P1- Maven y Netbeans

Objetivos

- Vamos a ver cómo usar Maven con el IDE Netbeans. Continuaremos trabajando conceptos tales como dependencias y configuración de *plugins*, así como con otros tipos de proyectos Maven

Herramientas

- Netbeans 7.2 y Maven 3.0.4
- Se proporciona el fichero PlantillasP1.zip que contiene ficheros que vamos a utilizar para el desarrollo de los ejercicios de la práctica

---

Netbeans proporciona soporte para Maven, de forma que podemos utilizarlo desde este IDE. Netbeans tiene su propio Maven. Más adelante veremos que es una muy buena práctica independizar lo máximo posible nuestro desarrollo de un IDE concreto. Para ello tendremos que configurar Netbeans para que use un Maven externo, concretamente el que hemos utilizado desde línea de comandos, así como el repositorio local que ya hemos utilizado en la práctica anterior.

Para trabajar con proyectos Netbeans (crear, editar, borrar,...), utilizamos la Ventana **Projects** (si por cualquier motivo se ha cerrrado dicha ventana, podemos hacerla visible de nuevo desde la opción del Menú: Windows→Projects). Esta ventana nos muestra una vista de proyectos Netbeans. Otra vista que nos resultará útil es la proporcionada por la ventana **Files** (para mostrar esta vista ir a Windows→Files). La vista **Files** nos muestra exactamente todos los ficheros de nuestro disco duro que forman cada uno de nuestros proyectos Netbeans. Finalmente, una tercera vista que vamos a utilizar es la vista **Services**. Aquí podemos ver algunos de los servicios que nos ofrece Netbeans, como por ejemplo servidores de aplicaciones y repositorios Maven, entre otros.

Ejercicios

1. **Ejercicio 1**. Para **CONFIGURAR** Netbeans para utilizar un Maven externo haremos lo siguiente:
   A) Desde el menú `Preferencias→Java→Maven`, cambiamos el valor de `Maven Home` a:

      /usr/local/apache-maven/apache-maven-3.0.4

   B) Desde la pestaña `Services→Maven Repositories→Local`, con el botón derecho del ratón elegimos `Properties`, y nos aseguramos que el campo "`Local repository path`" tiene el valor $HOME/.m2/repository

   C) Una vez configurado, vamos a probar Maven abriendo el proyecto multi-módulo que hemos creado en la sesión anterior. Para ello, desde la vista Projects, pulsamos el botón derecho del ratón y elegimos "Open Project". A continuación buscaremos el proyecto miMultiModulo en nuestro disco duro, lo seleccionamos y observamos que Netbeans ha "reconocido" el tipo de proyecto multimódulo, y en la parte derecha de la ventana nos aparece la lista de proyectos que forman parte de nuestro proyecto maven multi-módulo. Vamos a marcar la casilla "Open Required Projects", así Netbeans abrirá tanto el proyecto multi-módulo como los proyectos a los que incluye. En el elemento "Project Files" de cada proyecto podemos ver el fichero "pom.xml" (en la vista Projects, los iconos con forma de carpeta no representan directorios "físicos", sino la estructura que Netbeans muestra de los proyectos. Podemos cambiar a la vista Files y veremos que en

esta vista el fichero pom.xml está justamente en el directorio de nuestro proyecto Maven, es decir, en la vista Files, los iconos con forma de carpetas sí representan directorios físicos tal cual están en nuestro disco duro).

