Creación del primer servlet

¡Comencemos!

¿Qué aprenderás?

- Entender cómo se compone un servlet.
- Compilar un servlet.
- Probar servlets en el contenedor.
- Conocer los métodos de los servlets.

Introducción

En una arquitectura web, la comunicación entre cliente y servidor es crucial para un funcionamiento colaborativo, y dentro de la tecnología JEE un servlet es uno de los principales actores. Ya estudiamos que es un servlet y cual es su ciclo de vida, por lo que ahora es tiempo de empezar a implementarlos para descubrir poco a poco cuales son sus funciones y que nos permiten hacer.

Composición de un servlet

Los servlets se construyen basados en la herencia de su clase padre Http Servlets, clase pública y abstracta que permite que una clase herede los métodos y atributos que hacen que se convierta en un servlets. La clase que herede de esta clase abstracta, debe sobreescribir al menos:

- Método doGet.
- Método doPost.
- Método doDelete.

Esta regla es de oro y, teniendo esto en mente, comenzaremos a generar el primer servlet.

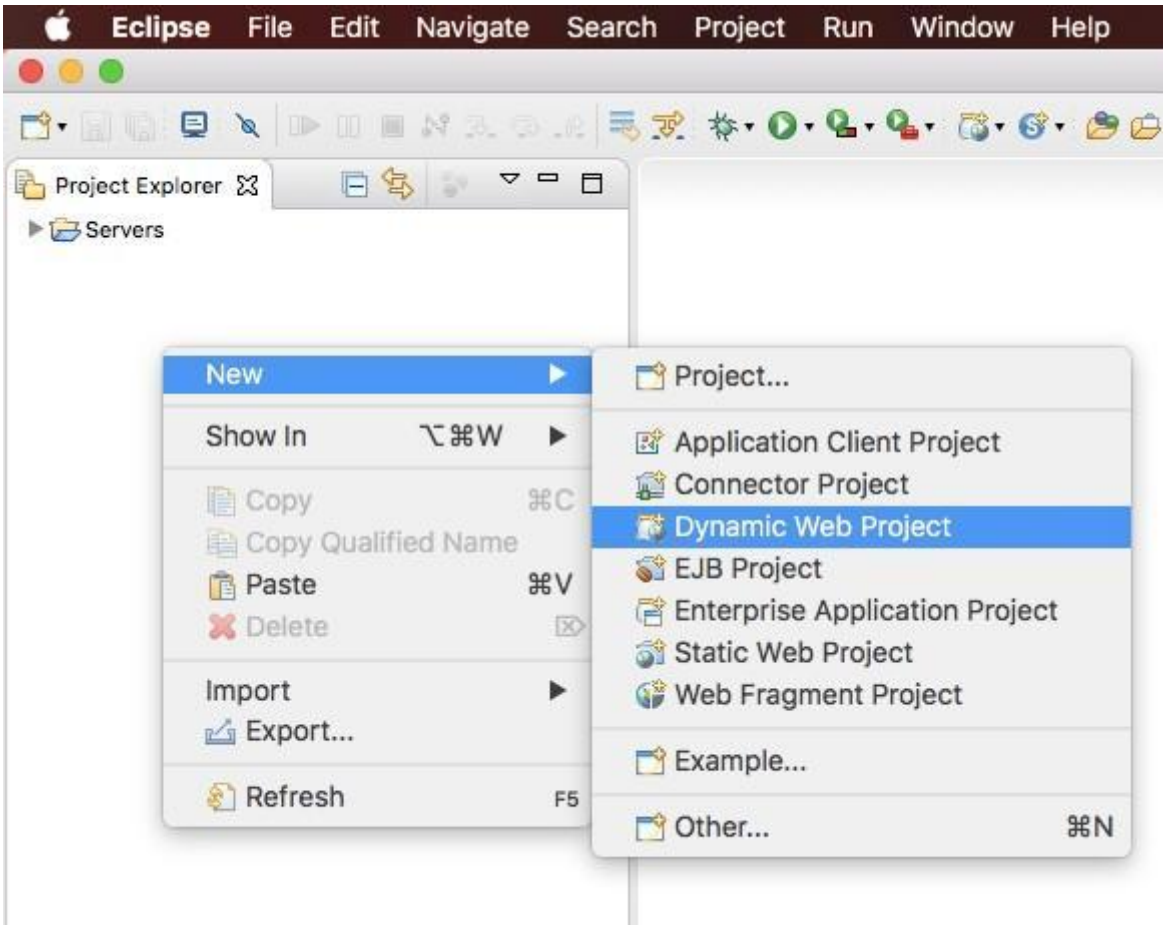

Imagen 1. Creando un nuevo proyecto.
Fuente: Desafío Latam

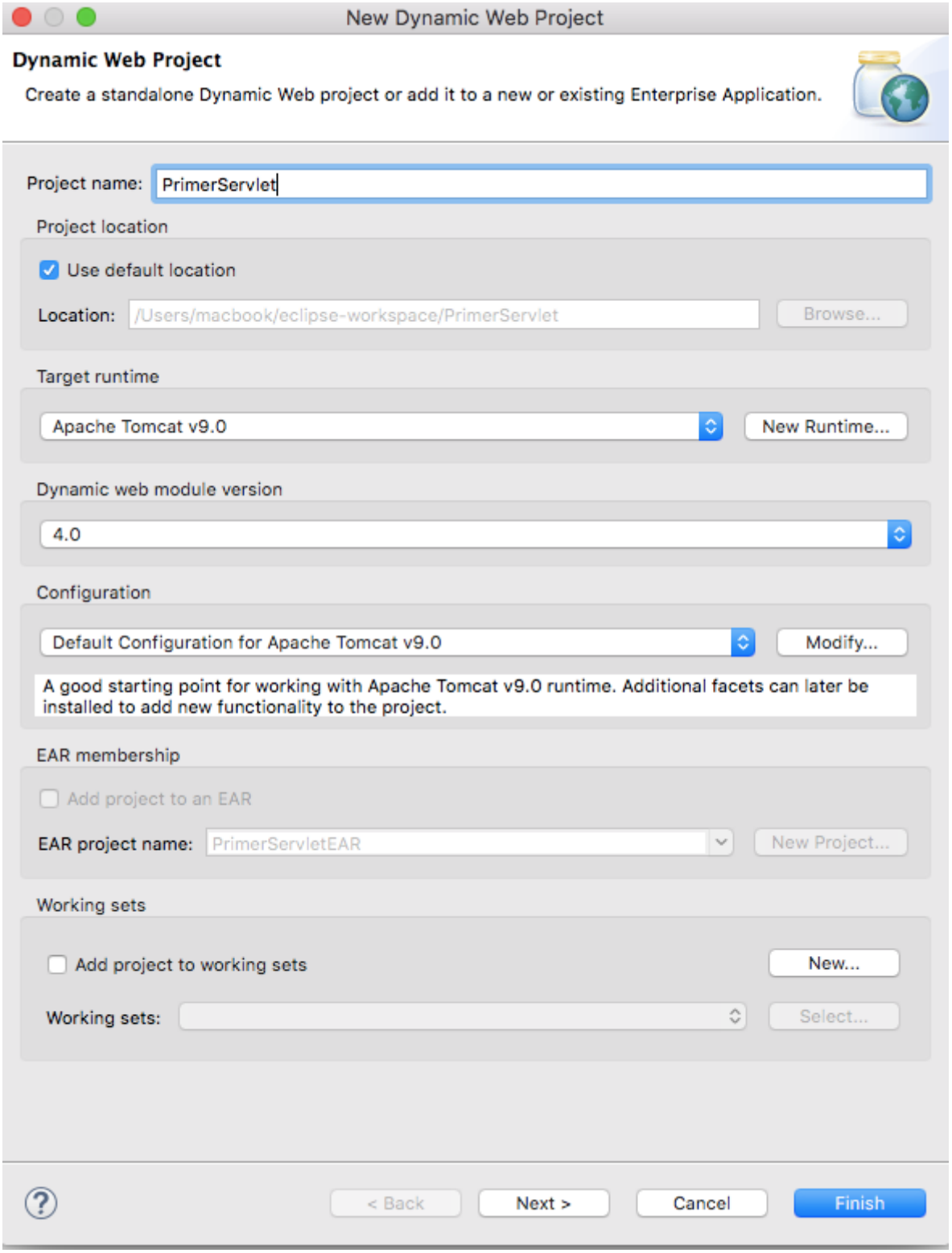

Imagen 2. Lo llamamos "PrimerServlet".
Fuente: Desafío Latam

Creación de un servlet

1. Creamos un nuevo proyecto de tipo *Dinamic Web Project* y nombramos al mismo como *PrimerServlet*.

2. Avanzamos en el *wizard* con *next* y *finish*. Por el momento dejamos las opciones de *web.xml* por defecto.

3. Al final Eclipse nos genera una estructura de Proyecto, la cual usaremos como base para nuestro trabajo.

Por el momento, la carpeta más importante para nosotros es la *Java Resources*, que contiene el código fuente de nuestro programa, la cual se compone de:

- La carpeta **src**, que aloja las clases java y la estructura de paquetes de la aplicación.
- La carpeta **libraries**, que mantiene las librerías utilitarias.
- La carpeta **build**, que mantiene las clases autogeneradas.

