

P06A- Proyectos Maven multimódulo

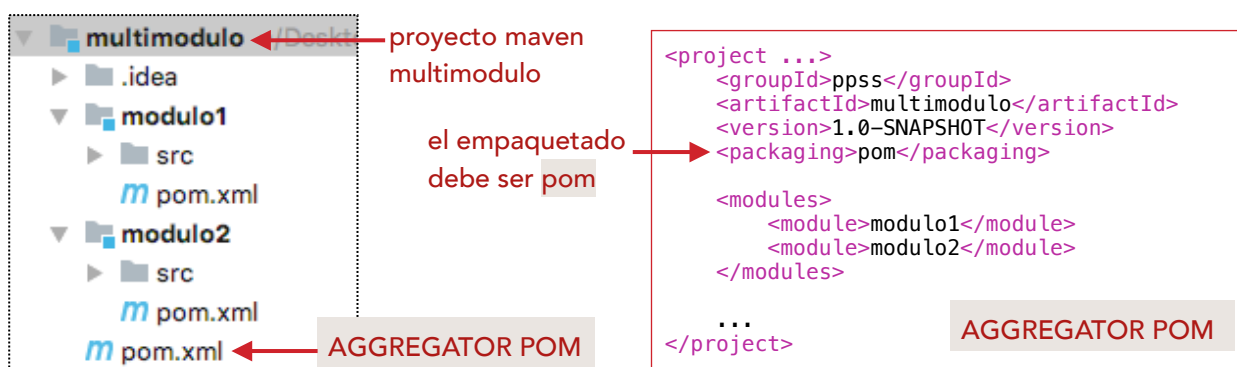
Proyectos Maven multimódulo

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 necesitemos, 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 orden 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.

Mostramos un ejemplo de construcción de nuestro proyecto multimódulo. En este caso vamos a ejecutar el comando:

```
mvn compile.
```

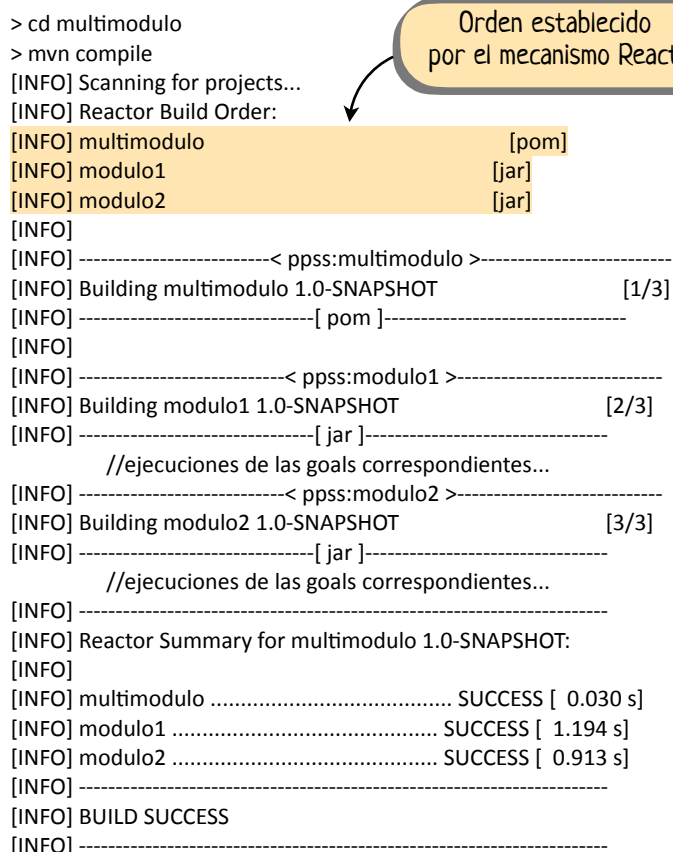
Podemos observar que se ha establecido un orden entre los tres módulos:

- 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, el ciclo de vida por defecto solamente tiene asociadas las goals *install:install*, y *deploy:deploy* a las fases *install* y *deploy*, respectivamente.

