INSTITUTO SAN ISIDRO

Curso de Java a distancia

Clase 35: Hilos en Java - método synchronized

Los métodos sincronizados en Java solo pueden tener un hilo ejecutándose dentro de ellos al mismo tiempo.

Son necesarios cuando un método accede a un recurso que puede ser consumido por un único proceso.

Para evitar que un algoritmos sea ejecutado por más de un hilo en forma simultanea, Java nos permite definir un método con el modificador: synchronized.

Cuando un método se lo define synchronized luego solo un hilo puede estar ejecutándolo en un mismo momento.

Problema:

Confeccionar un programa que imprima la cantidad de archivos de texto de un directorio que contienen una determinada palabra. Efectuar la búsqueda dentro de un hilo.

Lanzar la búsqueda de dos palabras en dos hilos separados.

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;

public class BuscarPalabra implements Runnable {
   private String palabra;
   private Thread hilo;
   private int cant;

   public BuscarPalabra(String palabra) {
      this.palabra = palabra;
      hilo = new Thread(this);
      hilo.start();
      while (hilo.isAlive())
         ;
      System.out.println("La palabra " + palabra + " se encuentra contenida en " +
                  cant + " archivos");
   }

   @Override
   public void run() {
      File ar = new File("C:\\documentos\\");
      String[] directorio = ar.list();
```

```
        for (String arch : directorio) {
            if (tiene(arch))
                cant++;
        }
    }

    private synchronized boolean tiene(String archivo) {
        boolean existe = false;
        try {
            FileReader fr = new FileReader("c:\\documentos\\" + archivo);
            BufferedReader br = new BufferedReader(fr);
            String linea = br.readLine();
            while (linea != null) {
                if (linea.indexOf(palabra) != -1)
                    existe = true;
                linea = br.readLine();
            }
            br.close();
            fr.close();
        } catch (IOException e) {
            System.out.println(e.toString());
        }
        return existe;
    }

    public static void main(String[] ar) {
        new BuscarPalabra("rojo");
        new BuscarPalabra("verde");

    }

}
```

La clase 'BuscarPalabra' define como atributos la palabra a buscar, un hilo y el contador:

```
public class BuscarPalabra implements Runnable {
    private String palabra;
    private Thread hilo;
    private int cant;
```

En el constructor recibimos la palabra a buscar, creamos el hilo y mientras el mismo no finalice no se procede a mostrar el contador de palabras:

```
    public BuscarPalabra(String palabra) {
        this.palabra = palabra;
        hilo = new Thread(this);
        hilo.start();
        while (hilo.isAlive())
            ;
        System.out.println("La palabra " + palabra + " se encuentra contenida en " +
                        cant + " archivos");
    }
```

En el método run que se inicia al llamar al método 'start' del hilo procedemos a obtener todos los archivos del directorio a procesar y llamamos al metodo 'tiene' para cada uno de los archivos de texto:

```
    @Override
    public void run() {
        File ar = new File("C:\\documentos\\");
        String[] directorio = ar.list();
        for (String arch : directorio) {
```

```
            if (tiene(arch))
                cant++;
        }
    }
```

El método 'tiene' es el método sincronizado, el cual evita que dos hilos puedan en forma simultánea acceder al sistema de archivos.

Abrimos, leemos el archivo de texto y buscamos en cada línea si tiene la palabra:

```
    private synchronized boolean tiene(String archivo) {
        boolean existe = false;
        try {
            FileReader fr = new FileReader("c:\\documentos\\" + archivo);
            BufferedReader br = new BufferedReader(fr);
            String linea = br.readLine();
            while (linea != null) {
                if (linea.indexOf(palabra) != -1)
                    existe = true;
                linea = br.readLine();
            }
            br.close();
            fr.close();
        } catch (IOException e) {
            System.out.println(e.toString());
        }
        return existe;

    }
```

En la main creamos dos objetos de la clase 'BuscarPalabra':

```
    public static void main(String[] ar) {
        new BuscarPalabra("rojo");
        new BuscarPalabra("verde");

    }
```

Sincronización a nivel de bloques.

Una variante de la sincronización de métodos es definir la sincronización a nivel de bloque, es decir que no afecte a todo el método. La sintaxis luego queda:

```
    private boolean tiene(String archivo) {
        boolean existe = false;
        synchronized (this) {
            try {
                FileReader fr = new FileReader("c:\\documentos\\" + archivo);
                BufferedReader br = new BufferedReader(fr);
                String linea = br.readLine();
                while (linea != null) {
                    if (linea.indexOf(palabra) != -1)
                        existe = true;
                    linea = br.readLine();
                }
                br.close();
                fr.close();
            } catch (IOException e) {
                System.out.println(e.toString());
            }
        }
```