4. Luego de generar el Proyecto, vamos a crear nuestra primera clase. Para hacer esta labor, tenemos que seleccionar el nombre del proyecto con botón derecho, y presionar *new class*.

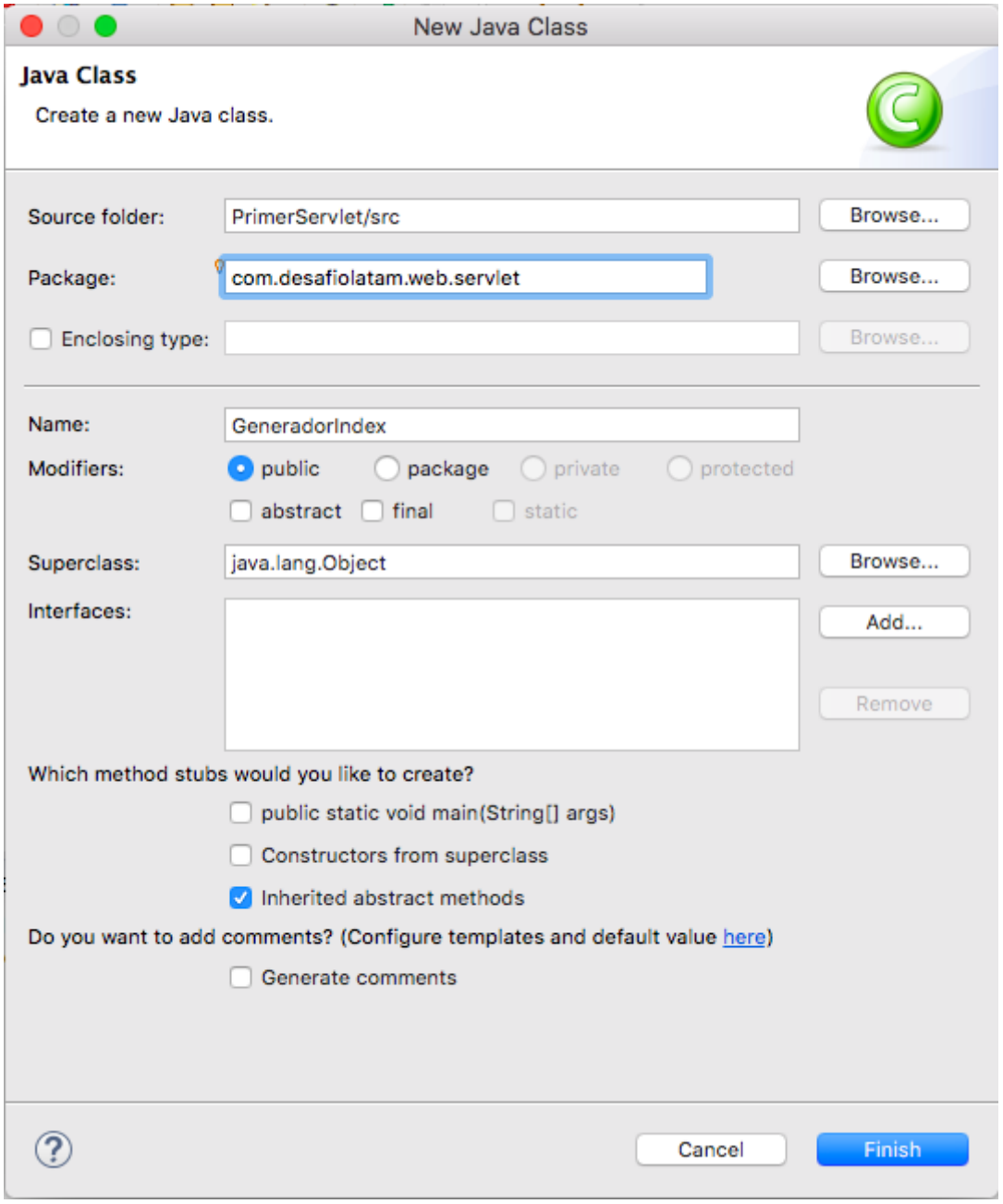

Imagen 3. Creamos una nueva clase.
Fuente: Desafío Latam

5. Al finalizar, Eclipse nos genera la clase y abre el archivo en el área de código.

```
package com.desafiolatam.web.servlet;

public class GeneradorIndex{

}
```

Hasta ahora, solo tenemos una clase java de nombre *GeneradorIndex.java* la cual no hace nada, no tiene ningún servicio interesante ni nada que lo haga parecer un *servlet*. Aquí es donde comienza la verdadera creación de este componente.

