Programación Frontend y Backend

BLOQUE JAVA

Maven

GOBIERNO DE ESPAÑA
MINISTERIO DE INDUSTRIA, COMERCIO Y TURISMO
EOI Escuela de organización industrial
Unión Europea
Fondo Social Europeo
Iniciativa de Empleo Juvenil
El FSE invierte en tu futuro
SISTEMA NACIONAL DE GARANTÍA JUVENIL

Maven

GOBIERNO DE ESPAÑA
MINISTERIO DE INDUSTRIA, COMERCIO Y TURISMO
EOI Escuela de organización industrial

Unión Europea
Fondo Social Europeo
Iniciativa de Empleo Juvenil
El FSE invierte en tu futuro

SISTEMA NACIONAL DE GARANTÍA JUVENIL

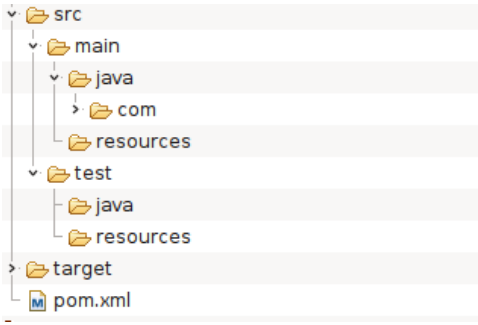
MAVEN

- ¿Qué es Maven?
- Instalación de Maven
- Creación nuevo proyecto
- Ciclo de vida de los proyectos Maven
- Repositorio de Maven
- Gestión de Dependencias
- Maven compile plugin
- Centralizando configuración en el pom padre
- Ejercicios

GOBIERNO DE ESPAÑA MINISTERIO DE INDUSTRIA, COMERCIO Y TURISMO
EOI Escuela de organización industrial
Unión Europea
Fondo Social Europeo
Iniciativa de Empleo Juvenil
El FSE invierte en tu futuro
SISTEMA NACIONAL DE GARANTÍA JUVENIL

MAVEN

- **Maven** es una herramienta *open source* para ***administrar proyectos*** de software. Es decir ***gestionar el ciclo de vida*** desde la creación de un proyecto en un lenguaje dado, hasta la generación de un binario que pueda distribuirse con el proyecto.
- Todos los proyectos Maven tienen por default la misma estructura de directorios.
- Avanzada gestión en el manejo de dependencias, informes sobre tests automáticos y extensibilidad a través de ***plugins***.
- Es una herramienta de gestión y comprensión de software basada en el concepto de ***Modelo de Objeto de Proyecto*** (***POM***) que puede gestionar la generación de proyectos, la generación de informes y la documentación a partir de un punto central.



MAVEN

Instalación

1. Descarga del sitio de Maven la última versión distribuida en .zip, que contiene la distribución binaria: http://maven.apache.org/download.html

