P07- Proyectos Maven multimódulo

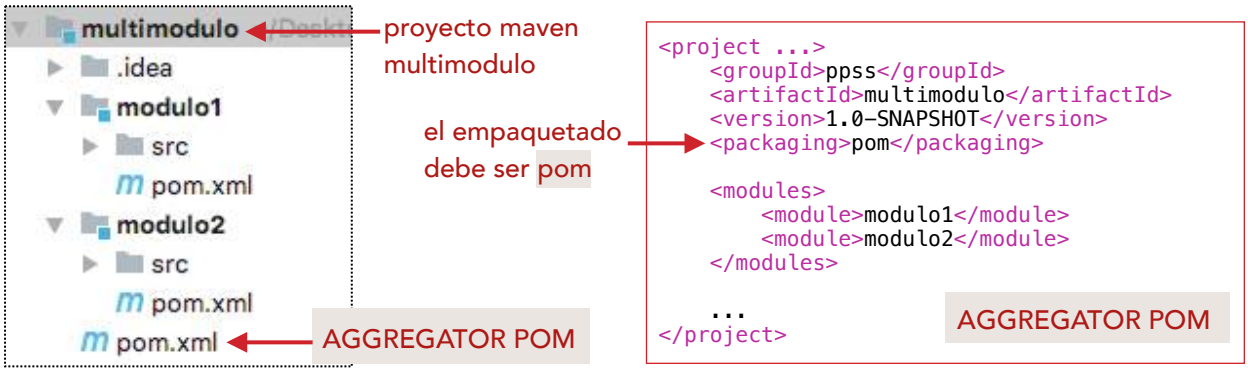
Proyectos Maven multimódulo

Dependiendo de lo "grande" que sea nuestro proyecto software, es habitual "distribuir" el código en varios "subproyectos". Hasta ahora hemos trabajado con proyectos maven independientes, de forma que cada uno contenía "todo" nuestro código.

Maven permite trabajar con proyectos que a su vez están formados por otros proyectos maven, a los que llamaremos "**módulos**", de forma que podemos establecer diferentes "**agrupaciones**" entre ellos, tantas como sean necesarias. Un proyecto Maven que "agrupa" a otros proyectos se denomina proyecto maven multimódulo

Un proyecto maven multimódulo, tiene un empaquetado (etiqueta <packaging>) con el valor "pom", y contiene una lista de módulos "agregados" (que son otros proyectos Maven). Por eso a este pom también se le denomina "***aggregator pom***".

A continuación mostramos un ejemplo de un proyecto maven multimódulo que "contiene" dos módulos. Dichos módulos pueden estar físicamente en cualquier ruta de directorios, pero lo más práctico es que los módulos se encuentren físicamente en el directorio del *pom aggregator*



Observa que el proyecto multimódulo sólo contiene el pom.xml, NO tiene código (carpeta src). En el directorio del proyecto añadiremos tantos módulos como queramos agrupar, y lo indicaremos en el pom anidando etiquetas <module>.

Los módulos agregados de nuestro proyecto multimódulo son proyectos maven "normales", y pueden construirse de forma separada o a través del *aggregator pom*.

Un **proyecto multimódulo** se **construye** a partir de su *aggregator* pom, que gestiona al grupo de módulos agregados. Cuando construimos el proyecto a través del *aggregator pom*, cada uno de los proyectos maven agrupados se construyen si tienen un empaquetado distinto de *pom*.

El mecanismo usado por maven para gestionar los proyectos multimódulo recibe el nombre de ***reactor***. Se encarga de "recopilar" todos los módulos, los ordena para construirlos en el oren correcto, y finalmente realiza la construcción de todos los módulos en ese orden. El orden en el que se construyen los módulos es importante, ya que éstos pueden tener dependencias entre ellos.

A continuación mostramos un ejemplo de construcción de nuestro proyecto multimódulo. En este caso vamos a ejecutar el comando: *mvn compile*.