Por lo tanto, para el proyecto que contiene el *aggregator pom*, no ejecutará ninguna acción en el resto de fases durante el proceso de construcción.



```
> 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 ]-----
[INFO]
//ejecuciones de las goals correspondientes...
[INFO] -----< ppss:modulo2 >-----
[INFO] Building modulo2 1.0-SNAPSHOT [3/3]
[INFO] -----[ jar ]-----
[INFO]
//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, o plugins, entre los módulos agregados usando el mecanismo de **herencia** que nos proporciona maven.

Maven soporta la relación de herencia, por lo que podemos crear un pom que nos sirva para identificar a un proyecto “padre”. Al incluir cualquier configuración en el pom del proyecto padre, sus módulos “hijo” la heredarán, 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>`.

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

  <artifactId>modulo1</artifactId>

  ...
</project>
```

modulo1/pom.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>, <version>,... Por eso no es necesario añadir las coordenadas <groupId> y <version> en los “hijos” si van 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.

GitHub

El trabajo de esta sesión también debes subirlo a *GitHub*. Todo el trabajo de esta práctica deberá estar en el directorio **P06A-Multimodulo**, dentro de tu espacio de trabajo.

Ejercicios

En esta sesión vamos a crear un proyecto IntelliJ, el cual contendrá DOS proyectos Maven multimódulo.

En el directorio **Plantillas-P06A**, os proporcionamos las carpetas con los ficheros que usaremos en cada uno de los ejercicios.

Primero crearemos un proyecto IntelliJ **vacío** e iremos añadiendo los módulos (proyectos Maven multimódulo), que correspondan en cada caso.

NOTA: Cuidado: no confundas el proyecto IntelliJ con el proyecto maven, son completamente diferentes. El proyecto IntelliJ es simplemente la carpeta contenedora todo nuestro trabajo. Nuestro "proyecto IntelliJ" puede contener proyectos maven "simples", como hemos usado hasta ahora, o proyectos maven multimódulo, una mezcla de ambos, o incluir además, cualquier otro tipo de proyecto (proyectos NO maven).

Los proyectos contenidos en nuestro proyecto IntelliJ también se llaman módulos, pero no tienen nada que ver con los módulos de un proyecto maven multimódulo! Recordad que los nombres que las herramientas que usamos dan a los diferentes elementos pueden confundiros. Hay que tener claro en todo momento qué estamos haciendo y por qué lo estamos haciendo así y no de otra forma. Cuidado con eso!!!

Para **crear el proyecto IntelliJ**:

proyecto
IntelliJ
multimodulos

- **File→New Project.** Seleccionamos “Empty Project”
- **Project name:** **multimodulos**. **Project Location:** **\$HOME/ppss-2024-Gx-.../P06A-Multimodulo/**

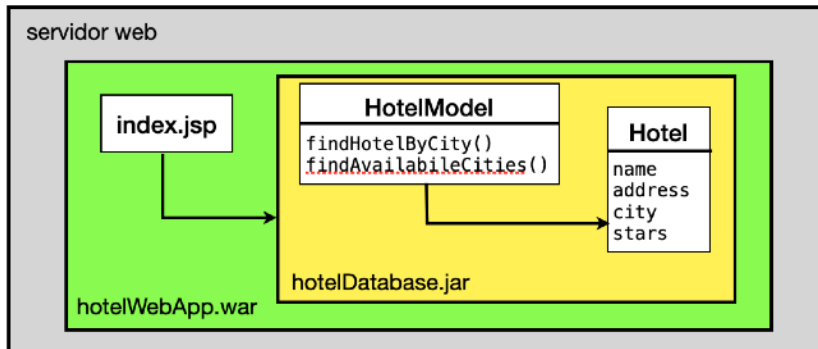
Recuerda que debes **ELIMINAR** el módulo *P06A-Multimodulo* y el fichero *P06A-Multimodulo.iml*

➡ Ejercicio 1: proyecto multimódulo *hotel*

Nuestro proyecto multimódulo **hotel** estará formado por:

- una aplicación Web (que empaquetaremos como *hotelWebApp.war*), y
- una aplicación java (que empaquetaremos como *hotelDatabase.jar*).

A continuación mostramos de forma gráfica en la **Figura 1** la relación entre ambas aplicaciones.



La aplicación **hotelWebApp** se ejecutará en un contenedor web de un servidor web o un servidor de aplicaciones. Por eso tendrá un empaquetado **war**.