2. Descomprime este archivo en cualquier ubicación de tu disco duro local.

3. Configurar la variable de ambiente **JAVA_HOME** con el path de la instalación del JDK.

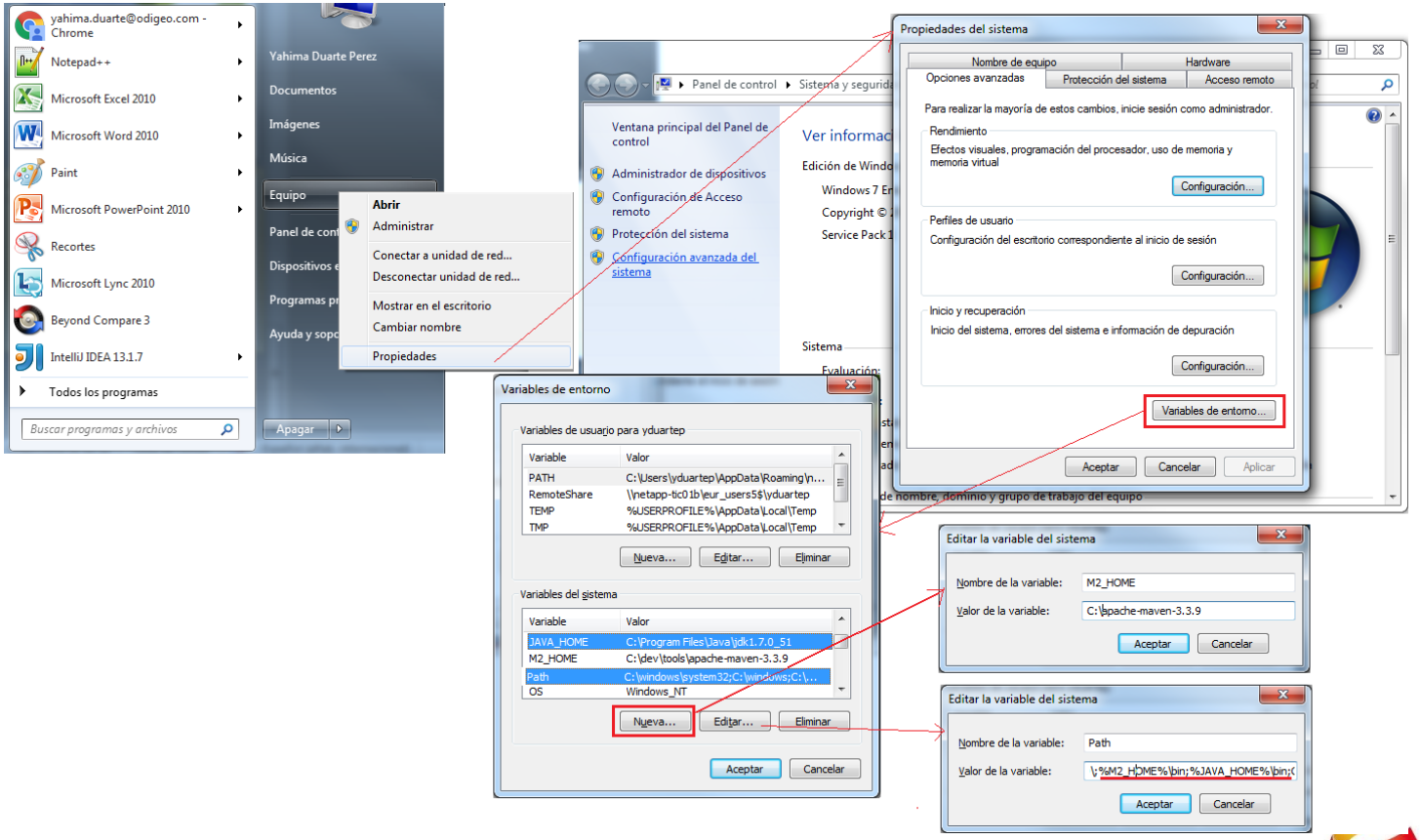
4. Configurar la variable de ambiente **M2_HOME** con el path de la instalación de Maven (*C:\apache-maven-3.3.9*).

5. Configurar la carpeta **bin** de ambas en la variable de ambiente **Path** agregando al final de la línea: %**M2_HOME**%**bin**;%**JAVA_HOME**%**bin**;

6. Abre una nueva consola de comandos y ejecuta el comando:

mvn -version.

```
C:\Users\yduartep>mvn -version
Apache Maven 3.3.9 (bb52d8502b132ec0a5a3f4c09453c07478323dc5; 2015-11-10T17:41:4
7+01:00)
Maven home: C:\dev\tools\apache-maven-3.3.9
Java version: 1.7.0_51, vendor: Oracle Corporation
Java home: C:\Program Files\Java\jdk1.7.0_51\jre
Default locale: es_ES, platform encoding: Cp1252
OS name: "windows 7", version: "6.1", arch: "amd64", family: "windows"
```



MAVEN

Un Proyecto en Maven se define mediante un archivo **POM**, llamado **pom.xml** con una etiqueta inicial de <**project**>. En dicho archivo se definen cosas como las *instrucciones para compilar* el proyecto, las *librerías* necesarias, etc.

En Maven, la ejecución de un archivo POM siempre genera un "artefacto“ que puede ser de tipo: **jar, war, ear, swf** de flash**, zip** o el mismo archivo **pom** para organizar el proyecto y definir características comunes a los proyectos hijos que se creen a partir de el como por ejemplo: ***nombre*** del proyecto o el número de ***versión***, las ***dependencias***, los ***repositorios***, etc.

MAVEN

Para crear un proyecto en Maven puedes hacerlo a mano siguiendo ciertas convenciones:

- El nombre de la carpeta debe ser igual al nombre del proyecto.
- A nivel raíz de la carpeta debe estar el archivo **pom.xml** con la descripción del proyecto que puede ser creado a mano o usando ***archetypes*** predefinidos.