```
    return existe;
}
```

Instalación del "Eclipse IDE for Java EE Developers" y el servidor "Apache Tomcat"

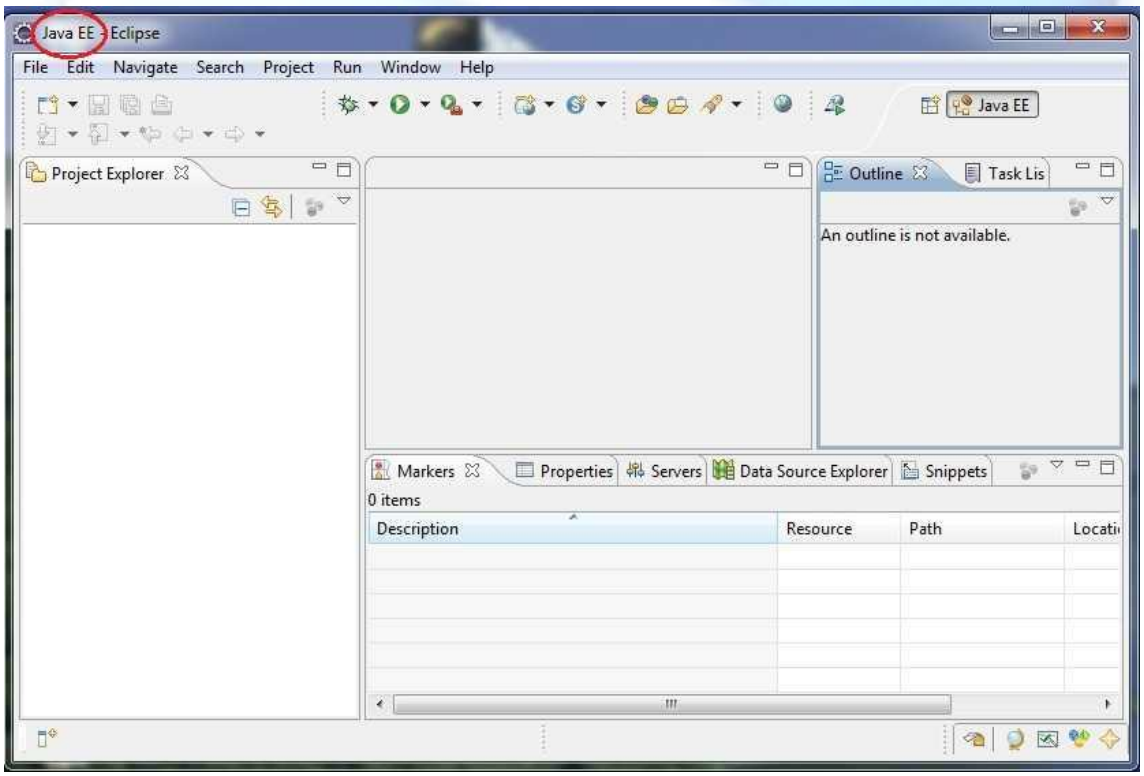
"Eclipse IDE for Java EE Developers"

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.

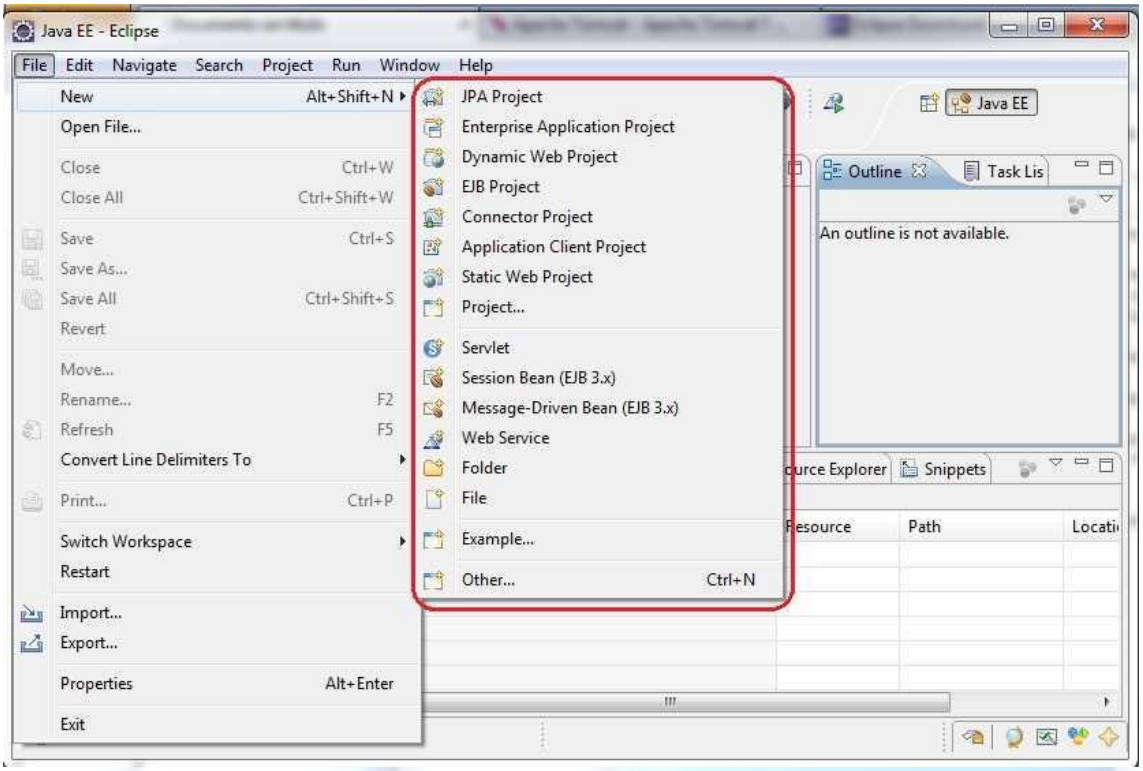
La versión que debemos descargar es Eclipse IDE for Java EE Developers, como podemos ver el tamaños es mayor que la versión que hemos utilizado hasta este momento (Eclipse IDE for Java Developers)

Podemos crear otra carpeta con otro nombre para no perder la versión de Eclipse que hemos utilizado para el desarrollo de aplicaciones de escritorio (swing)