La aplicación **hotelDatabase** formará parte del fichero war, y tendrá un empaquetado **jar**.

Figura 1. Componentes de la aplicación multimódulo hotel

Ambas aplicaciones son proyectos maven que pueden desarrollarse de forma independiente por diferentes grupos de trabajo. Pero ambos proyectos maven forman parte de la misma aplicación.

Recordemos que hemos creado un proyecto IntelliJ vacío con nombre **multimodulos**.

Añadimos un nuevo módulo (**File→Project Structure→ + → NewModule...**) y seleccionamos Maven Archetype.

- **Name:** *hotel*
- **Location:** *"\$HOME/ppss-2024-Gx-.../P06A-Multimodulo/multimodulos/"*
- Elegimos **JDK 17**
- **Parent:** *<none>*
- **Catalog:** *Maven Central*
- **Archetype:** *org.codehaus.mojo.archetypes:pom-root ; Version: 1,1*
- Desde **Advanced Settings**, indicamos los valores **GroupId:** *ppss* ; **ArtifactId:** *hotel*

proyecto
multimódulo
hotel

En esta ocasión, hemos generado un proyecto maven multimódulo a partir de un "arquetipo" (o plantilla).

Abre el pom del proyecto y verás que únicamente contiene sus coordenadas. Fíjate que el empaquetado ahora es "pom" (no "jar" como hemos usado hasta ahora). Observa también que nuestro proyecto maven no tiene ninguna carpeta "src").

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

```
> cd P06A-Multimodulo/multimodulos
> mvn archetype:generate \
-DarchetypeGroupId=org.codehaus.mojo.archetypes \
-DarchetypeArtifactId=pom-root \
-DarchetypeVersion=1.1 \
-DgroupId=ppss \
-DartifactId=hotel \
-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. El carácter "\ " se usa para indicar que el comando continúa en la siguiente línea.

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.

Añadimos un nuevo módulo (**File→New→Module...**) a nuestro proyecto multimódulo.

- Asegúrate que está seleccionado **New Module** en el panel de la izquierda.
- **Name:** *hotelDatabase*
- **Location:** *"\$HOME/ppss-2024-Gx-.../P06A-Multimodulo/multimodulos/hotel/"*
- Seleccionamos **Java**, y **Maven**, y nos aseguramos de elegir el **JDK 17**
- **Parent:** *hotel*
- Desmarcamos la casilla *Add sample code*
- Desde **Advanced Settings**, **GroupId:** *ppss*; **ArtifactId:** *hotelDatabase*

módulo
hotelDatabase

Comprueba que en el *pom* del nuevo módulo se ha generado la etiqueta *<parent>*.

El código fuente del módulo **hotelDatabase** está formado por los ficheros *Hotel.java*, y *HotelModel.java* (del directorio *Plantillas-P06A/hotel/*).

Finalmente añadimos un segundo módulo “hotelWebApp” al proyecto “hotel” (**File→New Module**):

- Asegúrate que está seleccionado **New Module** en el panel de la izquierda.
- **Name:** *hotelWebApp*
- **Location:** *"\$HOME/ppss-2024-Gx-.../P06A-Multimodulo/multimodulos/hotel/"*
- Seleccionamos **Java**, y **Maven**, y nos aseguramos de elegir el **JDK 17**
- **Parent:** *hotel*
- Desmarcamos la casilla *Add sample code*
- Desde **Advanced Settings**, **GroupId:** *ppss*; **ArtifactId:** *hotelWebApp*

módulo
hotelWebApp

Comprueba de que en el *pom* del nuevo módulo se ha generado la etiqueta *<parent>*.

Tendrás que cambiar el empaquetado del proyecto hotelWebApp a **war**. Y debes añadir la carpeta **webapp** en el directorio *src/main*.

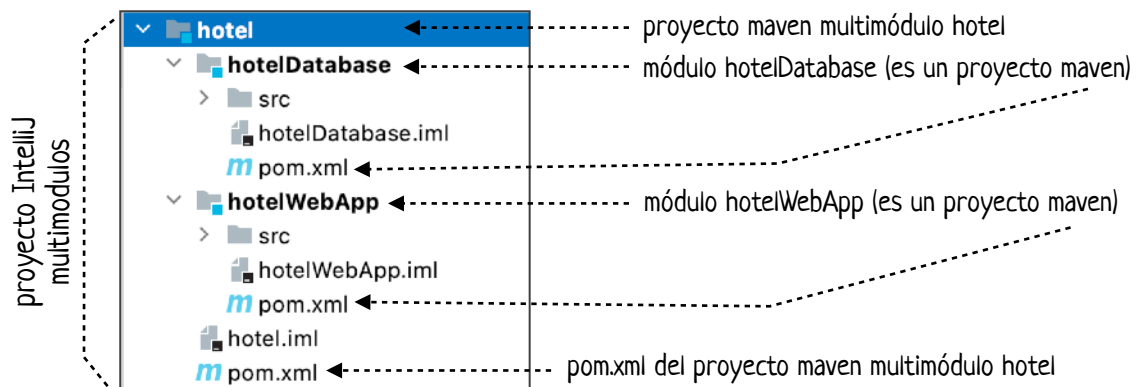
El código fuente del módulo **hotelWebApp** es el fichero *Plantillas-P06A/hotel/index.jsp*, que deberás copiar en el directorio *src/main/webapp*.

En el **pom** del proyecto *hotelWebApp*:

- Añade el plugin maven-war-plugin con la versión 3.4.0:


```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-war-plugin</artifactId>
  <version>3.4.0</version>
  <configuration>
    <failOnMissingWebXml>>false</failOnMissingWebXml>
  </configuration>
</plugin>
```

A continuación mostramos la estructura de tu **proyecto IntelliJ multimodulos** en este momento:



Los proyectos maven *hotelDatabase* y *hotelWebApp* se han añadido como módulos del proyecto maven multimódulo *hotel*. Además, los proyectos *hotelDatabase* y *hotelWebApp* son hijos del proyecto *hotel*. Compruébalo.

Añade en el *pom* del proyecto multimódulo (*hotel*) las **propiedades** que hemos usado en todos nuestros proyectos maven anteriores y revisa las propiedades de los dos módulos hijo para evitar redundancias. Cambia la versión del plugin *compiler* por defecto por la versión 3.12.1, tal y como hemos hecho en las prácticas anteriores.

Observa que el *pom* del proyecto *hotel* contiene la lista con los módulos "hotelDatabase" y "hotelWebApp".

El módulo *hotelWebApp* "depende" del módulo *hotelDatabase*, tal y como se muestra en la **Figura 1**. Por lo tanto tenemos que añadir esta dependencia en el *pom* del módulo *hotelWebApp*.

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

El fichero "hotelDatabase-\${project.version}.jar" indicado en la dependencia anterior se generará y se guardará en el repositorio local cuando ejecutemos la fase *install* del proyecto *hotelDatabase*.

Finalmente, vamos a desplegar nuestra aplicación web y ejecutarla. Necesitamos un servidor de aplicaciones. Lo descargaremos desde <https://github.com/wildfly/wildfly/releases/download/26.1.2.Final/wildfly-26.1.2.Final.zip>. Descomprime el fichero en tu \$HOME.

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



```
<plugin>
  <groupId>org.wildfly.plugins</groupId>
  <artifactId>wildfly-maven-plugin</artifactId>
  <version>4.2.2.Final</version>
  <configuration>
    <hostname>localhost</hostname>
    <port>9990</port>
    <jbossHome>/home/ppss/wildfly-26.1.2.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, el artefacto generado se llamará *hotelWebApp-1.0-SNAPSHOT.war*. Vamos a cambiarle el nombre (para que sólo sea "hotelWebApp"), 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*
el *pom* de "hotel" contiene la lista de módulos agregados. Las fases "clean" e "install" se ejecutan para todos los módulos en el orden determinado por el mecanismo reactor
- > 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

Ejecuta los comandos maven desde la **ventana Maven** de IntelliJ. Verás que, para cada proyecto maven se muestran las fases más comúnmente usadas, los *plugins*,... Para ejecutar sólo las *goals* también puedes hacerlo desde la ventana Maven de IntelliJ (desde los *plugins* de cada proyecto).

Si quieres que se muestren todas las fases del ciclo de vida por defecto (más alguna de los otros dos ciclos), desde el icono con tres puntos en vertical, en la parte superior derecha de la ventana Maven, debes desmarcar la opción "Show Basic Phases Only".