2. **Ejercicio 2**. Ahora vamos a ver cómo podemos **EJECUTAR COMANDOS** sobre un proyecto Maven. Ilustraremos el proceso utilizando, por ejemplo, el proyecto Java Prueba. Seleccionamos el proyecto java (los proyectos java Maven se identifican con un icono con forma de taza con una M). Pulsamos el botón derecho para mostrar el menú contextual, en el que veremos las opciones Run, Debug, Profile y Test (entre otras). Cada una de estas opciones se llaman "**Actions**" y representan diferentes comandos Maven, que podemos configurar. Por ejemplo, vamos a ver cómo está configurada la `Action Run`. Para ello elegimos la opción `Properties` del menú contextual de nuestro proyecto maven. En la ventana asociada, marcamos el elemento `Actions`, y veremos, a la derecha de la ventana, una lista con las Actions disponibles. Pinchamos sobre la Action "`Run Project`", y vemos que en el campo "Excute goals" aparece: "`process-classes org.codehaus.mojo:exec-maven-plugin:1.2:exec`". Esto significa que Netbeans ejecuta el siguiente comando de Maven: "`mvn process-classes org.codehaus.mojo:exec-maven-plugin:1.2:exec`". Además se pueden incluir valores concretos para algunas propiedades de dichas goals, en el campo "Set Properties". Se pide:
   A) Ejecuta la opción **Run** sobre el proyecto maven prueba
   B) Añade una nueva Action en el menú contextual del proyecto prueba, utilizando la opción "`Add Custom...`" (desde `Properties→Actions`). La nueva *action* se llamará **Instalar** y tendrá asociado el comando `mvn install`. Para ejecutar dicha *action* podremos hacerlo desde al menú contextual del proyecto, y eligiendo la opción `custom→Instalar`. Indica si hay alguna diferencia entre ejecutar desde línea de comandos `mvn install` desde el directorio prueba, o pulsar la opción **Instalar** desde el menú contextual de Netbeans.
3. **Ejercicio 3**. En este ejercicio vamos a trabajar con un **PROYECTO WEB**, al que denominaremos Hotel. Se trata de un proyecto multi-módulo formado por una aplicación Web (empaquetada como HotelWebApp.war) y una aplicación java (empaquetada como HotelDatabase.jar). La siguiente figura muestra de forma gráfica la relación entre ambas aplicaciones.

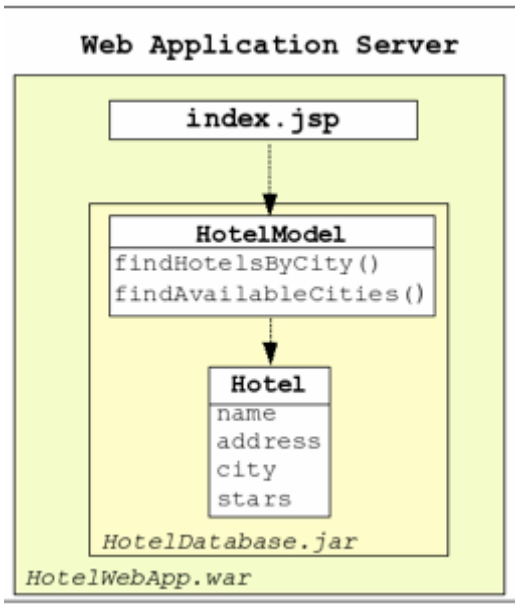


Figura 1. Aplicación HotelWebApp

Para ello:

A) Comenzaremos por crear la aplicación multi-módulo con NetBeans (botón derecho, y elegimos New Project→Maven→POM Project), con el nombre **Hotel**, y asignamos el valor **ppss** a la propiedad `GroupId`. Dejamos el valor por defecto para la propiedad `Versión`, y asignamos el valor `ppss` a la propiedad `package`.

B) Ahora creamos el proyecto java con nombre **HotelDatabase** (elige un proyecto Maven de tipo "Java application"). Dado que este proyecto será un módulo del proyecto Hotel, vamos a crearlo dentro del directorio Hotel. Si lo hacemos así, Netbeans añade automáticamente el proyecto HotelDatabase como un módulo del proyecto Hotel (compruébalo). ¿Qué ocurre si NO creamos el proyecto HotelDatabase como un subdirectorio de Hotel? ¿Qué tendríamos que hacer entonces para que el proyecto HotelDatabase sea un módulo del proyecto Hotel? En el directorio **PlantillasP1/hotel** encontrarás los ficheros `HotelModel.java` y `Hotel.java` que contienen la implementación de las clases `ppss.model.HotelModel` y `ppss.businessobjects.Hotel`, respectivamente, y que constituirán los fuentes de tu proyecto (para ello puedes previamente crear los paquetes (New→Java Pakages, desde "Source Packages") y las clases con Netbeans (New→Java Class, desde el paquete correspondiente), y luego copiar y pegar el código de la plantilla en las clases creadas. La clase por defecto App tendrás que borrarla porque no será necesaria). Explica el código de ambas clases. Finalmente ejecuta la opción "`Build`" desde el menú contextual del proyecto HotelDatabase. ¿Cómo está configurada la Action asociada y cuál es el resultado de ejecutar dicha opción?