Usando el siguiente comando, aparecerá la lista completa de artefactos existentes en el repositorio central de Maven:

  – *mvn archetype:generate*

Si no quieres que te aparezca la lista, puedes definir el arquetipo desde línea de comandos tecleando:

mvn archetype:generate
-DarchetypeGroupId=org.apache.maven.archetypes
-DarchetypeArtifactId=maven-archetype-quickstart

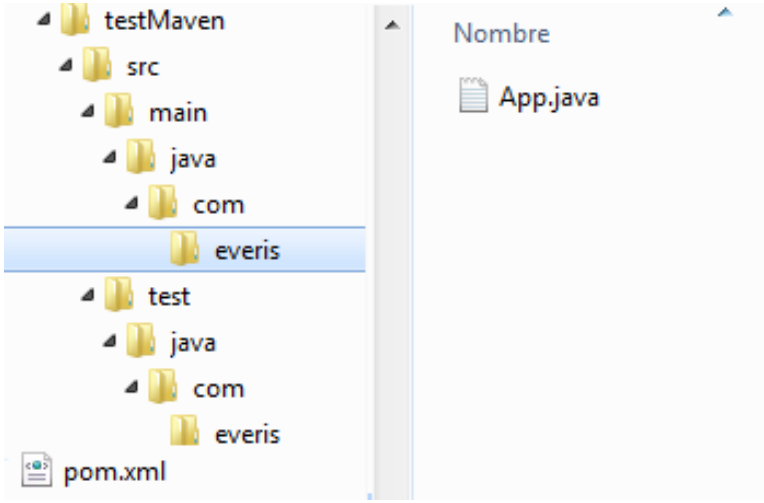
MAVEN

El arquetipo te pedirá varios atributos sobre tu proyecto:

- **groupId**: Piensa en él como en el paquete de proyecto. Típicamente aquí se pone el nombre de tu empresa u organización (com.everis).
- **artifactId**: Es el nombre de tu proyecto (testMaven).
- **version**: Número de versión de tu proyecto (1.0-SNAPSHOT).
- **package**: Paquete base donde irá tu código fuente (com.everis).

MAVEN

Una vez generado el proyecto la estructura de carpetas quedaría




```xml
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.everis</groupId>
  <artifactId>testMaven</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>testMaven</name>
  <url>http://maven.apache.org</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

MAVEN

Existen 3 ciclos de vida en Maven:

- **clean**. Elimina las clases compiladas y los archivos binarios generados del proyecto.
- **default**. Genera los archivos binarios (artefactos) de nuestro proyecto.
- **site**. Genera archivos html que describen nuestro proyecto.

Para ejecutarlos, excepto para el default, se debe de poner **mvn ciclo**:

- **mvn clean** o **mvn site**

- Cada ciclo de vida está constituido por varias fases.

- Antes de ejecutar cualquier fase del ciclo **default**, se suele usar el comando **clean** para limpiar todo lo que fue creado por las compilaciones anteriores seguido de la fase deseada. Ejemplo si deseamos compilar e instalar nuestro proyecto en el repositorio local de Maven se ejecuta:

- **mvn clean install**

mvn fase1 fase2:goal1 fase2:goal2

MAVEN

Cada ciclo de vida está constituido por varias ***fases*** y el ciclo por **defecto** incluye las siguientes etapas:

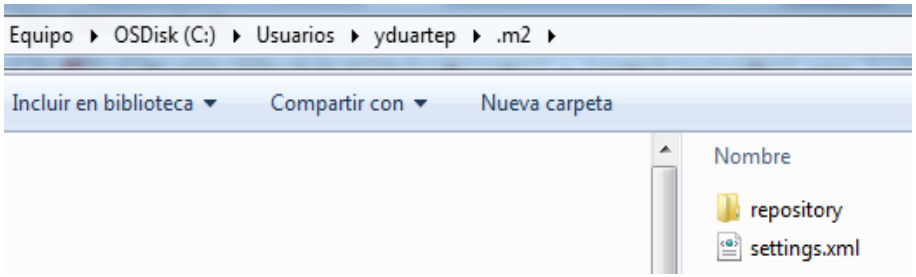
- **validate**: Validar que el proyecto es correcto.
- **initialize**: Configura propiedades y crea directorios.
- **compile**: Compilación del código fuente.
- **test**: ejecuta los test unitarios definidos en la carpeta ***src/test/java***.
- **package**: Para empaquetar el código compilado y transformarlo en algún formato tipo **.jar, .war o .ear**.
- **integration-test**: Procesar y desplegar el código en algún entorno donde se puedan ejecutar las pruebas de integración.
- **verify**: Verificar que el código empaquetado es válido y cumple los criterios de calidad.
- **install**: Instalar el código empaquetado en el repositorio local de Maven (.m2), para usarlo como dependencia de otros proyectos.
- **deploy**: Para desplegar el código a un entorno especifico.

MAVEN

¿Qué es el repositorio local? Maven descarga *sus dependencias* y las dependencias de *tus proyectos* de un repositorio central. Por default, usa el repositorio central de Maven (**http://repo1.maven.org/maven2**) aunque puedes definir que use otros incluido alguno que tu hayas

creado dentro de tu intranet.

Estas dependencias las almacena en un repositorio local que se encuentra normalmente en la carpeta usuarios bajo el nombre **.m2** con el fin de no tener que descargarlas otra vez.



MAVEN

Es recomendable, cambiar en Windows el repositorio de usuario y ponerlo en una carpeta en cuya ruta ***no existan espacios***, para evitar problemas con algunos plugins.

La ubicación por defecto se cambia en el fichero **settings.xml** que se encuentra en la carpeta **conf** al interno de la carpeta de instalación de Maven bajo la voz localRepository:

- ***C:\apache-maven-3.3.9\conf\settings.xml***