Podemos observar que se ha establecido un orden entre los tres módulos: multimodulo, modulo1 y modulo2. Y a continuación se ejecuta el comando para todos y cada uno de los módulos en el orden establecido por *reactor*. Cuando en el empaquetado es pom, por defecto solamente tiene asociadas las goals *install:install*, y *deploy:deploy* a las fases *install* y *deploy*, respectivamente.

```
> cd multimodulo
> mvn compile
[INFO] Scanning for projects...
[INFO] Reactor Build Order:
[INFO] multimodulo                                        [pom]
[INFO] modulo1                                            [jar]
[INFO] modulo2                                            [jar]
[INFO]
[INFO] -------------------------< ppss:multimodulo >-------------------------
[INFO] Building multimodulo 1.0-SNAPSHOT                        [1/3]
[INFO] --------------------------------[ pom ]---------------------------------
[INFO]
[INFO] ----------------------------< ppss:modulo1 >----------------------------
[INFO] Building modulo1 1.0-SNAPSHOT                            [2/3]
[INFO] --------------------------------[ jar ]---------------------------------
            //ejecuciones de las goals correspondientes...
[INFO] ----------------------------< ppss:modulo2 >----------------------------
[INFO] Building modulo2 1.0-SNAPSHOT                            [3/3]
[INFO] --------------------------------[ jar ]---------------------------------
            //ejecuciones de las goals correspondientes...
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Summary for multimodulo 1.0-SNAPSHOT:
[INFO]
[INFO] multimodulo ........................................ SUCCESS [  0.030 s]
[INFO] modulo1 ............................................ SUCCESS [  1.194 s]
[INFO] modulo2 ............................................ SUCCESS [  0.913 s]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
```

Beneficios de usar multimódulos

La ventaja significativa del uso de multimódulos es la **reducción de duplicación**. Por ejemplo, podremos construir todos los módulos con **un único comando** (aplicado al aggregator pom), sin preocuparnos por el orden de construcción, ya que maven tendrá en cuenta las dependencias entre ellos y los ordenará convenientemente.

También podremos "compartir" elementos como propiedades, dependencias, plugins, entre los módulos agregados usando el mecanismo de **herencia** que nos proporciona maven.

Maven soporta la relación de herencia, de forma que podemos crear un pom que nos sirva para identificar a un proyecto "padre". Podemos incluir cualquier configuración en el pom del proyecto padre , de forma que sus módulos "hijo" la hereden, evitando de nuevo duplicaciones.

Siguiendo con nuestro ejemplo, el proyecto multimodulo será nuestro proyecto "padre", y sus hijos serán los módulos *modulo1* y *modulo2*. Para ello, tendremos que referenciar al proyecto "padre" en cada uno de los módulos "hijo" usando la etiqueta <parent>.

```xml
<project ...>

  <parent>
    <artifactId>multimodulo</artifactId>
    <groupId>ppss</groupId>
    <version>1.0-SNAPSHOT</version>
  </parent>

  <artifactId>modulo1</artifactId>

  ...
</project>
```

modulo1/pom.xml

```xml
<project ...>

  <parent>
    <artifactId>multimodulo</artifactId>
    <groupId>ppss</groupId>
    <version>1.0-SNAPSHOT</version>
  </parent>

  <artifactId>modulo2</artifactId>

  ...
</project>
```

modulo2/pom.xml

En este caso el proyecto maven *multimodulo* es el proyecto "padre. NO tenemos que indicar en el pom "padre" quienes son sus "hijos", sino que en cada módulo "hijo" añadiremos las coordenadas del proyecto "padre", del cual heredarán su configuración. Prácticamente se heredan la mayoría de elementos: <properties>, <dependencies>, <plugins>, <artifactId>, <version>,... Por eso no es necesario añadir las coordenadas <groupId> y <version> en los "hijos" si vana a ser las mismas que las del proyecto "padre". Entre las (pocas) etiquetas que NO se heredan están <artifactId> y <name>.

Podemos "sobreescribir" cualquier elemento heredado o usar el del proyecto "padre".

No es necesario usar los mecanismos de herencia y agregación conjuntamente. Es decir, podemos tener únicamente una relación de herencia entre nuestros proyectos maven, o sólo relaciones de agregación, o ambas, como hemos mostrado en nuestro ejemplo.

Obviamente, la reducción de duplicaciones será mayor si usamos a la vez herencia y agregación en nuestros proyectos maven.

Bitbucket

El trabajo de esta sesión también debes subirlo a *Bitbucket*. Todo el trabajo de esta práctica deberá estar en el directorio **P07-Multimodulo** dentro de tu espacio de trabajo, es decir, dentro de tu carpeta: ppss-Gx-apellido1-apellido2-2019.

Recuerda que si trabajas desde los ordenadores del laboratorio primero deberás configurar git y clonar tu repositorio de Bitbucket.

Ejercicios

En esta sesión trabajaremos con un proyecto IntelliJ **vacío**, e iremos añadiendo los módulos (proyectos Maven), en cada uno de los ejercicios.