Creemos la carpeta eclipsej2ee y dentro de la misma descomprimamos el entorno de Eclipse que acabamos de descargar "Eclipse IDE for Java EE Developers".

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):



Pero 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:

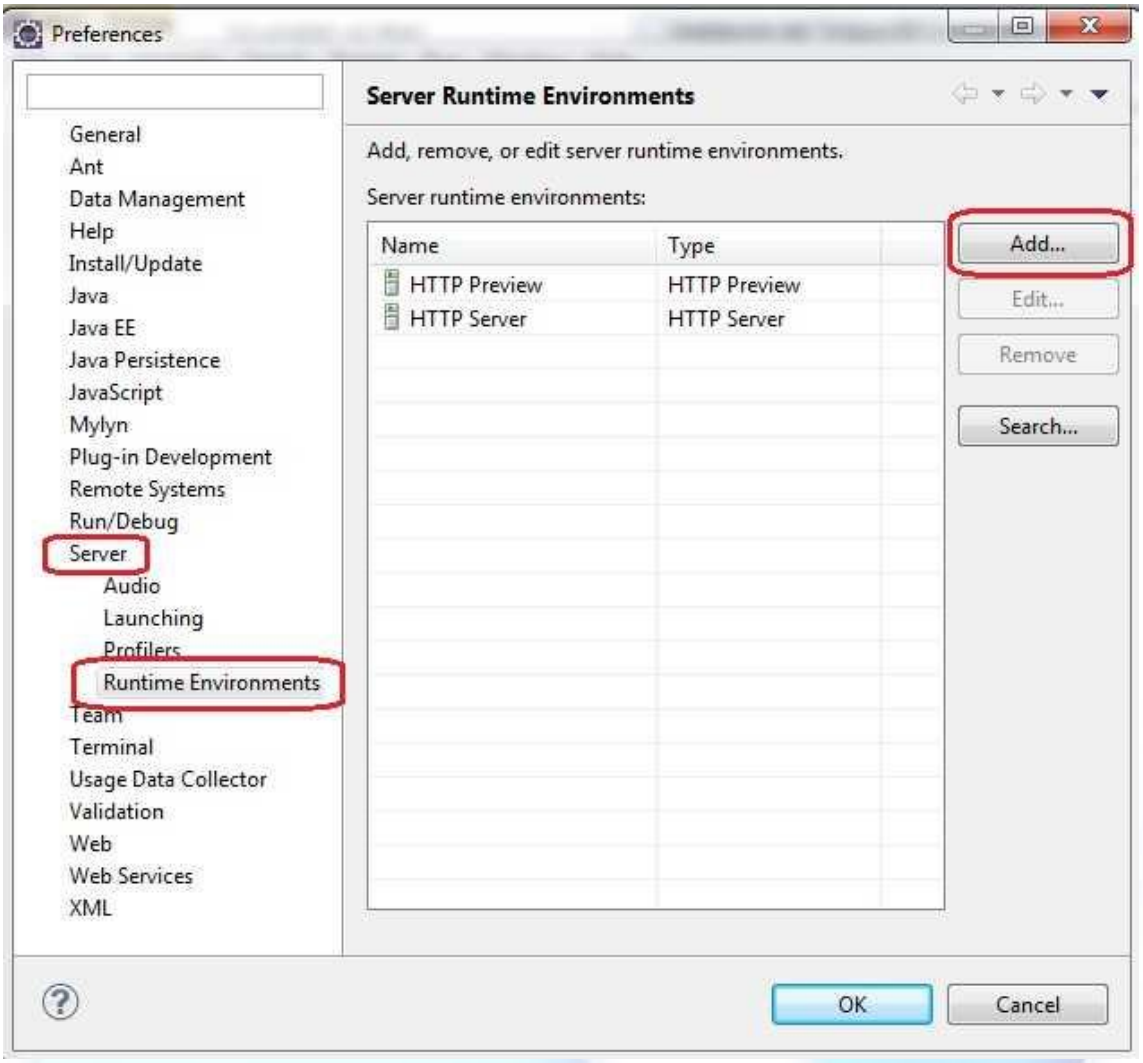


"APACHE TOMCAT"

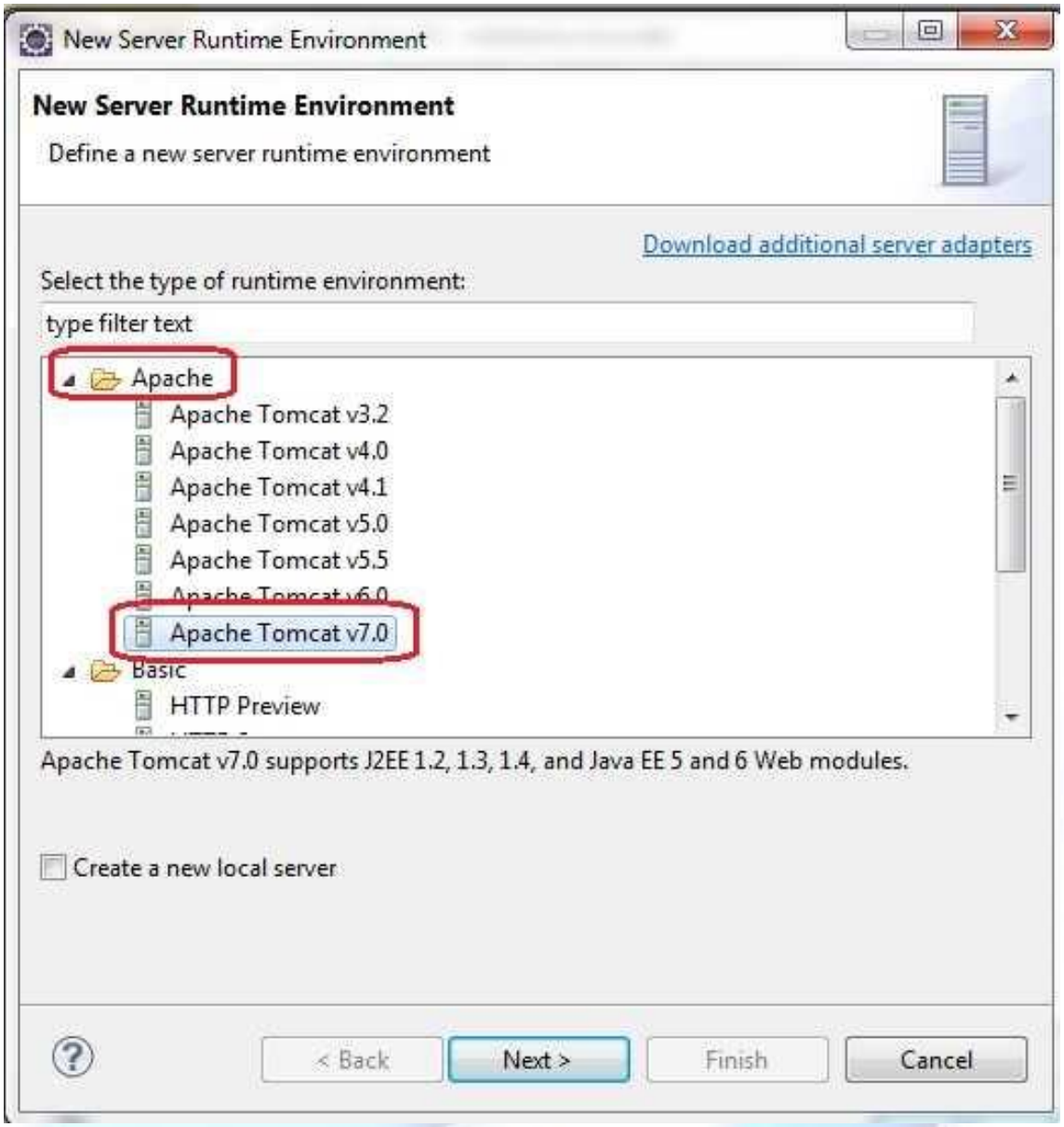
Ahora pasaremos a instalar un servidor web "Apache Tomcat" que nos permitirá ejecutar servlet y páginas dinámicas.

Podemos descargar el "Apache Tomcat" de aquí (descargar el archivo Binary Distributions Core 32-bit Windows zip) y descomprimirlo en una carpeta.

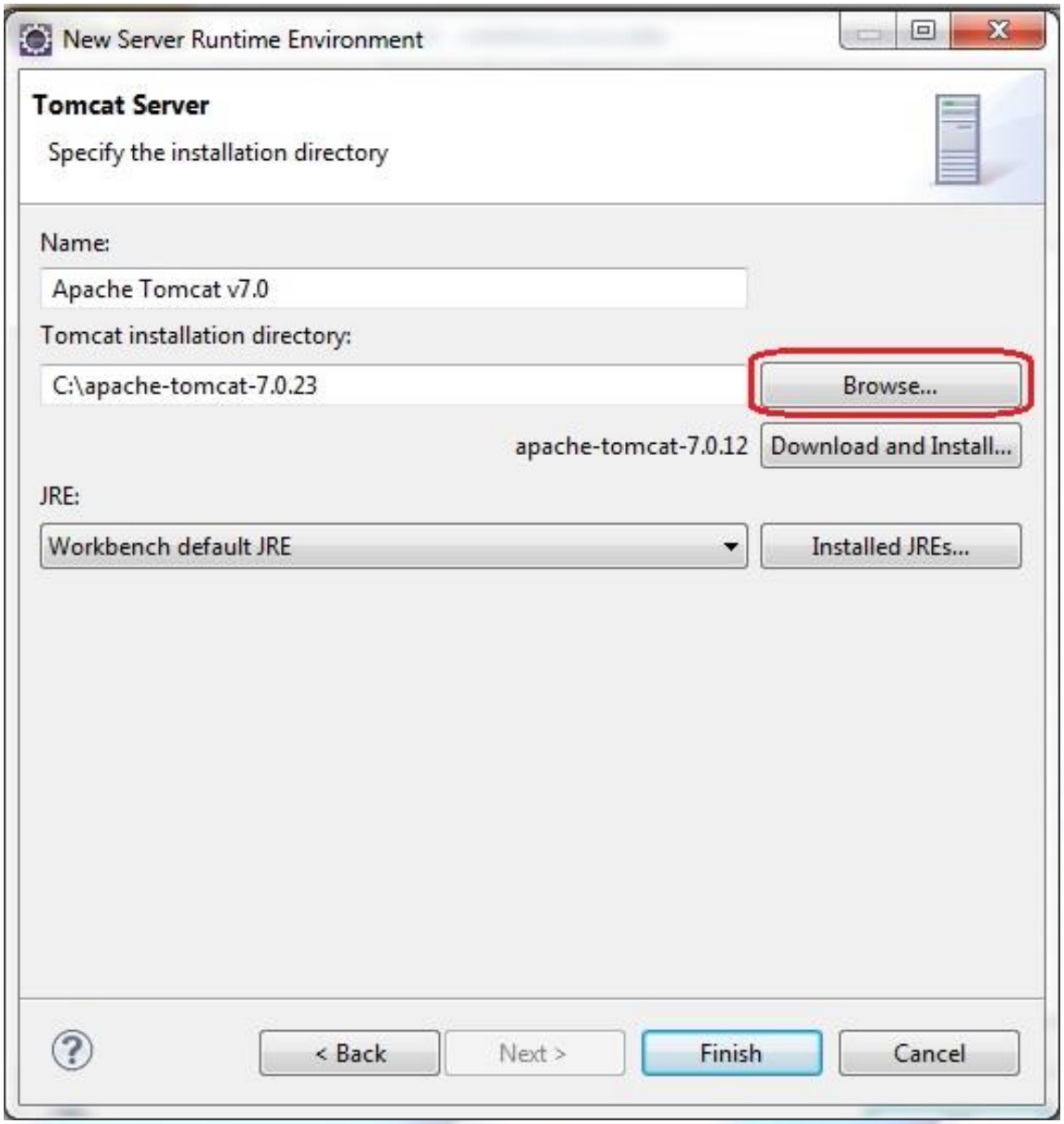
