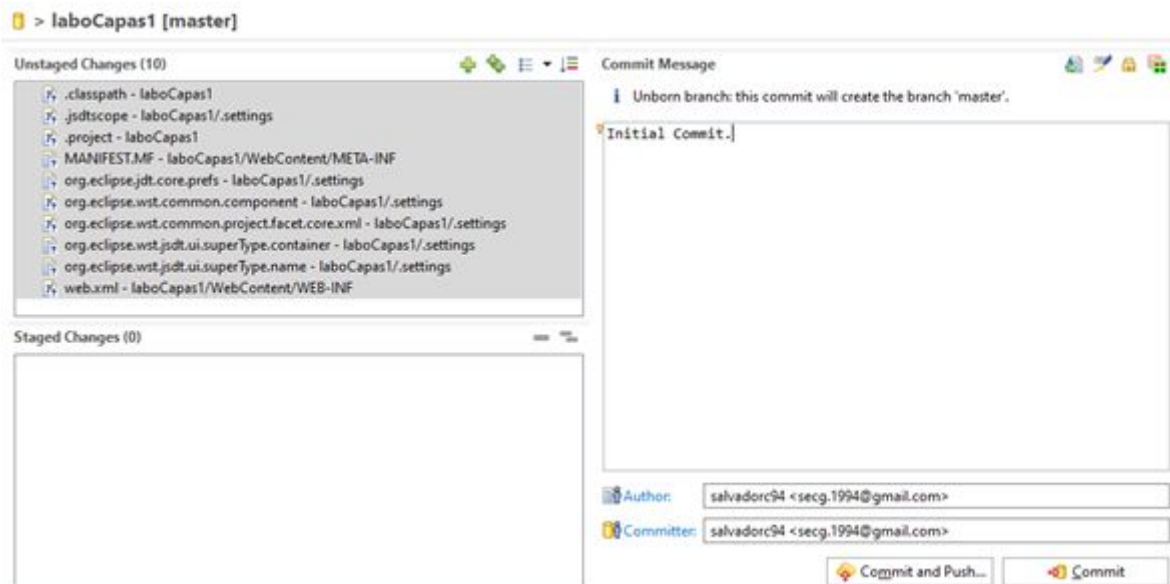
Vamos a crear el repositorio nuevo, por lo que damos click en el botón Create..



El repositorio se nombra igual al que se creará en GitHub "laboCapas1". Y damos en finalizar. Y otra vez en finalizar. Con esto ya tenemos nuestro repositorio pero de manera local y podemos hacer uso de todas las funcionalidades de git como ramas, restaurar una versión anterior, etc.

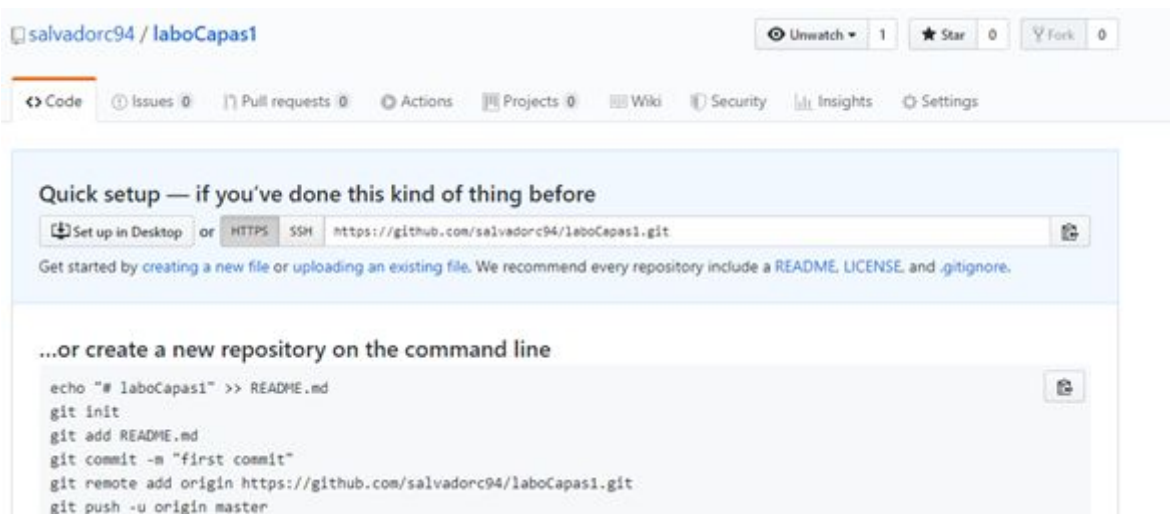
Luego en la pestaña TEAM del proyecto ahora aparecerán más opciones, le damos a COMMIT.

Se abrirá la siguiente ventana:



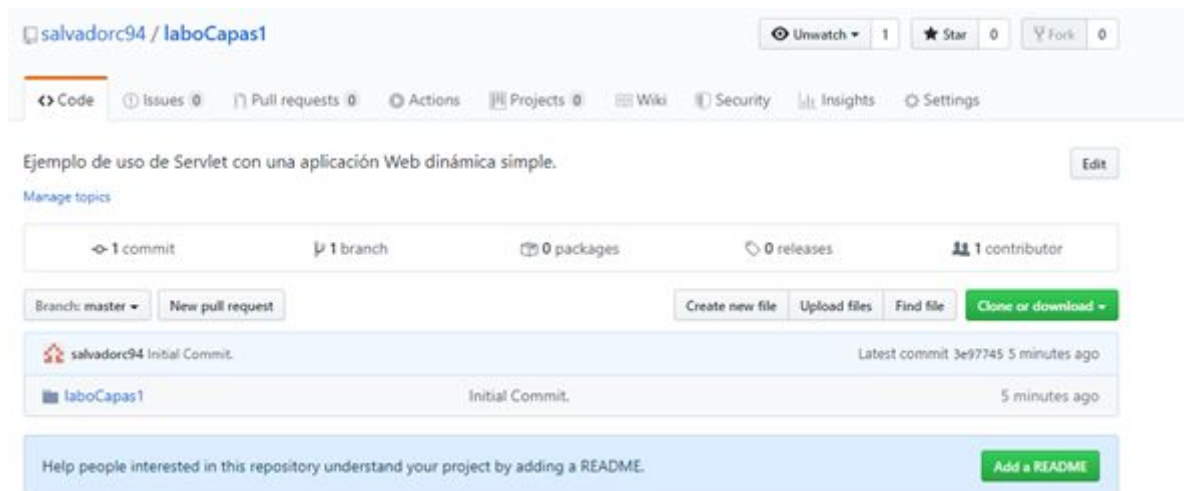
Dentro de esta movemos todos los Unstaged Changes a Staged Changes usando el icono con dos “+”.

Damos clic al botón de Commit y listo, el repositorio local está actualizado. Para hacer push a un repositorio remoto, lo primero que tenemos que hacer es crear el repo vacío en github.



Una vez lo tenemos, vamos de nuevo a la pestaña de Team y buscamos la opción Remote y seleccionamos Push.

Colocamos el link del repositorio vacío que creamos anteriormente y llenamos nuestras credenciales en la parte de autenticación. Damos click a Next, luego Finish y el proyecto debe haber sido pusheado a github.



Para subir los cambios se siguen los mismos pasos mencionados anteriormente, se hace Commit y luego se Pushea remotamente.