Servidores Web

- Sesión 4: Herramientas para aplicaciones web

Puntos a tratar

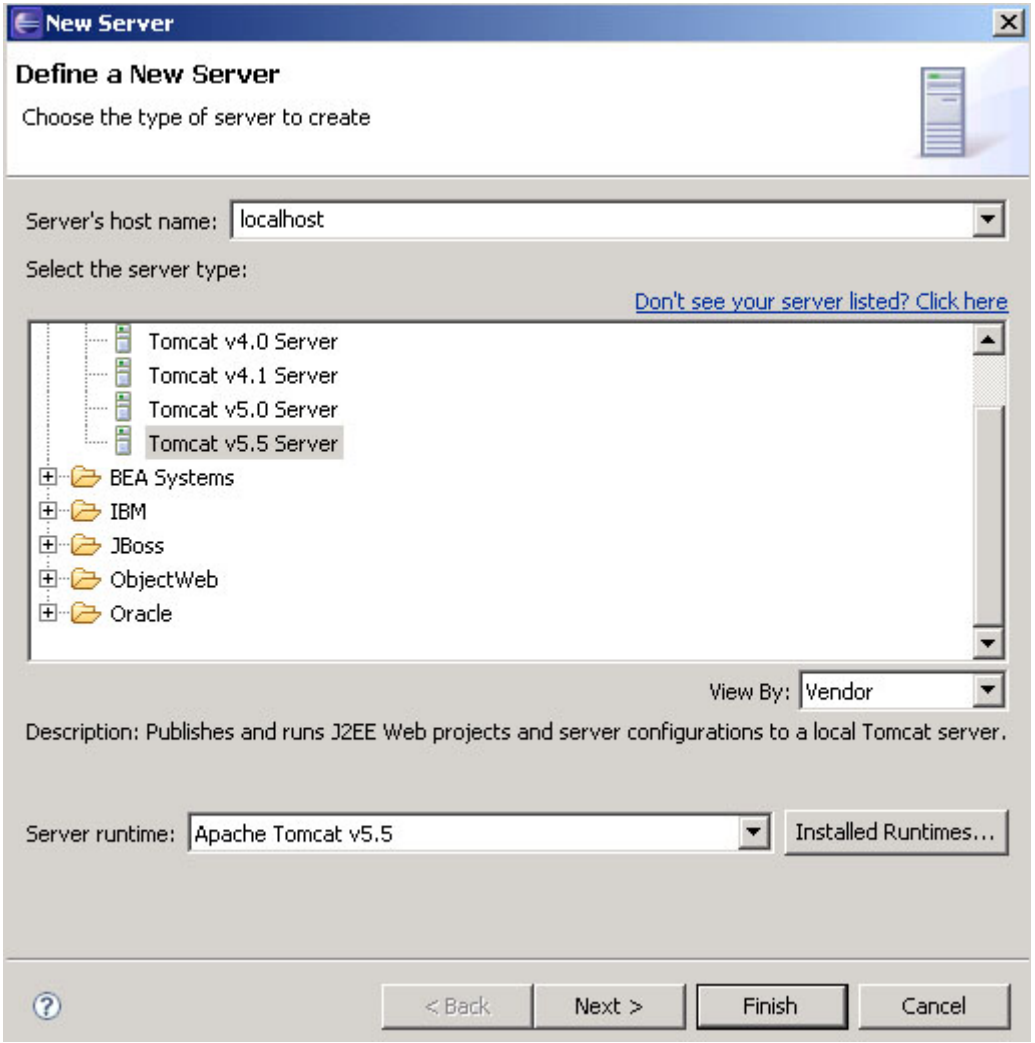
- Desarrollo y despliegue con WebTools
- Pooling de conexiones en Tomcat
- Introducción a las pruebas con Cactus
- Introducción al logging en aplicaciones web

WebTools

- Es un plugin de Eclipse que gestiona aplicaciones web como proyectos autointegrados
- Podremos:
  - Gestionar el servidor web en que desplegar
  - Crear y desarrollar la aplicación web
  - Desplegar y probar la aplicación en el servidor