```
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
          http://maven.apache.org/xsd/settings-1.0.0.xsd">
  <!-- localRepository
   | The path to the local repository maven will use to store artifacts.
   |
   | Default: ${user.home}/.m2/repository
  <localRepository>/path/to/local/repo</localRepository>
  -->
<localRepository>C:/Users/yduartep/.m2/repository</localRepository>
```

MAVEN

En el repositorio local **.m2** se descargan los **arquetipos** que vamos usando en los proyectos desde el ***repositorio central*** de Maven.

Las librerías no presente en tal repositorio, se pueden descargar desde un ***nuevo repositorio*** configurado desde **pom.xml** o desde **settings.xml**.

- Podemos introducir la configuración en el **pom.xml** así**:**

```
<repositories>
   <repository>
      <id>Spark repository</id>
      <url>https://oss.sonatype.org/content/repositories/snapshots/</url>
  </repository>
</repositories>
```

- Para añadir el repositorio en el **settings.xml** hay

- Hay que hacerlo a través de un perfil, que debemos dejar activo

```
<profiles>
...
  <profile>
    <id>perfil-repo</id>
    <repositories>
      <repository>
        <id>nuevo-repositorio</id>
        <name>Un nuevo repositorio</name>
        <url>http://direccion.al.nuevo.repositorio.com</url>
      </repository>
    </repositories>
  </profile>
...
 </profiles>
 <activeprofiles>
    <activeprofile>perfil-repo</activeprofile>
 </activeprofiles>
```

MAVEN

Si la librería que necesitamos no esta presente en ningún repositorio, podemos descargar el **jar** y añadirlo al repositorio local usando el comando:

mvn install:install-file

- -**Dfile**={Path/al/fichero/ojdbc.jar}
- **-DgroupId**=com.oracle
- **-DartifactId**=ojdbc6
- **-Dversion**=11.2.0
- **-Dpackaging**=jar

Toda compañía debería disponer de un repositorio centralizado, también llamado***, repositorio corporativo*** donde publicar todas las librerías que se van desarrollando en la compañía por distintos proyectos.

Existen varias soluciones en lo que concierne a la gestión de repositorios, los más utilizados para crear repositorios corporativo son:

- Archiva de Apache
- Artifáctory de JFrog
- Nexus de Sonatype, que son los creadores de Maven

MAVEN

Para definir una nueva dependencia en Maven debemos definir en el **pom**:

- groupId
- artifactId
- version
- scope (que puede ser compile | test | runtime…).