Una vez descomprimido procedemos a registrarlo en Eclipse. Desde el menú de opciones seleccionamos Window -> Preferences y en el diálogo que aparece debemos seleccionar Server -> Runtimes Environments y presionar el botón "Add...":



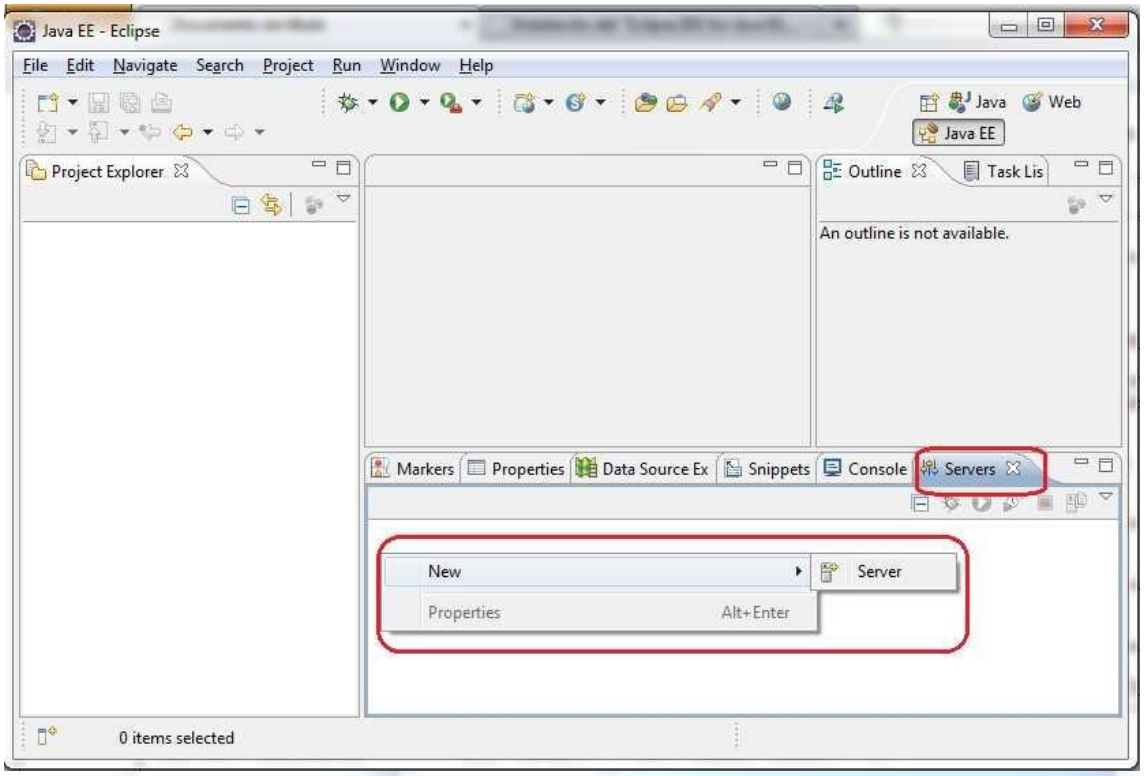
En el nuevo diálogo que aparece seleccionamos de la carpeta "Apache" la versión 7 que es la que acabamos de descargar y descomprimir en una carpeta de nuestro disco duro:



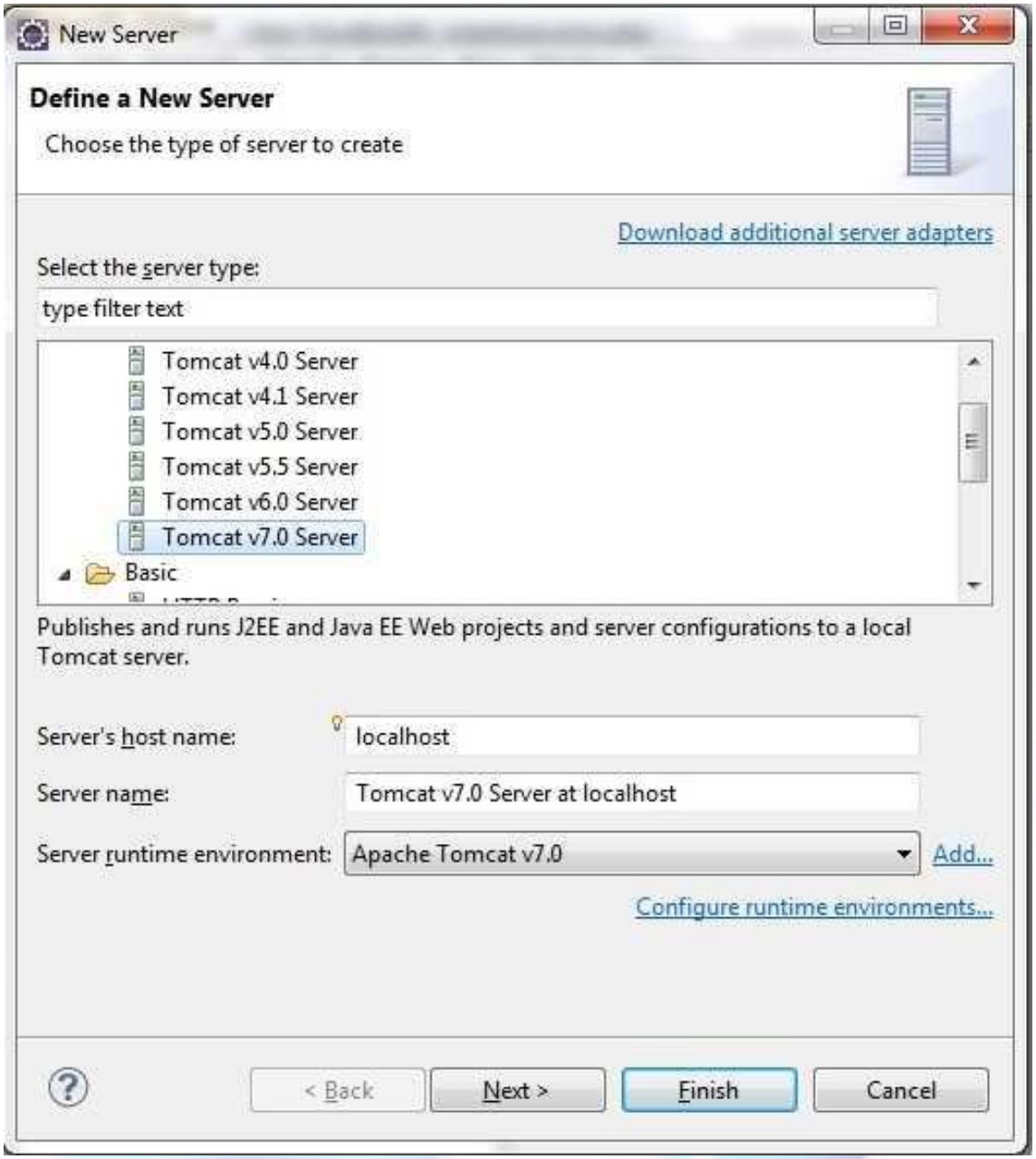
En el último diálogo que aparece debemos seleccionar la carpeta donde hemos descomprimido el "Apache Tomcat" y presionar el botón "Finish:



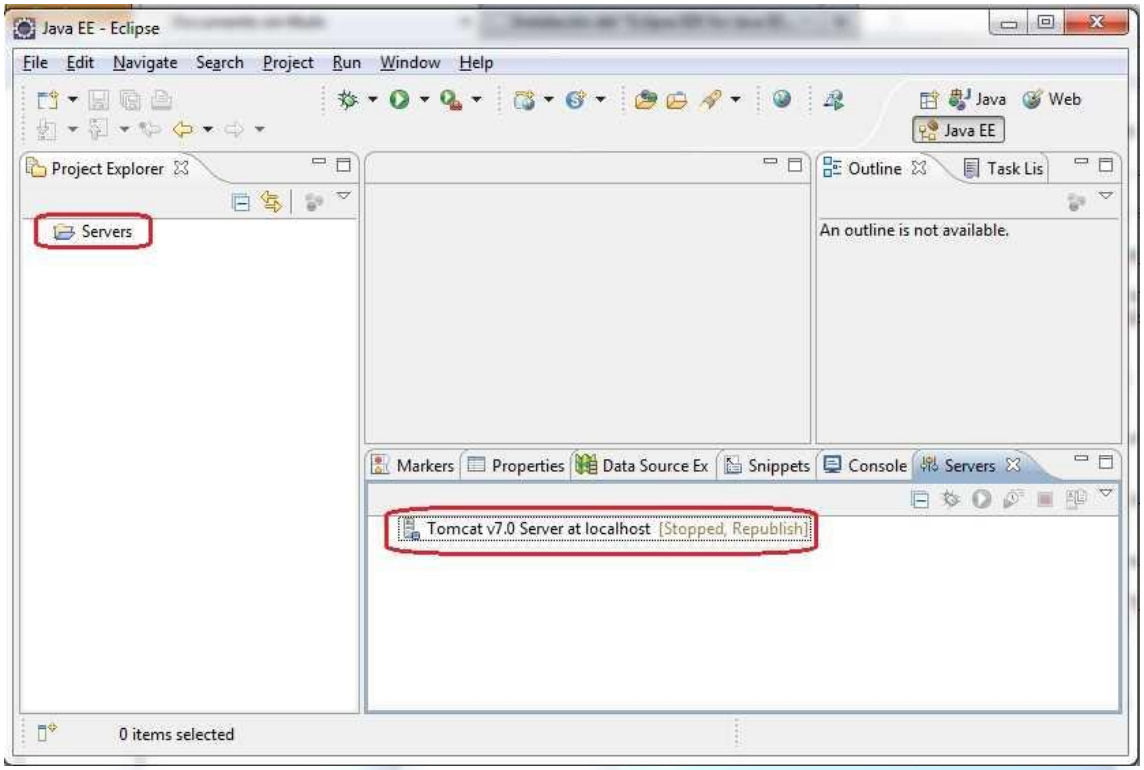
Ahora debemos iniciar los servicios del servidos "Apache Tomcat" para podes hacer aplicaciones que hagan peticiones.
Para arrancar el Tomcat debemos presionar el botón derecho del mouse sobre la ventana "Server", si no parece esta ventana podemos activarla desde el menú (Window -> Show View -> Servers) 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:

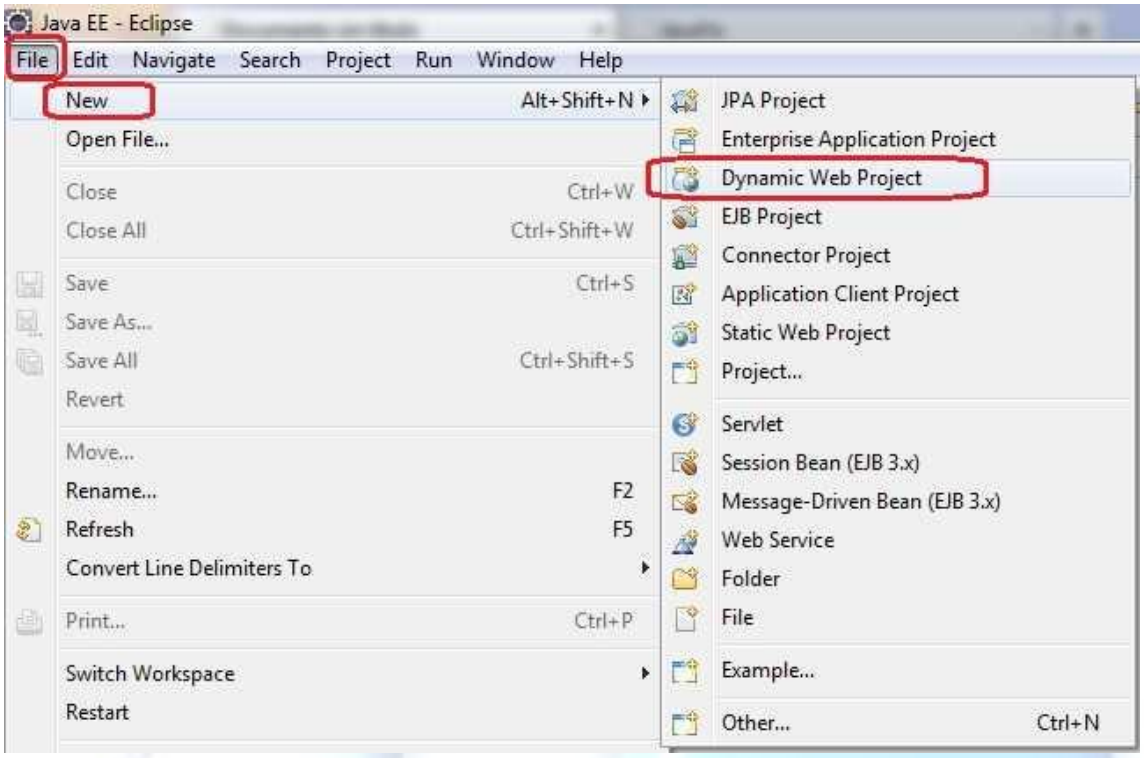


Servlet

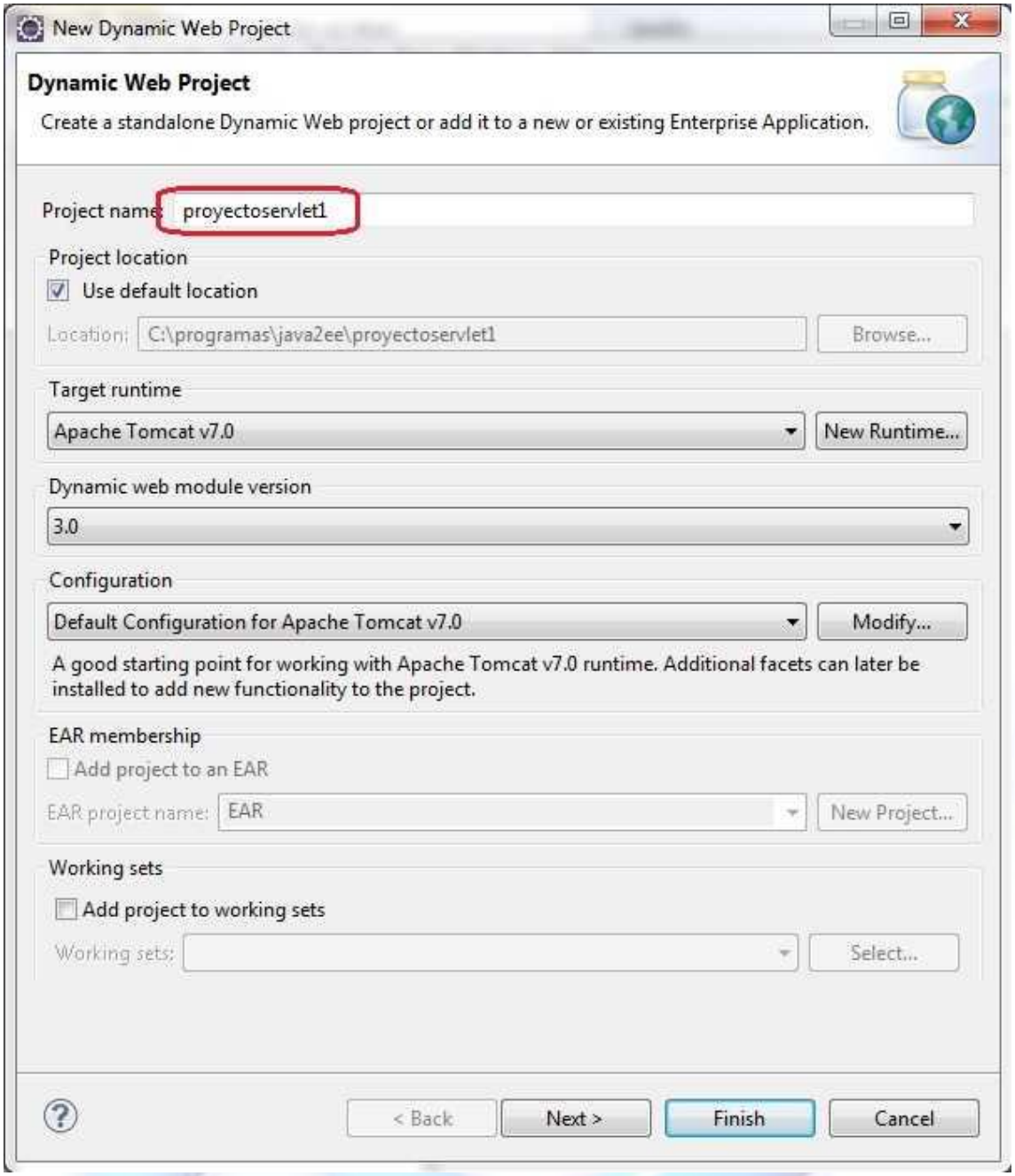
Un servlet es una clase que se ejecuta en el contexto de un servidor web (en nuestro caso el Apache Tomcat)