Para **crear el proyecto IntelliJ**, simplemente tendremos que realizar lo siguiente:

- ***File→New Project***. A continuación elegimos "Empty Project" y pulsamos sobre Next,
- ***Project name*** : "P07-multimodulos". ***Project Location:*** "$HOME/ppss-Gx-.../P07-Multimodulo/P07-multimodulos". Es decir, que tenemos que crear el proyecto dentro de nuestro directorio de trabajo, y del directorio P07-Multimodulo.

De forma automática, IntelliJ nos da la posibilidad de añadir un nuevo módulo a nuestro proyecto (desde la ventana ***Project Structure***), antes de crear el proyecto. Cada ejercicio lo haremos en un módulo diferente. Recuerda que CADA módulo es un PROYECTO MAVEN.

➪ ➪ Ejercicio 1: proyecto multimódulo `hotel`

En el directorio Plantillas-P07A/hotel encontraréis los ficheros que vamos a necesitar para crear nuestro proyecto maven multimódulo, al que llamaremos "hotel".

Nuestro proyecto multimódulo estará formado por una aplicación Web (empaquetada como HotelWebApp.war) y una aplicación java (empaquetada como HotelDatabase.jar). La **Figura1** muestra de forma gráfica la relación entre ambas aplicaciones.

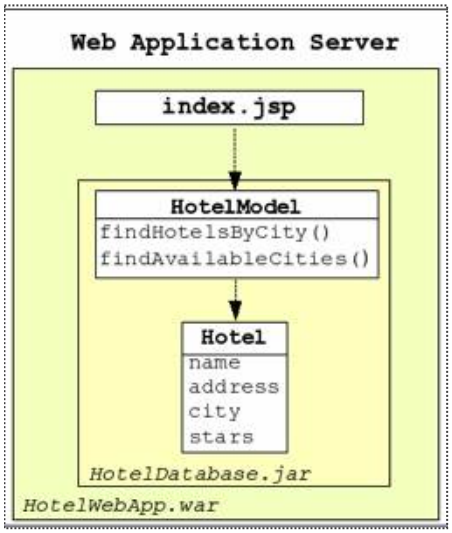


Figura 1. El módulo HotelWebApp depende del módulo HotelDatabase

Vamos a añadir un módulo a nuestro proyecto IntelliJ (***File→New Module)***:

- Seleccionamos ***Maven***, y nos aseguramos de elegir el ***JDK 1.8***
- Seleccionamos "Create from archetype", y buscamos en la lista de arquetipos : "org.codehaus.mojo.archetypes:pom-root". Si no aparece en la lista lo añadiremos desde "Add archetype" (GroupId="org.codehaus.mojo.archetypes"; ArtifactId="pom-root"; Version=1.1).

  Seleccionamos el arquetipo *pom-root* y pulsamos sobre ***Next***

hotel (multi modulo)

- ***Add as a module to***: *<none>*; ***Parent:*** *<none>*.
- ***GroupId***: "*ppss*"; ***ArtifactId:*** *"hotel*".
- ***ModuleName***: "*hotel*". ***Content Root:*** *"$HOME/ppss-Gx-.../P07-Multimodulo/P07-multimodulos/hotel*". ***Module file location***: *"$HOME/ppss-Gx-.../P07-Multimodulo/P07-multimodulos/hotel*".

Finalmente pulsamos sobre OK. Verás que el proyecto maven "hotel" únicamente contiene el fichero pom.xml cuyo empaquetado es "pom".

De forma alternativa, podríamos haber generado el proyecto multimódulo, desde el **terminal**, usando el siguiente comando maven:

```
> cd P07-Multimodulo
> mvn archetype:generate \
        -DarchetypeGroupId=org.codehaus.mojo.archetypes \
        -DarchetypeArtifactId=pom-root \
        -DarchetypeVersion=1.1 \
        -DgroupId=ppss \
        -DartifactId=P07-multimodulos \
        -DinteractiveMode=false
```

Nota: Se trata de un único comando maven que podemos teclear en una única línea. Por claridad, lo hemos escrito en varias líneas. Las "\" se usan para indicar en el terminal que ignore el retorno de carro.

En este caso estamos ejecutando la goal "generate" del plugin "archetype", y necesitamos indicar las coordenadas del arquetipo pom-root, y las coordenadas del proyecto que queremos crear. El arquetipo pom-root nos genera una "plantilla" de un proyecto maven que únicamente contiene un fichero pom.xml con un empaquetado pom.

En realidad nuestro proyecto maven "hotel" todavía no es un proyecto multimódulo. Lo será cuando agreguemos en el pom la lista de módulos.