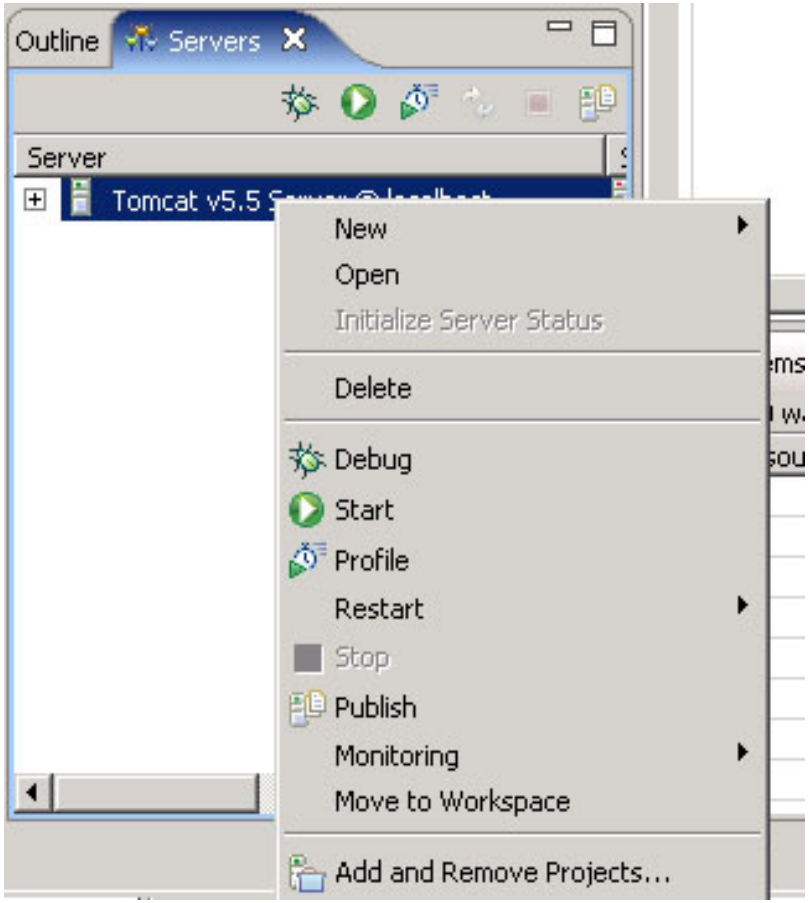
Añadir servidores web

- Vista *Servers*, *New - Server*



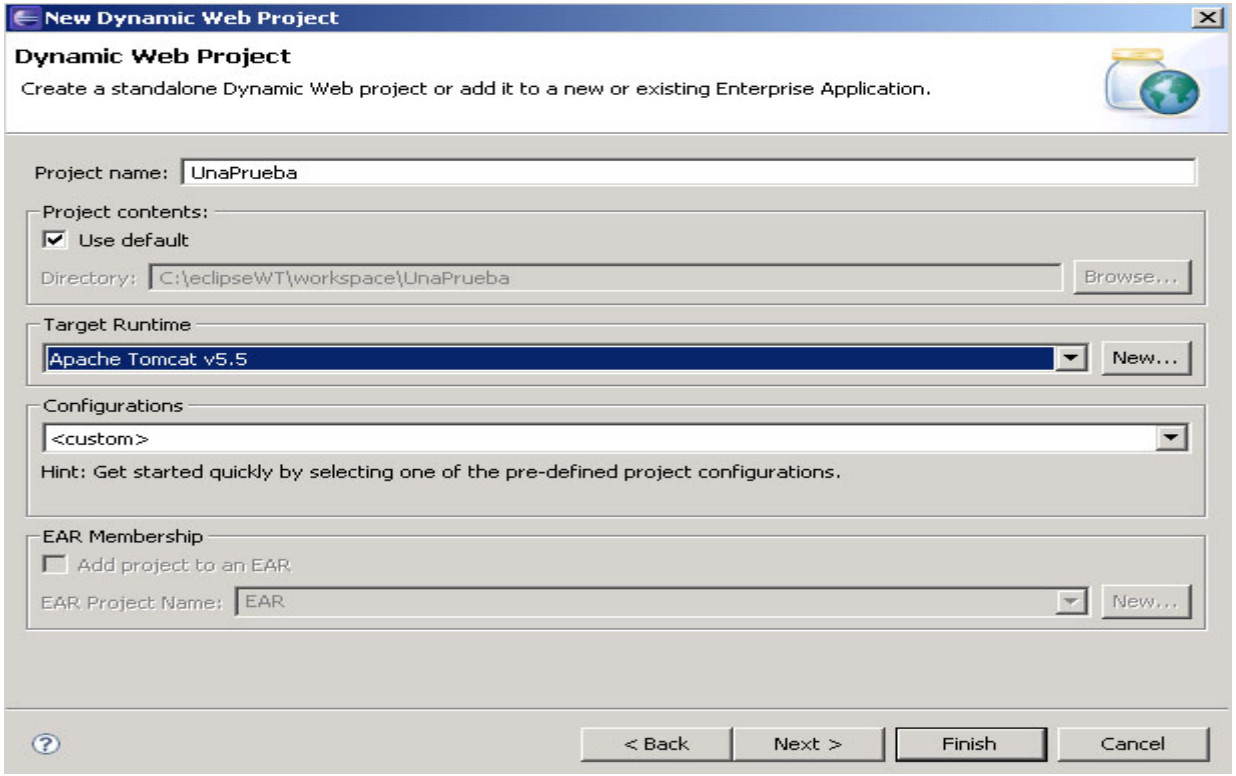
Gestionar los servidores web

- Botón derecho sobre el servidor en la vista *Servers*
- Tenemos opciones para pararlo, reanudarlo, etc