```
1    <dependency>
2      <groupId>com.mycompany.app</groupId>
3      <artifactId>my-app</artifactId>
4      <version>1.0-SNAPSHOT</version>
5      <scope>compile</scope>
6    </dependency>
```

Para buscar una dependencia especifica se va en el repositorio http://mvnrepository.com/, se busca desde ahí usando el nombre, selecciono la versión especifica que me interesa y copio de ahí el **xml** con la descripción de la dependencia.

MAVEN


Search for groups, artifacts, categories
Search
Home » commons-io » commons-io » 2.5
Apache Commons IO » 2.5
The Apache Commons IO library contains utility classes, stream implementations, file filters, file comparators, endian transformation classes, and much more.
License
Apache 2.0
Categories
I/O Utilities
HomePage
http://commons.apache.org/proper/commons-io/
Date
(Apr 22, 2016)
Files
Download (JAR) (269 KB)
Repositories
Central
Used By
9,152 artifacts
Maven
Gradle
SBT
Ivy
Grape
Leiningen
Buildr
<!-- https://mvnrepository.com/artifact/commons-io/commons-io -->
<dependency>
<groupId>commons-io</groupId>
<artifactId>commons-io</artifactId>
<version>2.5</version>
</dependency>
Include comment with link to declaration

MAVEN

Las dependencias se pueden definir con 6 tipos de **scope** diferentes:

- **compile** → Por defecto. Las dependencias de compilación están disponibles en todos los classpaths de un proyecto. Por otra parte, esas dependencias se propagan a proyectos dependientes.

- **provided** → Es similar a la compilación, pero indica que espera encontrar la dependencia del JDK o un contenedor en tiempo de ejecución. Por ejemplo, al crear una aplicación web Java Enterprise, se debe establecer la dependencia de la APIs Servlet y Java EE relacionadas con el alcance provided ya que el contenedor web proporciona esas clases. Este ámbito de aplicación sólo está disponible en la compilación y test, y no es transitiva.

MAVEN

- **runtime** → Indica que la dependencia no se necesita para compilar, pero si en tiempo de ejecución. Está en el classpath de tiempo de ejecución y test classpaths, pero no en el de compilación.
- **test** → Indica que esta dependencia no es necesaria para el uso normal de la aplicación y sólo está disponible para las fases de compilación y ejecución de los test.
- **system** → Similar a provided con la excepción de que tienes que proporcionar el JAR de manera explícita. El artefacto está siempre disponible y no se encuentra en los repositorios.
- **import** → Únicamente utilizado en dependencias de tipo pom . De este modo se importan las dependencias definidas en otro pom

MAVEN

Respecto a la **versión** a utilizar en una dependencia, se puede especificar una versión en concreto de una librería, por ejemplo:

- **<version>1.0.23</version>**

Si se define una dependencia de tipo **SNAPSHOT**, se está indicando que se está trabajando con una ***librería en desarrollo***, Maven comprobará si hay versiones SNAPSHOT más actualizadas en el repositorio, y en caso de que así sea, se la descargará.

Cada vez que actualizamos una versión **SNAPSHOT**, Maven almacena en el repositorio cada una de estas versiones, sustituyendo "SNAPSHOT" por el ***timestamp*** del momento en el que la actualizamos:

- **<version>1.5.0-SNAPSHOT</version>**

```
8d1b5a3e60d796869ea0da5cda1b1516208fac35 1.0.1
618303ee9d23a177561c5f365db48910fdf45121 1.2.1
2146adf455cc1b2001a247da301873fdf743fd7e 1.2.2
5140c207b5256ca5a151e423031d8881a3215b05 1.3.0
1c5fbb2079d0167677ce2840699fda7c38c7067a 1.4.0
fa09290b22b6b34029f2d2a73cc065698a354b90 1.4.1
cce36d54ebb0156a4f73cb64c0ff995697ba8d82 1.5.0
```

MAVEN

También se pueden especificar **rangos de versiones**, de manera que intente recuperar la versión más alta que esté dentro del rango:

<version>[1.0,1.9]**</version>** Descarga version mas alta en este rango.

<version>[1.0,)**</version>** Descarga mínimo version 1.0 o una mayor si aparece.

<version>[1.0,1.14)**</version>** Descarga mínimo version 1.0 o una mayor si aparece hasta la version 1.14.

Cuando introducimos una dependencia, y ésta a su vez **depende** de otras librería ya definida en el pom, esto podría crear conflicto de versiones por lo que se suele **excluir** la dependencia que no necesitamos:

```
<dependency>
   <groupId>junit</groupId>
   <artifactId>junit</artifactId>
   <version>4.11</version>
   <scope>test</scope>
   <exclusions>
     <exclusion>
       <artifactId>hamcrest-core</artifactId>
       <groupId>org.hamcrest</groupId>
     </exclusion>
   </exclusions>
</dependency>
```

MAVEN

Por default, Maven **compila** usando como target java 1.4. Así que una de las tareas que hay que hacer casi siempre, es modificar esto para indicarle que usaremos una versión superior de java.

Esto se define desde el pom.xml usando el plugin **maven-compiler-plugin**:

Compila tu proyecto con **mvn install** y verifica que la compilación se realiza con la version de java especificada.