Un servlet se ejecuta en un servidor web y el resultado de ejecución viaja por internet para ser visualizado en un navegador web (normalmente un servlet genera HTML, pero puede generar otros formatos de archivos)

Veremos los pasos en Eclipse para crear un servlet mínimo que nos muestre un mensaje y los números del 1 al 10000.

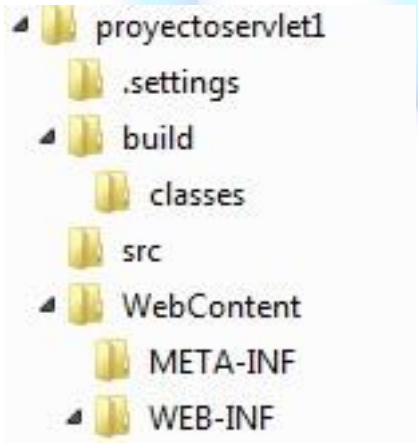
Desde el menú de opciones seleccionamos File -> New -> Dynamic Web Project:



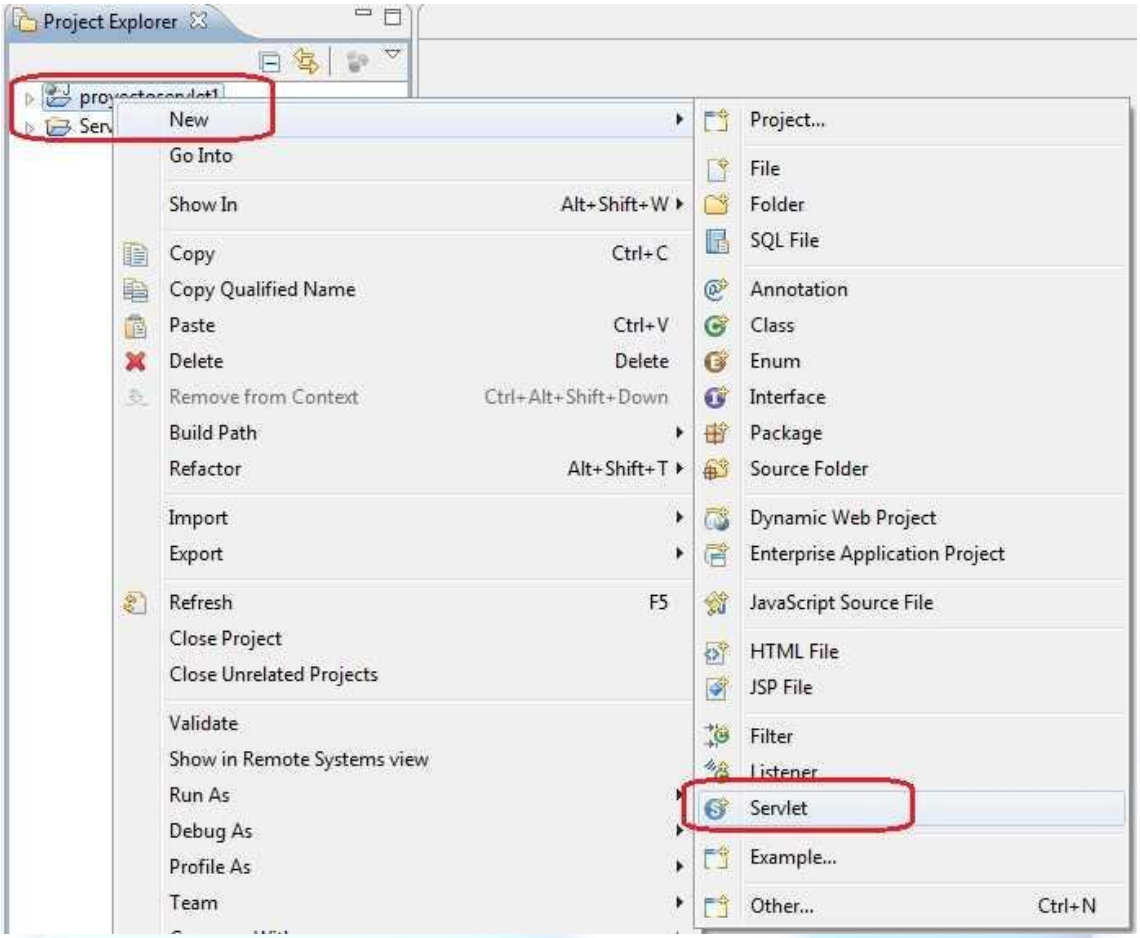
En el diálogo siguiente especificamos el nombre del proyecto (en nuestro caso le llamaremos proyectoservlet1) y presionamos el botón "Finish":



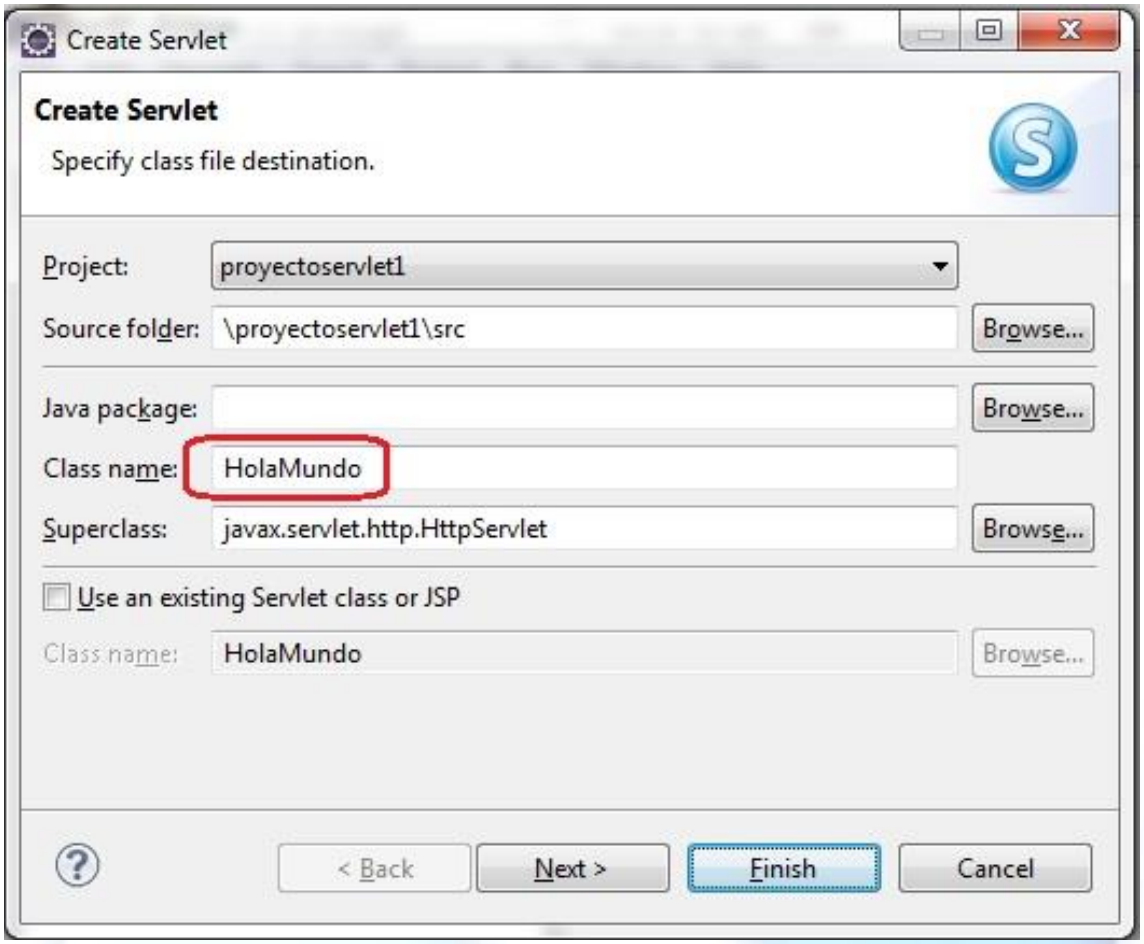
El Eclipse nos crea una serie de carpetas y archivos donde alojaremos los servlet:



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;

   /**
    * @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
       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>");
    }

    /**
     * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
     */
    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/proyectoservlet1/HolaMundo):

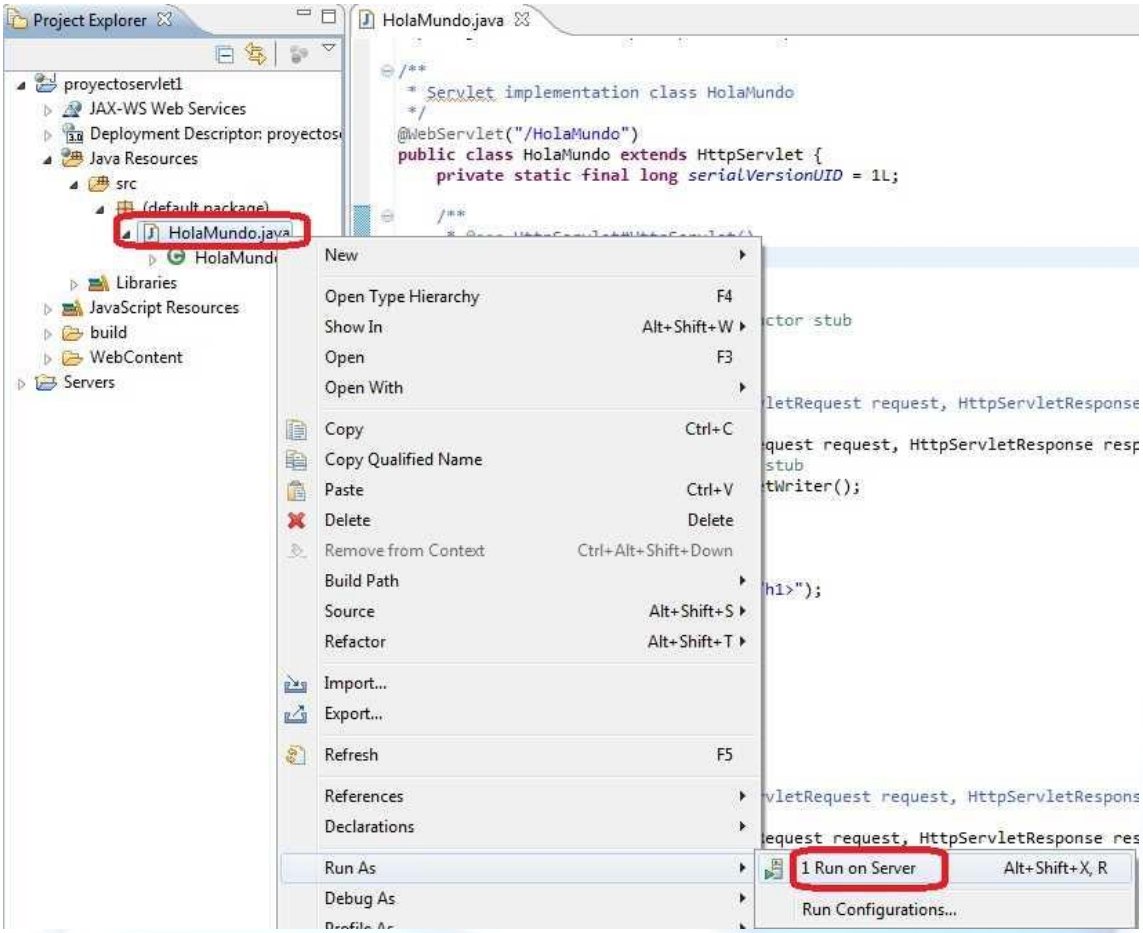
```
@WebServlet("/HolaMundo")
```

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:

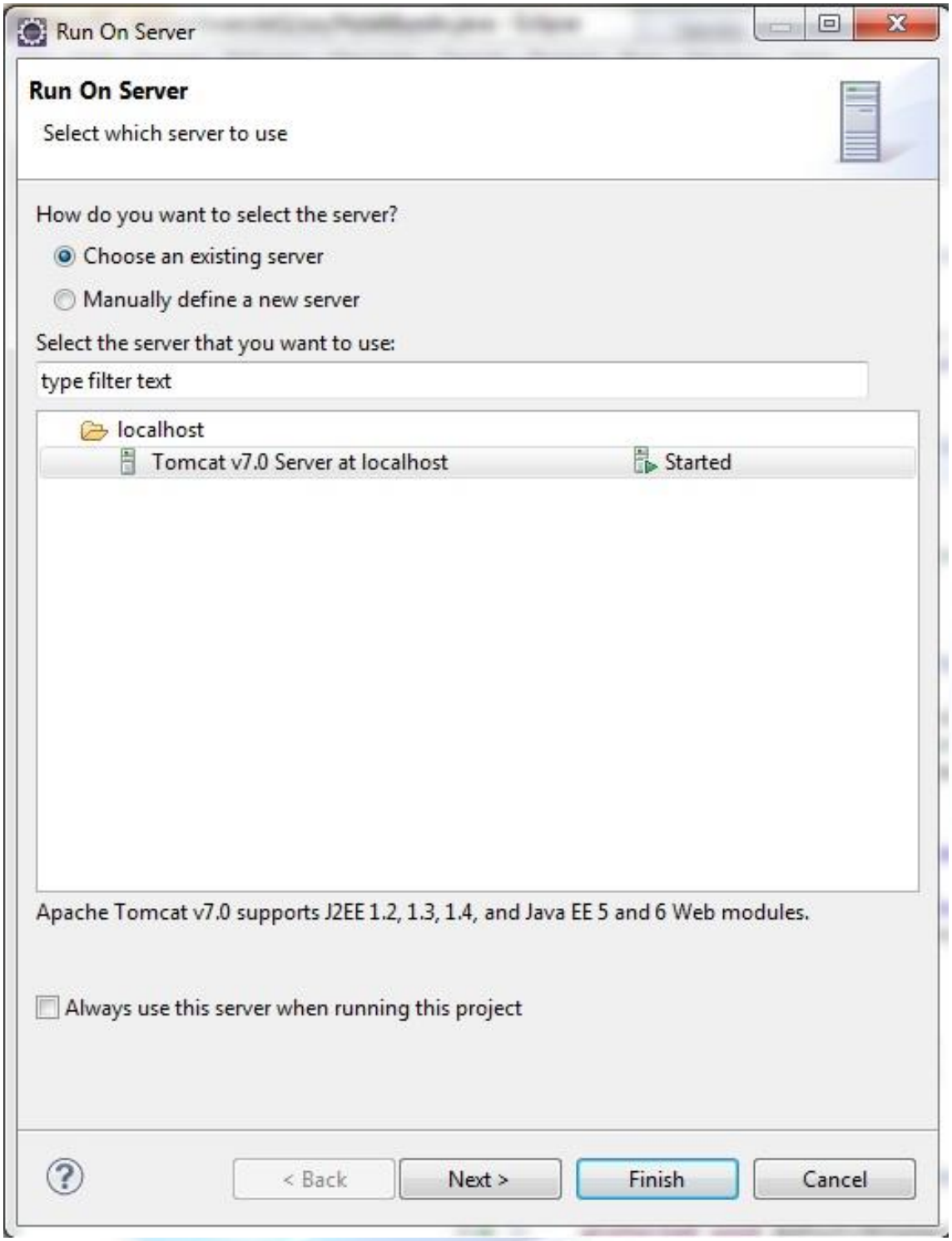
```
        PrintWriter out = response.getWriter();
```

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.