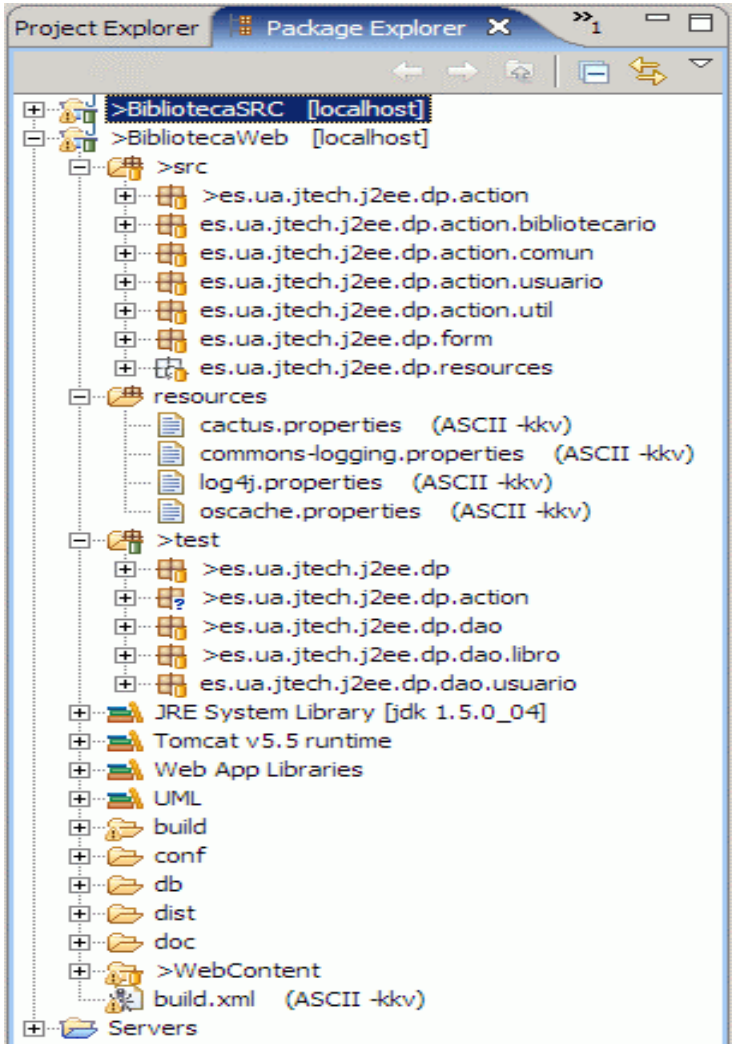
Crear proyecto de aplicación web

- Ir a *File – New – Other* y elegir *Web – Dynamic Web Project*



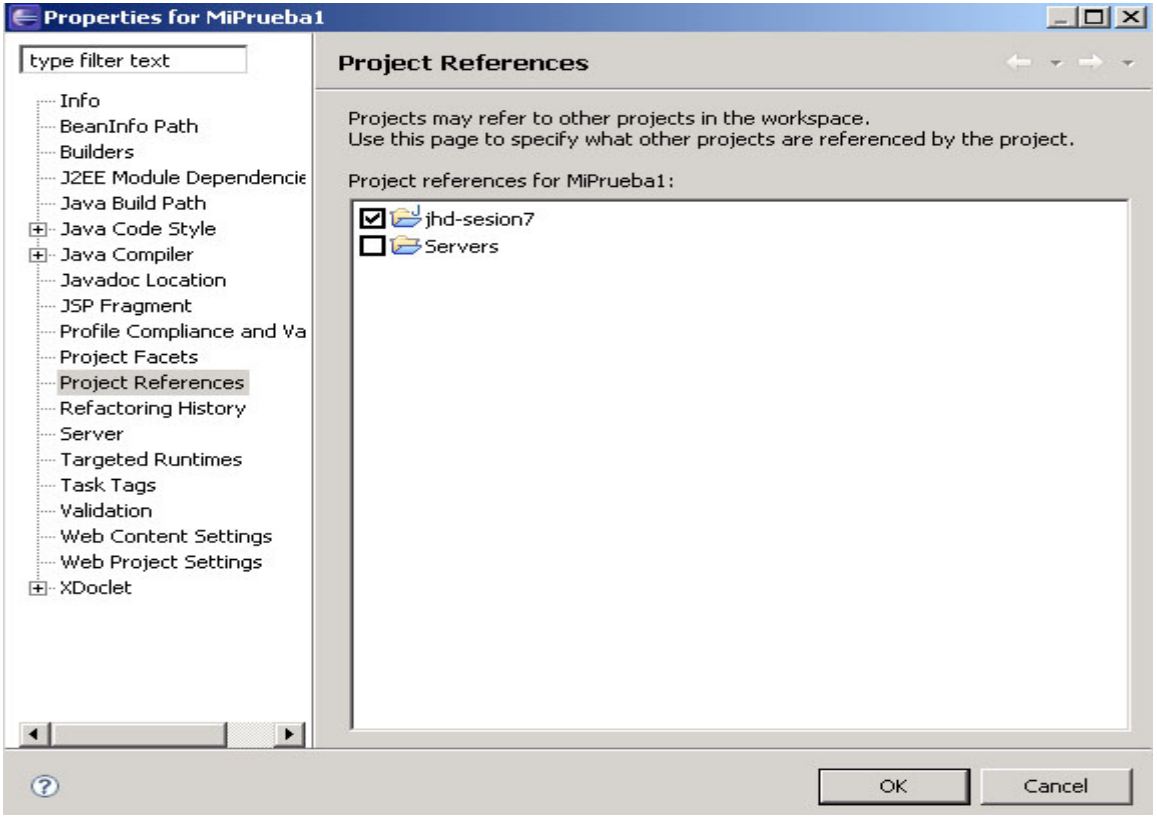
Crear proyecto de aplicación web (2)

- En los siguientes pasos del asistente, elegimos qué carpetas crear y la ruta del contexto que tendrá la aplicación
- Carpetas creadas por defecto:
  - **src**: fuentes
  - **WebContent**: esqueleto aplicación web (con WEB-INF y sus subcarpetas)
  - El resto de carpetas las crearemos nosotros



Interdependencias entre proyectos

- Podemos hacer que nuestro proyecto web dependa de otros proyectos previos



Despliegue de la aplicación

- La aplicación se desplegará sobre el servidor que tengamos asignado en la vista *Servers*.
- Pulsamos botón derecho sobre el proyecto web y elegimos *Run As – Run on Server*
  - En la siguiente pantalla podemos elegir sobre qué servidor de la vista de *Servers* ejecutarlo, si tuviésemos más de uno configurado.
- Repetiremos la operación tras cada cambio que queramos comprobar en la aplicación.

