Ejercicios de desarrollo, despliegue y pruebas de aplicaciones web con Maven

Índice

Instalación del extra de m2eclipse para WTP

Antes de realizar los ejercicios de esta sesión deberías instalar el extra para WTP del plugin m2eclipse, que estamos usando para poder importar proyectos Maven a este entorno. Consulta los apuntes para ver el proceso. Si no instalas este extra no podrás ejecutar el proyecto web desde Eclipse, solo desde Maven. Eso sí, aunque no tengas el extra instalado podrás compilarlo desde Eclipse sin problemas y al menos estar seguro de que no tiene errores de compilación

1. Creación de un proyecto web en Eclipse basado en Maven (1 pto)

Vamos a crear una pequeña aplicación web que va a convertir una cantidad de dinero entre pesetas y euros. La aplicación tendrá un HTML con un formulario para escribir los datos y un servlet para calcular y mostrar el resultado.

Debéis realizar los siguientes pasos:

1. **Crear el proyecto:** `File > New > Other...` y dentro de "Maven" escoger `Maven Project`.
   - En la pantalla de selección del arquetipo, escoger `webapp-jee5`. Este arquetipo ya tiene configurada la versión adecuada del API de servlets (2.5) y la del código generado por el compilador Java (1.5).
   - En la última pantalla del asistente, poner como groupId `es.ua.jtech` y como artifactId `conversor`
2. **Incluir el código fuente:**
   - Tomar el "index.html" incluido en las plantillas de la sesión y copiarlo a la carpeta "Web Resources" del proyecto. El "index.jsp" que contiene "Web Resources" debéis eliminarlo (si no lo hacéis aparecerá como página principal en lugar del index.html).
   - Crear un servlet llamado Conversor: `File > New > Servlet`. En la pantalla del asistente debes poner como package "es.ua.jtech" y como nombre de la clase "ConversorServlet". No darle al botón "Finish" sino a "Next" para que nos cree automáticamente el mapeo del servlet en el web.xml. En esta segunda pantalla del asistente puedes aceptar los valores por defecto (asociar con la URL "/ConversorServlet")
   - Sustituye el código fuente del servlet generado por el código incluido en el fichero "ConversorServlet.txt" de las plantillas de la sesión.
3. **Ejecutar el proyecto:** ejecuta el proyecto desde Eclipse como es habitual y comprueba que funciona correctamente. CUIDADO: esto no podrás hacerlo si no has instalado el extra del plugin m2eclipse.

2. Despliegue de la aplicación en Maven usando Cargo (1 pto)

1. Incluye en el "pom.xml" la configuración del plugin de Cargo como se explica en el

apartado "Despliegue de aplicaciones web con maven" de los apuntes.

2. Desde un terminal muévete al directorio del proyecto y ejecuta

```
mvn install cargo:start
```

para ejecutar la aplicación en Tomcat. La primera vez tardará un rato ya que tiene que bajarse Cargo y todas sus dependencias. Abre una ventana del navegador y comprueba que la aplicación funciona. Puedes parar el servidor pulsando Crtl-C en la terminal desde la que lo has arrancado o ejecutando mvn cargo:stop desde otra terminal y desde el directorio del proyecto

3. Comenta todo lo relativo a la "configuration" en el plugin anterior (es decir, solo debería quedar el "groupId" y el "artifactId"). No lo borres, solo ponlo entre comentarios (<!-- y -->). De este modo usarás la configuración por defecto de Cargo, que es el servidor Jetty. Arranca de nuevo la aplicación y comprueba que funciona en este servidor.

3. Pruebas funcionales con JWebUnit (1 pto)

1. Si has hecho el ejercicio 2, haz una copia del fichero pom.xml como "pom_ejer2.xml". Así quedará una copia de lo que hiciste en el ejercicio anterior, ya que vas a modificar de nuevo el pom.xml.
2. Incluye en el pom.xml la nueva configuración del plugin de Cargo. Esta configuración gestionará el arranque y parada automáticos del servidor en la fase de "integration-test" (cuidado, tienes que sustituir la configuración de Cargo que has puesto en el ejercicio 2, esta nueva incluye también la anterior)

```
<plugin>
        <groupId>org.codehaus.cargo</groupId>
        <artifactId>cargo-maven2-plugin</artifactId>
        <configuration>
                <wait>false</wait>
                <container>
                        <containerId>tomcat6x</containerId>
                        <home>/opt/apache-tomcat-6.0.29</home>
                </container>
                <configuration>
<home>${project.build.directory}/tomcat6x</home>
                </configuration>
        </configuration>
        <executions>
                <execution>
                        <id>start</id>
                        <phase>pre-integration-test</phase>
                        <goals>
                                <goal>start</goal>
                        </goals>
                </execution>
                <execution>
                        <id>stop</id>
                        <phase>post-integration-test</phase>
```

```
                        <goals>
                                <goal>stop</goal>
                        </goals>
                </execution>
        </executions>
</plugin>
```

3. Incluye la configuración del plugin "surefire" para que se usen los test del directorio "it" en la fase de integración:

```
<plugin>
 <groupId>org.apache.maven.plugins</groupId>
 <artifactId>maven-surefire-plugin</artifactId>
 <configuration>
  <!--
        Excluir los test de integración de la fase de test
        (fase de pruebas unitarias)
  -->
  <excludes>
     <exclude>**/it/**</exclude>
  </excludes>
 </configuration>
 <executions>
   <execution>
      <phase>integration-test</phase>
         <goals>
            <goal>test</goal>
         </goals>
         <configuration>
            <!--
                Incluir los test de integración en la fase
                integration-test
            -->
            <excludes>
               <exclude>none</exclude>
            </excludes>
            <includes>
               <include>**/it/**</include>
            </includes>
         </configuration>
   </execution>
 </executions>
</plugin>
```

4. Incluye la dependencia de JWebUnit en la sección de `<dependencies>`

```
<dependency>
        <groupId>net.sourceforge.jwebunit</groupId>
        <artifactId>jwebunit-htmlunit-plugin</artifactId>
        <version>2.2</version>
        <scope>test</scope>
</dependency>
```

5. Escribe la clase de prueba:
   - Crea una nueva clase llamada `ConversorTest` que herede de `WebTestCase` y

resida en el package **es.ua.jtech.conversor.it** pero asegúrate de que se mete en la carpeta `src/test/java`, y no `src/main/java`, ya que es un test.

- Fijándote en el ejemplo de test JWebUnit de los apuntes, escribe el método `setUp` estableciendo como URL base "http://localhost:8080/conversor"
- Escribe un método de prueba llamado `testConversor` que introduzca un valor en euros en el campo "cantidad", envíe el formulario y compruebe que en la página resultante aparece el resultado correcto.