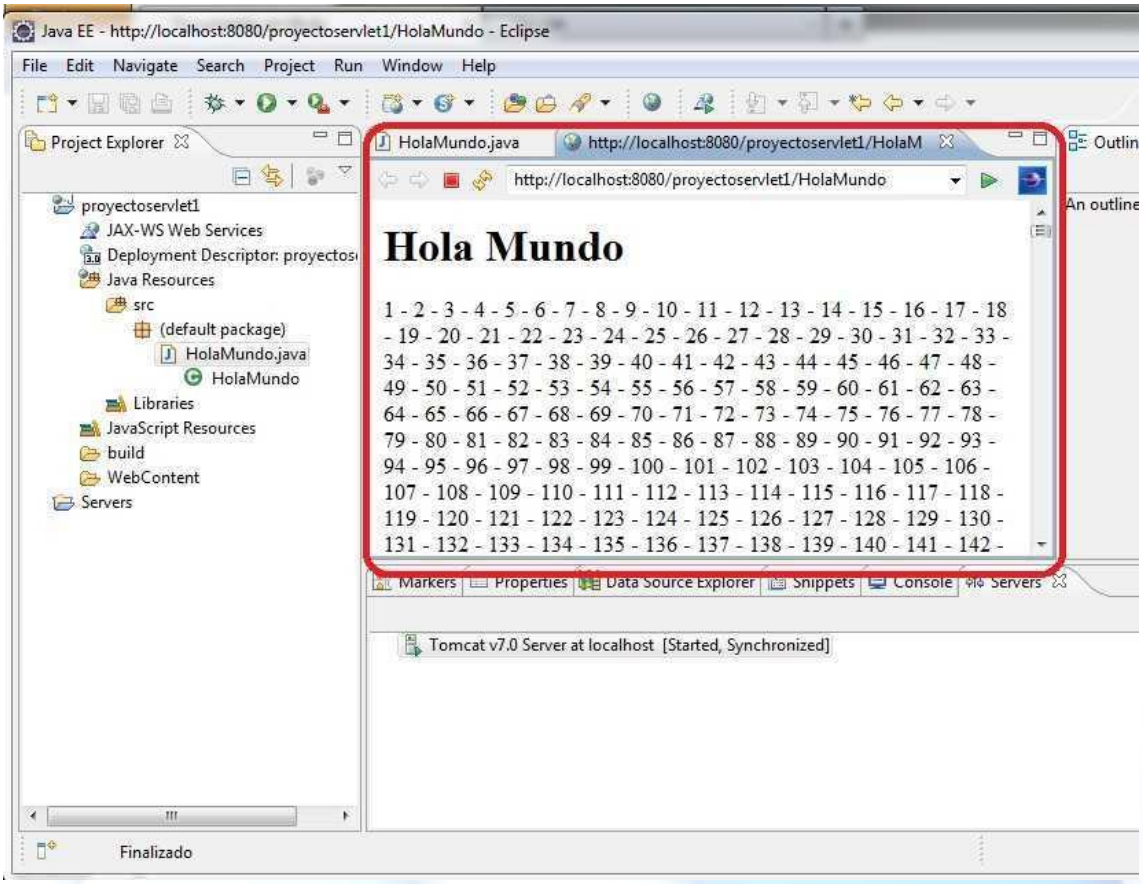
Para probar el servlet que acabamos de codificar debemos presionar el botón derecho del mouse sobre el nombre de la clase y seleccionar "Run on Server":



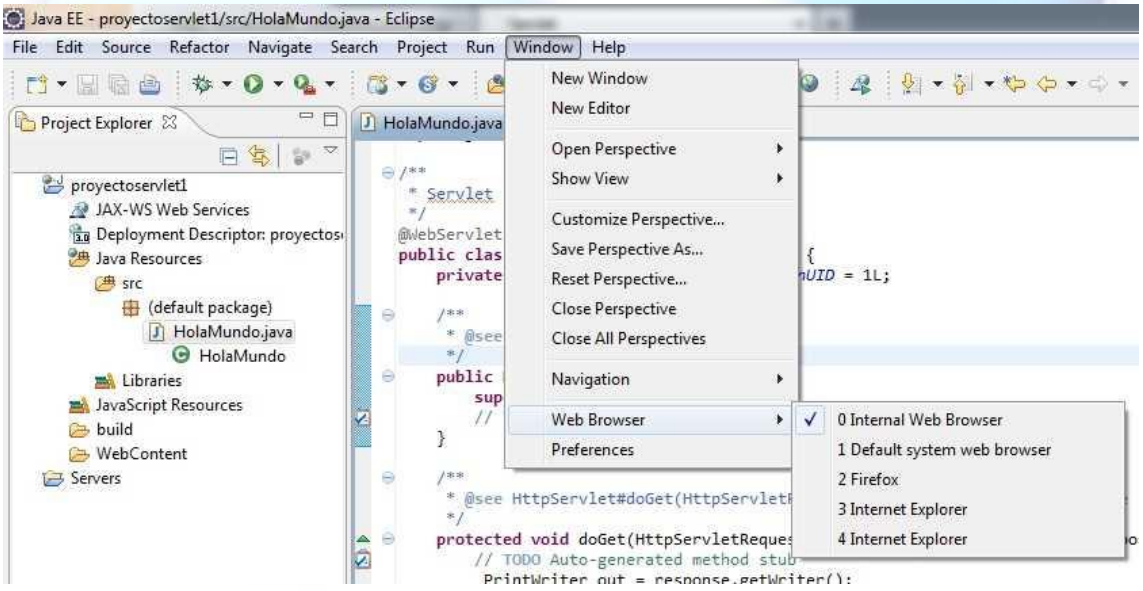
Aparece un diálogo que debemos seleccionar el botón "Finish" ya que está seleccionado el servidor "Tomcat" para ejecutar el servlet:



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:



Muchas gracias hasta la próxima clase.

Alsina 16 [B1642FNB] San Isidro | Pcia. De Buenos Aires |Argentina |

TEL.: [011] 4742-1532 o [011] 4742-1665 |

www.institutosanisidro.com.ar info@institutosanisidro.com.ar