Anteriormente, mencionamos que toda clase que quiera convertirse en servlet debe heredar de la abstracta padre *HttpServlet*, para luego implementar algunos de sus métodos. En el siguiente código, vamos a analizar un servlet sencillo que cuenta con todo lo necesario para proveer servicios y generar una página web dinámica.

```
package com.desafiolatam.web.servlet;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.logging.Logger;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
@WebServlet("/inicio")
public class GeneradorIndex extends HttpServlet {

   private static final long serialVersionUID = 1L;
   Logger milog = Logger.getLogger(Saludo.class.getName());
   @Override
   protected void doGet(HttpServletRequest req, HttpServletResponse resp) {
         try {
               PrintWriter pw = resp.getWriter();
               pw.println("<h1>Hola mundo</h1>");
         } catch (IOException e) {
               milog.severe(e.getMessage());
         }
   }
}
```

Lo primero que nos debe llamar la atención con el código fuente expuesto, es que aparece una sentencia en la declaración de la clase en la línea posterior a los *import*:

```
    @WebServlet("/inicio")
```

Es una anotación, que le indica al contenedor de servlets que esta clase es un servlet que debe ser accedido mediante URL y su path contenedor es */inicio*. Recordemos que un servlet es un componente web que será accedido por internet, por lo cual para ubicar el elemento debemos llamarlo mediante la dirección url del recurso.

El primer método protected es heredado desde la clase *HttpServlet* y su nombre es *doGet()*. Por el momento sabemos que este método permite devolver una respuesta al navegador del cliente mediante el método *get*.

```
    protected void doGet(HttpServletRequest request, HttpServletResponse
response)
```

El resto del código es java puro, simplemente estamos haciendo un pequeño "Hola Mundo" a través del *servlet*. Algo a destacar es la generación de código *html* en el mismo archivo java.

Puedes ver que al imprimir código estamos generando etiquetas *html*, las cuales el navegador puede interpretar directamente. El siguiente fragmento de código lo explica mejor:

```
    PrintWriter pw = resp.getWriter();
    pw.println("<h1>Hola mundo</h1>");
```

Similar al ***System.out.println();***

Creamos la instancia writer de la clase *PrintWriter* y le enviamos la variable response junto al método *getWriter()* la cual recibimos mediante los parámetros de entrada del método. Teniendo la instancia *writer*, podemos utilizar el método *println* y formatear una estructura de página web sin problemas, ya que el servlet se encargará de enviarlo al navegador y este podrá interpretarlo.

Para ver el resultado en Eclipse selecciona el proyecto con botón derecho y ve a la opción *Run As* como se muestra en la imagen.