Ahora vamos a añadir el módulo "HotelDatabase" a nuestro proyecto "hotel". Para ello seleccionamos el proyecto "hotel" y de nuevo usamos (***File→New Module):***

Hotel Database

- Seleccionamos ***Maven***, y nos aseguramos de elegir el ***JDK 1.8***
- ***Add as a module:*** ppss:hotel: 1.0-SNAPSHOT
- ***Parent***: ppss:hotel: 1.0-SNAPSHOT
- ***ArtifactId***: "*HotelDatabase"*.
- ***ModuleName***: "*HotelDatabase*". ***Content Root:*** *"$HOME/ppss-Gx-.../P07-Multimodulo/P07-multimodulos/hotel/HotelDatabase*". ***Module file location***: *"$HOME/ppss-Gx-.../P07-Multimodulo/P07-multimodulos/hotel/HotelDatabasel*".

El código fuente del módulo está formado por los ficheros *Hotel.java*, y *HotelModel.java* (del directorio *Plantillas-P07A/hotel/*).

Finalmente añadimos el módulo "HotelWebApp" al proyecto "hotel". Seleccionamos el proyecto "hotel" y usamos (***File→New Module***):

Hotel WebApp

- Seleccionamos ***Maven***, y nos aseguramos de elegir el ***JDK 1.8***
- Seleccionamos "Create from archetype", y buscamos en la lista de arquetipos : "org.codehaus.mojo.archetypes:webapp-javaee7". Si no aparece en la lista lo añadiremos desde "Add archetype" (GroupId="org.codehaus.mojo.archetypes"; ArtifactId="webapp-javaee7"; Version=1.1).

  Seleccionamos el arquetipo *webapp-javaee7* y pulsamos sobre ***Next***
- ***Add as a module:*** ppss:hotel: 1.0-SNAPSHOT
- ***Parent***: ppss:hotel: 1.0-SNAPSHOT
- ***ArtifactId***: "*HotelWebApp"*.
- ***ModuleName***: "*HotelWebApp*". ***Content Root:*** *"$HOME/ppss-Gx-.../P07-Multimodulo/P07-multimodulos/hotel/HotelWebApp*". ***Module file location***: *"$HOME/ppss-Gx-.../P07-Multimodulo/P07-multimodulos/hotel/HotelWebApp*".

El código fuente de este módulo es el fichero *Plantillas-P07A/hotel/index.jsp*, que deberás copiar en el directorio *src/main/webapp*. Borra el fichero *src/main/webapp/index.html* que te ha creado maven por defecto.

Añade en el pom del proyecto padre (*Hotel*) las propiedades que hemos usado en todos nuestros proyectos maven.

Comprueba que los módulos *HotelDatabase* (con empaquetado jar) y *HotelWebApp* (con empaquetado war) se han agregado como módulos en el proyecto multimódulo *Hotel*, y que además son "hijos" de *hotel*.

Ya hemos comentado que el módulo HotelWebApp "depende" del módulo *HotelDatabase*. Esto significa que tenemos que añadir esta dependencia en el módulo HotelWebApp. Para ello, primero vamos a guardar en nuestro repositorio local el *jar* del módulo *HotelDatabase* (comando *mvn install*, del módulo *HotelDatabase*).

Ahora que ya tenemos el **jar** en nuestro repositorio local, añadiremos dicha dependencia en el *pom* del módulo *HotelWebApp*:

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

Finalmente, vamos a desplegar nuestra aplicación web y ejecutarla. Necesitamos un servidor de aplicaciones. Lo tenéis comprimido en */home/ppss/Descargas/wildfly-11.0.0.Final.tar.gz*. Podéis descomprimirlo en vuestro $HOME. **IMPORTANTE**: NO BORRES el fichero comprimido de la carpeta Descargas!. Lo necesitaremos más adelante.

Añade en el *pom* del proyecto *HotelWebApp* el **plugin wildfly**:

Wildfly

```
<plugin>
    <groupId>org.wildfly.plugins</groupId>
    <artifactId>wildfly-maven-plugin</artifactId>
    <version>2.0.1.Final</version>
    <configuration>
        <hostname>localhost</hostname>
        <port>9990</port>
        <jbossHome>/home/ppss/wildfly-11.0.0.Final</jbossHome>
    </configuration>
</plugin>
```

Hemos configurado el plugin indicando la ruta donde tenemos instalado el servidor de aplicaciones. Los valores de*<hostname>* y *<port>* indican en qué máquina y puerto estará "escuchando" nuestro servidor.