C) A continuación creamos el proyecto web con nombre **HotelWebApp** (elige un proyecto Maven de tipo "Web Application", asegúrate de que el campo "Server" tenga el valor "Glassfish Server 3.1.2", y "Java EE Version" tenga el valor "Java EE 6 Web"). Comprueba que el proyecto creado se ha añadido como un segundo módulo en la aplicación Hotel.

D) Según hemos visto en la Figura 1, la aplicación Web, depende de la aplicación HotelDatabase. Por ello, el siguiente paso es modificar nuestro pom.xml para añadir la siguiente dependencia con la aplicación HotelDatabase. Para hacer esto nos situamos en el menú contextual del elemento "Dependencies" del proyecto HotelWebApp, y seleccionamos "Add Dependency..."→Open Projects→HotelDatabase. Si editamos el fichero pom.xml, veremos que se ha añadido la dependencia siguiente:

```
<dependency>
  <groupId>${project.groupId}</groupId>
  <artifactId>HotelDatabase</artifactId>
  <version>${project.version}</version>
</dependency>
```

¿Cuál es el significado de estas líneas en el pom.xml? ¿Cuál es el valor de "scope" para esta dependencia? Netbeans nos permite ver de forma gráfica las dependencias de nuestro proyecto con la opción "Open POM" del menú contextual del proyecto. Una vez que hemos ejecutado la opción, se abrirá el fichero pom.xml en la vista de edición, y podemos ver las pestañas "Source" (que es la que está activada por defecto), "Graph" e "History". Si elegimos la vista "Graph" veremos de forma gráfica las siguientes dependencias (el proyecto del cual se muestran las dependencias aparecerá en el grafo en negrita). Pruébalo para cada uno de los tres proyectos. (**Aclaración**: si el grafo de dependencias no se muestra correctamente, vuelve a intentarlo después de "desplegar" el proyecto, cosa que haremos un poco más adelante). Dadas las dependencias de nuestra aplicación, explica por qué hemos ejecutado la action *Build* para el proyecto `HotelDababase`?

E) Ahora vamos a implementar nuestra aplicación Web. Para ello se os proporciona el fichero **PlantillasP1/hotel/index.jsp**, que contiene la codificación de **index.jsp** (fichero `src/main/webapp/index.jsp` si utilizamos la vista *Files*, o fichero `Web Pages/index.jsp` si utilizamos la vista *Projects*). Finalmente vamos a "instalar" nuestra aplicación multi-módulo ejecutando "Build" desde el menú contextual de "Hotel". Con esto habremos generado en nuestro repositorio local todos los artefactos (ficheros jar y war) listos para ser desplegados en un servidor de aplicaciones.

F) El último paso el "**DESPLIEGUE**" y ejecución de nuestra aplicación web. Para ello iremos a la vista "Services" y pondremos en marcha el servidor de aplicaciones Glassfish (pulsando "`Start`" en el menú contextual de "`Servers→Glassfish Server 3.1.2`". Aparecerá un triángulo verde indicando que el servidor está en marcha). Seguidamente volvemos a la vista "Projects" y para desplegar nuestra aplicación web pinchamos en la opción "**Run**" del menú contextual de "HotelWebApp". Si todo ha ido bien se ejecutará la aplicación en una nueva ventana del navegador (http://localhost:8080/HotelWebApp). Podemos "ver" nuestra aplicación como "Servers→Glassfish Server 3.1.2→Applications→ppss_HotelWebApp_war_1.0-SNAPSHOT" en la vista "Services".

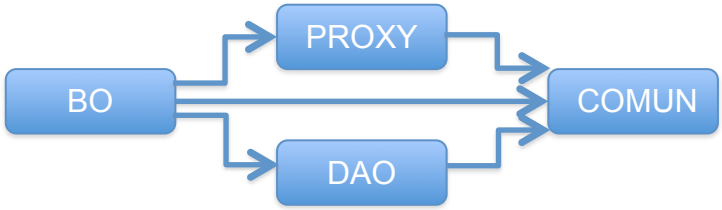
4. **Ejercicio 4**. En este ejercicio vamos a crear un proyecto multi-módulo denominado **Matriculacion**. Asignamos el valor **ppss.loginEPS** a la propiedad `GroupID`, siendo **loginEPS** el login de tu dirección de correo institucional, por ejemplo si tu correo institucional es abc@alu.ua.es, entonces como valor de `GroupID` pondremos `ppss.abc`. Dejamos el valor por defecto para la propiedad `Versión`, y asignamos el valor `ppss` a la propiedad `package` Los nombres de los módulos (proyectos Maven Java) que contiene, así como los nombres de los paquetes de cada uno se muestran en la siguiente tabla (recuerda que para cada uno de ellos el valor de la propiedad **GroupID**, que figurará en el pom.xml de cada módulo, debe ser **ppss.loginEPS**):