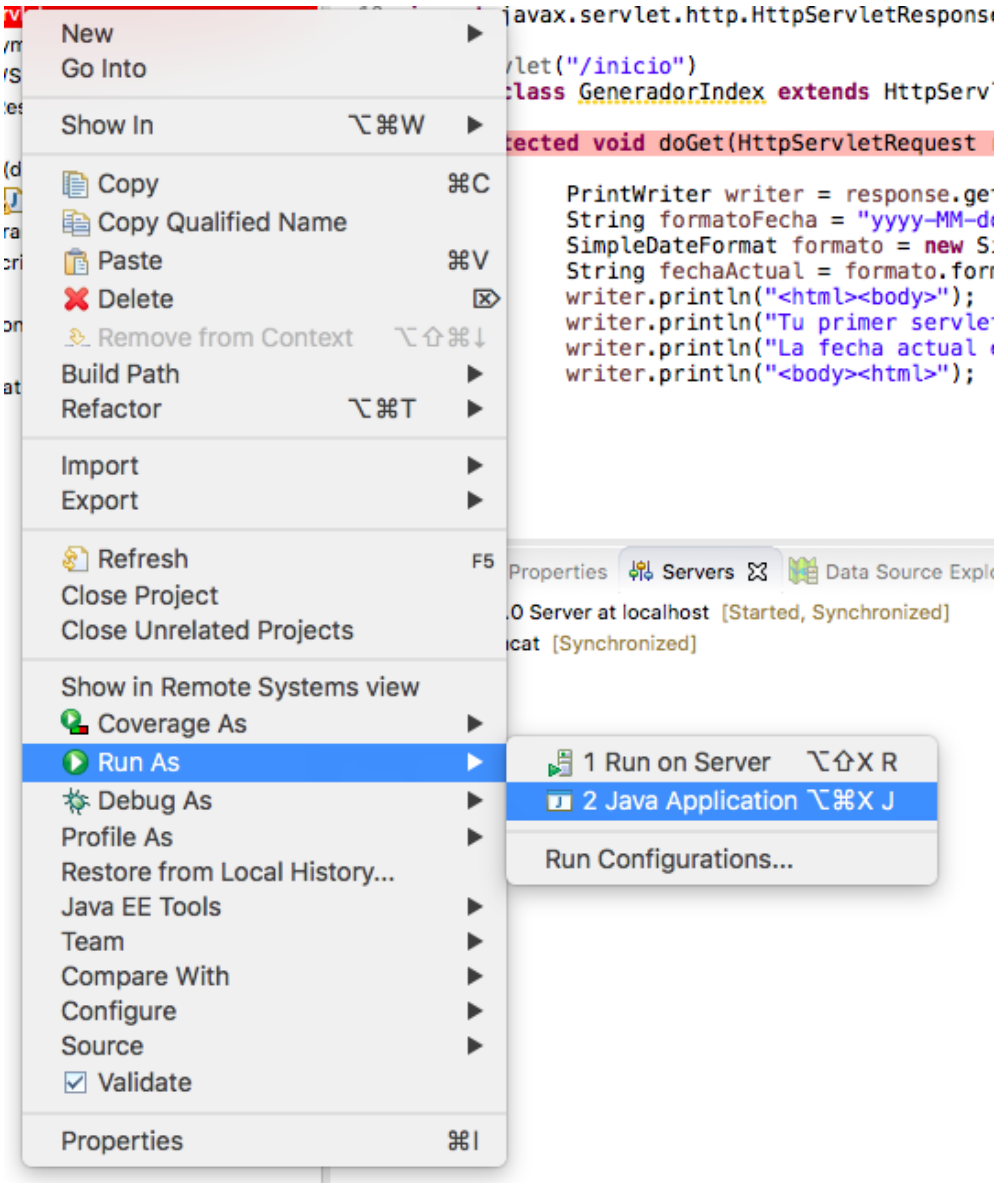

Imagen 4. Ejecutar código.
Fuente: Desafío Latam

Ya deberías tener seleccionado el apache Tomcat desde el ejercicio anterior por lo que empezará a levantar el server y, según la configuración de Eclipse, pueden pasar dos cosas:

- Se abrirá un navegador en el mismo IDE
- La consola dirá solo que comenzó a trabajar

Si ocurre lo primero, abrirá un navegador y mostrará una pantalla de error:

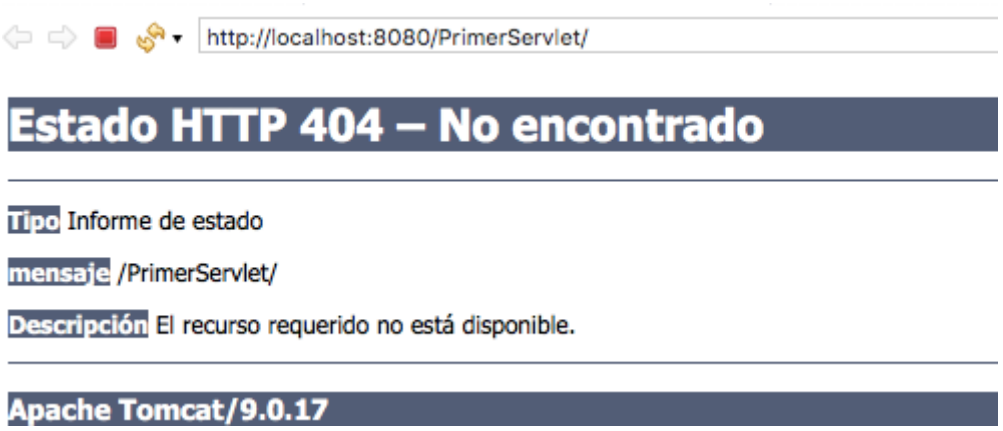

Imagen 5. Error 404.
Fuente: Desafío Latam

¿Por qué pasa esto? ¿Recuerdas la anotación que vimos al principio del ejercicio?

```
@WebServlet("/inicio")
```

Aquí se ve la utilidad de tal sentencia. Estamos intentando llamar a nuestro servlet mediante la dirección:

```
http://localhost:8080/PrimerServlet
```

Pero solo estamos llegando hasta la puerta de entrada del servlet, no le estamos diciendo en ningún lado a qué servicio (o método de clase) queremos acceder. Para poder ejecutar nuestro servlet, agrega a la dirección el path indicado en la anotación @WebServlet:

```
http://localhost:8080/PrimerServlet/inicio
```