🔗 Ejercicio 2: proyecto multimódulo *matriculacion*

En el directorio **Plantillas-P06A/matriculacion** encontraréis los ficheros que vamos a necesitar para crear nuestro proyecto maven multimódulo, al que llamaremos "**matriculacion**".

Añadimos un nuevo módulo (**File→Project Structure→ + → NewModule...**) y seleccionamos Maven Archetype.

Proyecto
multimódulo
matriculacion

- **Name:** *matriculacion*
- **Location:** *"\$HOME/ppss-2024-Gx-.../P06A-Multimodulo/multimodulos/"*
- Elegimos **JDK 17**
- **Parent:** *<none>*
- **Catalog:** *Maven Central*
- **Archetype:** *org.codehaus.mojo.archetypes:pom-root ; Version: 1,1*
- Desde **Advanced Settings**, indicamos los valores **GroupId:** *ppss ; ArtifactId:* *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 (el groupId de todos ellos es ppss):

Nombre del módulo (artifactId)	Paquete que contiene los fuentes	ficheros con los fuentes (carpeta <i>Plantillas-P06A</i>)
matriculacion-comun	ppss.matriculacion.to	en matriculacion/to
matriculacion-dao	ppss.matriculacion.dao	en matriculacion/dao
matriculacion-proxy	ppss.matriculacion.proxy	en matriculacion/proxy
matriculacion-bo	ppss.matriculacion.bo	en matriculacion/bo

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

Crea los módulos tal y como hemos explicado en el ejercicio anterior (recuerda que físicamente deberán ser subdirectorios del proyecto padre). No debes usar ningún "arquetipo" maven. Usa los ficheros de las plantillas para añadir el código fuente de cada uno de los módulos.

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

Modifica el pom de cada módulo para incluir las **dependencias** reflejadas en la **Figura 2** (por ejemplo BO → PROXY, se "lee" como "el módulo matriculacion-bo depende del módulo matriculacion-proxy").

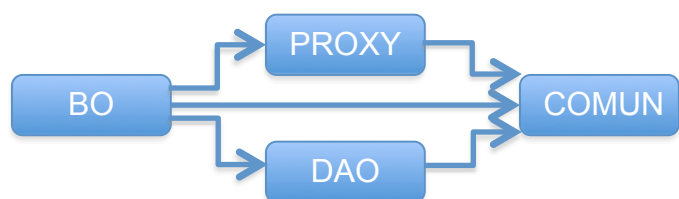


Figura 2. Dependencias para el proyecto *matriculación*

Recuerda, además, que deberás cambiar la versión del plugin "compiler" a la 3.10.1 en el pom del proyecto *matriculacion*, y añadir las propiedades que hemos usado en todas las prácticas. Revisa todos los pom.xml para evitar redundancias

En la **Figura 3** mostramos la vista *Project* de nuestro proyecto **multimodulos**, que contiene dos proyectos maven multimódulo: *hotel* y *matriculacion*

El proyecto **matriculacion** es un proyecto multimódulo (tiene empaquetado pom), y contiene los 4 módulos (proyectos maven) de la Figura 2.

Puesto que hemos "agregado" los cuatro módulos en el pom del proyecto *matriculacion*, cuando ejecutemos un comando maven desde el directorio *matriculacion*, dicho comando se ejecutará para todos los módulos agregados.

No tendremos que preocuparnos de las dependencias entre los módulos gracias al mecanismo reactor de maven, que los ordenará convenientemente.

Es conveniente que tengas esta vista, tal cual se muestra en la **Figura 3**, para que veas más fácilmente lo que ocurre cuando construimos el proyecto *matriculacion*.