Nombre del proyecto (artifactId)	Paquete (package)
`Matriculacion-comun`	`ppss.matriculacion.to`
`Matriculacion-dao`	`ppss.matriculacion.dao`
`Matriculacion-proxy`	`ppss.matriculacion.proxy`
`Matriculacion-bo`	`ppss.matriculacion.bo`

El proyecto **Matriculacion** gestiona el proceso de matriculación de alumnos en diferentes asignaturas, encargándose de calcular el importe de los recibos correspondientes en función de las asignaturas de las que se matricula dicho alumno.

Los ficheros de fuentes para cada uno de los paquetes del proyecto los encontrarás en **PlantillasP1/matriculacion/to**, **PlantillasP1/matriculacion/bo**, **PlantillasP1/matriculacion/proxy**, y **PlantillasP1/matriculacion/dao**.

A) A continuación añadimos las dependencias entre los módulos, que mostramos en la siguiente figura (la flecha X → Y, se lee "X depende de Y"):



Comprueba de forma gráfica que tienes bien las dependencias creadas en Netbeans.

B) Instala el proyecto Matriculacion en el repositorio local (opción "Build").

Nota: Como ya sabemos, las clases y artefactos (.jar, .war, etc...) se generan en el directorio "`target`" de cada proyecto. Puedes ver el contenido del directorio target de cada proyecto en la vista Files, expandiendo la carpeta denominada target. Aquí verás que los ficheros .class del directorio src están en la carpeta classes. Un poco más adelante, cuando utilicemos ficheros de recursos, veremos que en la carpeta target también se habrán copiado los ficheros de recursos correspondientes. Asímismo, en la carpeta `test-classes` se encuentran los ficheros *.class* del directorio `test` (y, como veremos más adelante, los ficheros de recursos asociados). Dentro del directorio `target` verás también el artefacto generado (fichero .jar, .war, ...). Como ya sabrás, estas extensiones representan ficheros comprimidos. Puedes ver exactamente cuáles son los contenidos de dichos ficheros pulsando sobre el triángulo situado a la izquierda del nombre del fichero.

C) El módulo **Matriculacion-dao** es el que gestiona el acceso a la base de datos. Vamos a configurar dicho acceso. Concretamente vamos a trabajar con una base de datos en memoria, denominada **HSQLDB**. Lo primero que tenemos que hacer es incluir el *driver* del sistema gestor de base de datos HSQLDB como una dependencia con la librería correspondiente. Concretamente, se trata de la siguiente dependencia:

```
<dependency>
   <groupId>hsqldb</groupId>
   <artifactId>hsqldb</artifactId>
   <version>1.8.0.10</version>
 </dependency>
```

Añade la dependencia con Netbeans (fíjate que en este caso tendremos que introducir los valores de `groupId`, `artifactId` y `version`, en los campos correspondientes). Edita el fichero pom.xml para comprobar que se ha añadido correctamente la dependencia.

D) Ahora vamos a introducir un par de *scripts* SQL para crear la base de datos e inicializarla con una serie de datos de prueba (son los ficheros **PlantillasP1 / matriculacion/matriculacion.sql** y **PlantillasP1 /matriculacion/datos.sql** respectivamente). El primero de ellos lo introduciremos en la carpeta `src/main/resources/sql`, y el segundo en `src/test/resources/sql` (tendrás que crear las carpetas correspondientes, New→Other→Folder). ¿Por qué crees que se ha separado de esta forma?