Una cosa más: cuando despleguemos nuestro *war* en el servidor, fíjate que el artefacto generado se llamará *HotelWebApp-1.0-SNAPSHOT.war*. Vamos a cambiarle el nombre, para lo cual añadimos la etiqueta *<finalName>* anidada en la sección *<build>* de nuestro *pom*.

```
<build>
    <!-- Especificamos el nombre del war que será usado como context root
    cuando despleguemos la aplicación -->
    <finalName>${project.artifactId}</finalName>
    <plugins>
    ...
```

Ahora ya podemos desplegar y ejecutar nuestra aplicación web. Para ello tendremos que usar:

```
> mvn clean install   //fases clean e install desde hotel
> mvn wildfly:start //arrancamos el servidor de aplicaciones, desde HotelWebApp
> mvn wildfly:deploy //desplegamos el war generado en el servidor de aplicaciones
> http://localhost:8080/HotelWebApp    //ejecutamos nuestra aplic. web, desde el navegador
> mvn wildfly:shutdown   //cuando queramos detener el servidor
```

➪ ➪ Ejercicio 2: proyecto multimódulo *matriculacion*

En el directorio Plantillas-P07A/matriculacion encontraréis los ficheros que vamos a necesitar para crear nuestro proyecto maven multimódulo, al que llamaremos "matriculacion".

Vamos a añadir un módulo a nuestro proyecto IntelliJ (***File→New Module)***:

matriculacion (multi modulo)

- Seleccionamos ***Maven***, y nos aseguramos de elegir el ***JDK 1.8***
- Seleccionamos "Create from archetype", y buscamos en la lista de arquetipos : "org.codehaus.mojo.archetypes:pom-root". Si no aparece en la lista lo añadiremos desde "Add archetype" (GroupId="org.codehaus.mojo.archetypes"; ArtifactId="pom-root"; Version=1.1).

  Seleccionamos el arquetipo *pom-root* y pulsamos sobre ***Next***
- ***Add as a module to***: *<none>*; ***Parent:*** *<none>*
- ***GroupId***: "*ppss*"; ***ArtifactId:*** "*matriculacion*".
- ***ModuleName***: "*matriculacion*". ***Content Root:*** "*$HOME/ppss-Gx-.../P07-Multimodulo/P07-multimodulos/matriculacion*". ***Module file location***: "*$HOME/ppss-Gx-.../P07-Multimodulo/P07-multimodulos/matriculacion*".

Añade al pom creado los valores de las propiedades que hemos usado en todos los ejercicios de prácticas.

El proyecto matriculación será un proyecto multimódulo, formado por los siguientes cuatro módulos:

Nombre del módulo (artifactId)	Paquete que contiene los fuentes
matriculacion-comun	ppss.matriculacion.to
matriculacion-dao	ppss.matriculacion.dao
matriculacion-proxy	ppss.matriculacion.proxy
matriculacion-bo	ppss.matriculacion.bo

Cada uno de los módulos, además, serán "hijos" de *matriculacion*.

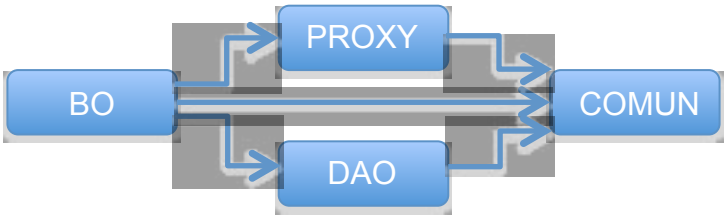
creamos los 4 módulos

Crea los módulos tal y como hemos explicado en el ejercicio anterior (recuerda que físicamente deberán ser subdirectorios del proyecto padre.

Nota: IntelliJ te marcará como errores todos los usos de los objetos *TO. No te preocupes. Se solucionarán cuando incluyamos las dependencias entre los módulos.

Incluye en los ficheros pom correspondientes, las **dependencias** reflejadas en la siguiente figura (por ejemplo BO → PROXY, se "lee" como "el módulo matriculacion-bo depende del módulo matriculacion-proxy).

dependencias



Finalmente vamos a construir el proyecto. De momento vamos a ejecutar "mvn package" (desde el proyecto "matriculacion".

Observa la salida por pantalla de maven y entiende bien por qué se realizan las acciones de construcción en ese orden. Haz una lista de todos los artefactos generados y dónde se han generado en cada una de las fases ejecutadas.

Necesitarás este proyecto multimódulo para la siguientes sesión de prácticas en la que añadiremos y ejecutaremos tests unitarios y de integración.