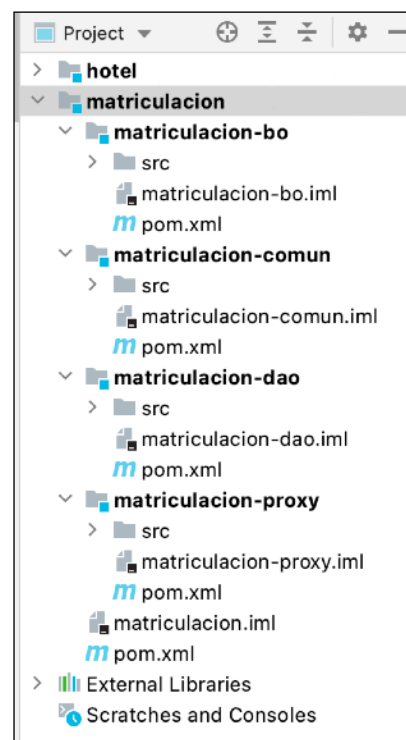


Figura 3. Vista *Project* del proyecto *multimodulos*

Finalmente vamos a construir el proyecto. Realiza las siguientes acciones:

- ejecuta la fase **compile** desde el proyecto *matriculacion*: verás que en la vista *Project* aparece el directorio *target* en todos los módulos del proyecto.
- en la vista *Project*, muestra el contenido de los directorios *target* de los módulos de *matriculacion*. Y a continuación ejecuta la fase **package** desde ese proyecto. Verás que en el *target* se generan los .jar de cada módulo.
- si ahora ejecutas la fase **clean** desde el proyecto *matriculacion*, verás que se borran todos los "targets" de los módulos

Cada uno de los módulos, también puede construirse por separado, vamos a comprobarlo:

- ejecuta la fase **compile** desde el proyecto *matriculacion-comun*: verás que en la vista *Project* aparece el directorio *target* sólo en ese proyecto.
- ahora ejecuta la misma fase desde el proyecto *matriculacion-dao*. Verás que no compila!!! Al estar ejecutando los proyectos por separado, fíjate que *matriculacion-dao* depende de *matriculacion-comun*. Lo deberías tener indicado en el pom en forma de dependencia. Por lo tanto, maven, busca el artefacto de *matriculacion-comun* en nuestro repositorio local. Como no lo encuentra, intentará buscar en la nube, y tampoco lo va a encontrar, y por lo tanto se generará un error de compilación, ya que en el código fuente de *matriculacion-dao* se usan las clases de *matriculacion-comun*.

Tendrás que ejecutar la fase **install** desde *matriculacion* para que se generen todos los artefactos de cada módulo y se guarden en nuestro repositorio local. Comprueba que están ahí.

Una vez hecho esto, puedes, por ejemplo, ejecutar de nuevo **clean** sobre el proyecto *matriculacion*. Y a continuación vuelve a ejecutar la fase **compile** desde el proyecto *matriculacion-dao*. Ahora debe compilar sin problema.

Observa la salida por pantalla de maven y entiende bien por qué se realizan las acciones de construcción en ese orden. Recuerda que debes tener claro qué ficheros y/o artefactos se generan en cada una de las fases (y qué goals los generan).

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

Resumen



¿Qué **conceptos** y **cuestiones** me deben quedar CLAROS después de hacer la práctica?



PROYECTOS INTELLIJ

- Un proyecto IntelliJ puede contener diferentes tipos de proyectos: proyectos maven, java, android, servicios rest, servicios web..., cada uno de ellos se denomina **MÓDULO**.
- Un proyecto IntelliJ no se construye ya que no contiene código, el proceso de construcción se realiza para cada uno de sus módulos.

PROYECTOS MAVEN MULTIMÓDULO

- Un proyecto **MAVEN** puede ser "single" (single-maven project) o puede contener, a su vez, varios proyectos maven. (multimodule maven project), cada uno de los cuales se denomina **MODULO**.
- El proyecto multimódulo tiene un empaquetado pom, en el que se pueden **AGREGAR** tantos módulos (proyectos maven) como se necesite. El mecanismo de agregación permite ejecutar un comando maven (una vez), y se ejecutará automáticamente en todos los módulos agregados, en el orden determinado por las dependencias entre ellos, e identificado por el mecanismo **REACTOR** de maven.
- Se pueden establecer adicionalmente relaciones de **HERENCIA** entre los módulos de un proyecto multimódulo, de esta forma pueden compartir propiedades, plugins, y dependencias.
- En un proyecto multimódulo se pueden usar ambas relaciones (herencia y agregación) o sólo una de ellas.
- Los módulos de un proyecto multimódulo también pueden construirse de forma individual: un módulo es un proyecto maven con un empaquetado diferente de pom: puede ser jar, war, ear.