E) Para ejecutar los *scripts* anteriores necesitamos utilizar el plugin *sql-maven-plugin* (http://mojo.codehaus.org/sql-maven-plugin/). A continuación incluimos el código con la configuración del plugin. Dicho código debe ir dentro de las etiquetas

`<build><plugins></build></plugins>` en el fichero *pom.xml* del proyecto *matriculacion-dao*.

```
<plugin>
    <groupId>org.codehaus.mojo</groupId>
    <artifactId>sql-maven-plugin</artifactId>
    <version>1.4</version>
    <!-- Dependencia con el driver JDBC -->
    <dependencies>
       <dependency>
          <groupId>hsqldb</groupId>
          <artifactId>hsqldb</artifactId>
          <version>1.8.0.10</version>
        </dependency>
    </dependencies>
    <!-- Configuración común a todas las ejecuciones -->
    <configuration>
       <driver>org.hsqldb.jdbcDriver</driver>
       <url>jdbc:hsqldb:file:matriculacionbd;shutdown=true</url>
       <username>SA</username>
       <password></password>
    </configuration>

    <executions>
      <execution>
        <id>create-db</id>
          <phase>process-test-resources</phase>
          <goals>
              <goal>execute</goal>
          </goals>
          <configuration>
            <srcFiles>
              <srcFile>src/main/resources/sql/matriculacion.sql</srcFile>
            </srcFiles>
          </configuration>
        </execution>

        <execution>
          <id>create-data</id>
          <phase>process-test-resources</phase>
          <goals>
            <goal>execute</goal>
          </goals>
          <configuration>
            <srcFiles>
              <srcFile>src/test/resources/sql/datos.sql</srcFile>
            </srcFiles>
          </configuration>
        </execution>
     </executions>
  </plugin>
```

¿Qué acciones realiza este *plugin*? ¿En que momento las realiza? Vuelve a hacer una instalación de este proyecto con Maven y comprueba en los mensajes de pantalla que se ejecutan los dos *scripts* sql

F) Crea dentro del paquete `ppss.matriculacion.dao` en la carpeta `src/test/java` una clase **ConsultaBD** que contenga un método *main* con el siguiente código:

```
public static void main(String[] args) throws DAOException {
  FactoriaDAO factoriaDao = new FactoriaDAO();
  IAlumnoDAO alumnoDao = factoriaDao.getAlumnoDAO();
  List<AlumnoTO> alumnos = alumnoDao.getAlumnos();
  System.out.println("Numero de alumnos: " + alumnos.size());

  for(AlumnoTO alumno: alumnos) {
     System.out.println(alumno.getNombre());
  }
}
```

G) Antes de continuar, vamos a comprobar qué clases de test han sido generadas para el proyecto **Matriculacion-dao** (contenido del directorio `target/test-classes` de dicho proyecto). Indica cuál es la ruta de directorios para los ficheros .sql que hemos incluido en el proyecto como ficheros de recursos. Ahora vamos a volver a generar el .jar del proyecto y copiarlo en el repositorio nuestro repositorio local. Y a continuación volvemos a comprobar qué clases de **test** han sido generadas para dicho proyecto.

H) Finalmente, vamos a ejecutar nuestra clase **ConsultaBD**. Para ello, vamos a generar un nueva *Action*. Esta vez vamos a hacerlo desde el menú contextual de **Matriculacion-dao**, con la opción "`Custom→Goals...`". En el campo *Goals*, pondremos el valor: `exec:java`. En el campo *Properties* añadimos las propiedades: `exec.mainClass=ppss.matriculacion.dao.ConsultaBD` y `exec.classpathScope=test`. Marcamos la casilla *"Remember as:"*, y le damos el valor `Run ConsultaBD`. Con esto habremos creado una nueva action. Para ejecutarla, lo hacemos desde el menú contextual del proyecto *Matriculacion-dao*, opción "`Custom→Run ConsultaDB`". Explica cuál es el resultado de la ejecución. ¿Qué ocurre si no ponemos la propiedad `exec.classpathScope`?

Nota: Seguiremos trabajando con el proyecto Matriculacion en las próximas sesiones. Por lo tanto, asegúrate de NO borrar el trabajo realizado. Si has creado los módulos de Matriculación como subdirectorios de éste, simplemente tendrás que asegurarte de no borrar la carpeta Matriculación. Para seguir trabajando en casa, tendrás que copiar la carpeta Matriculación y todo lo que contiene. Es conveniente "limpiar " los proyectos (eliminar todos los artefactos generados), para que ocupen menos espacio. Para ello utiliza la opción "Clean" del menú contextual del proyecto Matriculacion.