Pooling de conexiones en Tomcat

- Mediante ficheros de configuración podemos dejar definida una batería de conexiones a BD
- Así, cada petición de acceso a BD de las aplicaciones irá cogiendo conexiones libres, que quedarán ocupadas hasta que termine dicho acceso.
- De esta forma aseguramos poder atender hasta un máximo determinado de peticiones concurrentes

Configuración del pooling (1/2)

- Definir un fichero **context.xml** en la carpeta *WebContent/META-INF* de nuestro proyecto, indicando las características del pooling:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Context>
     <Resource name="miBD" type="javax.sql.DataSource“
       auth="Container" username="root" password="root“
       driverClassName="com.mysql.jdbc.Driver“
       url="jdbc:mysql://localhost:3306/bdprueba?autoReconnect=true"/>
     <ResourceParams name="miBD">
        <parameter><name>maxActive</name><value>20</value></parameter>
        <parameter><name>maxIdle</name><value>5</value></parameter>
        <parameter><name>maxWait</name><value>10000</value></parameter>
     </ResourceParams>
</Context>
```

Configuración del pooling (2/2)

- Añadimos un bloque **resource-ref** en nuestro **web.xml** referenciando al pooling creado en el paso anterior:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app ....>
      ...
      <resource-ref>
              <res-ref-name>miBD</res-ref-name>
              <res-type>javax.sql.DataSource</res-type>
              <res-auth>Container</res-auth>
      </resource-ref>
...
```

Cactus

- Es una librería de prueba de aplicaciones web de Jakarta, basada en JUnit.
- Su utilidad se basa en que es difícil probar el funcionamiento de todo lo que se envía y recibe en una petición/respuesta HTTP
- Se compone de:
  - *Framework*: el núcleo, que proporciona el API para implementar las pruebas
  - *Módulos de integración*: aplicaciones que facilitan el uso del framework (p.e., plugins en Eclipse)
  - *Ejemplos*: sobre cómo escribir y probar las pruebas

Instalación de Cactus: servidor

1. Copiar las librerías JAR en classpath (*WEB-INF/lib*)
2. Colocar clases originales y pruebas en sus lugares correspondientes en *WEB-INF/classes*
3. Añadir líneas en *web.xml* para mapear los redirectores usados por cactus
   - *Ver apuntes para ejemplo concreto*

Instalación de Cactus: cliente

1. Comprobar que las librerías JAR están en classpath (*WEB-INF/lib*)
2. Colocar en classpath (*WEB-INF/classes*) un fichero *cactus.properties* indicando URL de contexto de la aplicación:

```
cactus.contextURL=http://localhost:8080/BibliotecaWeb
```

- Este fichero lo copiaremos a nuestra carpeta *resources* del proyecto, y lo volcaremos automáticamente a *WEB-INF/classes*

Cactus y Ant

- Existe una tarea de Ant que despliega un fichero WAR con la aplicación en el servidor y ejecuta sobre él la clase de prueba que indiquemos

```
<taskdef resource="cactus.tasks">
        <classpath refid="project.classpath" /> </taskdef>
<target name="test" depends="dist">
        <cactus warfile="${dist.home}/${app.name}.war" fork="yes“
          printsummary="yes" showoutput="yes">
                <classpath refid="project.classpath" />
                <containerset timeout="180000">
                        <tomcat5x dir="${tomcat.home}“
                        serverxml="conf/server.xml" port="8080" />
                </containerset>
                <formatter type="plain" />
                <test name="es.ua.jtech.MiClaseTest" />
        </cactus>
</target>
...
```

Logging en aplicaciones web

- Utilizamos las mismas librerías **Log4J** y **commons-logging** vistas para Java
- Añadimos los ficheros JAR (*commons-logging-X.X.X.jar* y *log4j-X.X.X.jar*) en *WEB-INF/lib*
- Añadimos ficheros **commons-logging.properties** y **log4j.properties** en *WEB-INF/classes*
  - Realmente los añadiremos a la carpeta *resources*, para luego volcarlos automáticamente a *WEB-INF/classes*
- Sólo queda poner los mensajes de log en cada servlet o página JSP que queramos