```
<build>
        <plugins>
                <plugin>
                        <groupId>org.apache.maven.plugins</groupId>
                        <artifactId>maven-compiler-plugin</artifactId>
                        <configuration>
                                <source>1.5</source>
                                <target>1.5</target>
                        </configuration>
                </plugin>
        </plugins>
</build>
```

MAVEN

Usar un pom padre permite **centralizar configuración** común de tus módulos.

- **packaging** será de tipo “pom”.
- En **properties** se deben definir las características comunes de todos los módulos como por ejemplo version de una librería especifica:

```
<groupId>org.javahispano</groupId>
<artifactId>tutorial_maven</artifactId>
<packaging>pom</packaging>
<version>1.0</version>

<properties>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  <log4j.version>1.2.9</log4j.version>
</properties>
<dependencies>
  <dependency>
        <groupId>log4j</groupId>
        <artifactId>log4j</artifactId>
        <version>${log4j.version}</version>
  </dependency>
</dependencies>
<modules>
  <module>tutorial_maven_jar</module>
  <module>tutorial_maven_webapp</module>
</modules>
```

MAVEN

Además de las variables personalizadas, Maven tiene ***variables de proyecto***:

```
<groupId>${parent.groupId}</groupId>
<version>${parent.version}</version>
```

Se usan para guardar el **groupId** y la **version** del proyecto y usarlas en archivos pom hijos. Cambiando el pom.xml del padre cambia automáticamente el **groupId** y **version** del hijo.

MAVEN

1. Crear un proyecto Maven con una clase **Converter** que tenga un método **toFahrenheitFromCelcius**(double centigrados) que recibe como parámetro los grados centígrados y devuelva los grados en Fahrenheit. La formula correspondiente es: F = 32 + (9 * C / 5) .
2. Definir dentro de la clase **Converter** otro método **toCelciusFromFahrenheit**(double fahrenheit) que recibe como parámetro los grados en Fahrenheit y devuelva los grados centígrados. La formula correspondiente es: T(°C) = (T(°F) - 32) × 5/9.
3. Generar el artefacto de salida (**converter-1.0-SNAPSHOT.jar**).
4. Crear una clase de test que pruebe la función toFahrenheit, toCelcius y lanzarlos.
5. Instalar el proyecto con la clase **Converter** en el repositorio local.
6. Crear un nuevo proyecto Tiempo que contenga clase Temperatura. Definir en esta clase el método **displayTemperatura** (Unidad from, Unidad to, double valor) que imprima por consola:
   - **displayTemperatur**a(Unidad.Celcius, Unidad.Fahrenheit, 20) -> La temperatura es 68 °F.
   - **displayTemperatur**a(Unidad.Fahrenheit, Unidad.Celcius, 104) -> La temperatura es 40 °C.

Usar la libreria **converter** definida en el punto 1 para hacer la conversion antes de imprimir el mensaje final. El paquete final se llamara **temperatura-1.0-SNAPSHOT.jar**

MAVEN

7. Crear dos funciones nuevas **toKelvinFromCelcius**(double celcius) y **toCelciusFromKelvin**(double fahreinheit) en la clase **Converter** del proyecto Maven creado en el ejercicio 1, que convierta de kelvin a Celcius y viceversa usando las formulas: T(K) = T(°C) + 273.15 y T(°C) = T(K) - 273.15.
8. Modificar la version del proyecto aumentando en 1 la version actual que esta en desarrollo.
9. Generar el artefacto de salida usando la nueva version definida.
10. Modifica la clase de test para que pruebe las dos funciones nuevas y lanzarlos.
11. Instalar el proyecto con la clase **Converter** en el repositorio local. Ahora deben aparecer las dos versiones.
12. En el proyecto Tiempo agregar un nuevo caso que imprima por consola:
    - **displayTemperatur**a(Unidad.Kelvin, Unidad Celcius, 300) -> La temperatura es 26.85 °C.
    - **displayTemperatur**a(Unidad Celcius, Unidad.Kelvin, 60) -> La temperatura es 333.15 K.

Usar la libreria **converter** definida en el punto 1 para hacer la conversion antes de imprimir el mensaje final.