

Herramientas de construcción de software

1- Instalar Java JDK si no dispone del mismo.

2- Instalar Maven

3- Que es maven?

- Es una herramienta de software para la gestión y construcción de proyectos java, posee la capacidad de realizar diferentes tareas, como compilación y empaquetado del código, hace posible la creación de software con dependencias incluidas dentro de la estructura jar, se definen las dependencias en un archivo POM.xml que se nombra en las siguientes.

Que es POM?

- POM, sus siglas Project Object Model, es una representación XML de un proyecto en Maven, donde contiene la información del proyecto, como configuraciones para la compilación, dependencias que se instalan en el proyecto de librerías externas.

1.modelVersion

- Contiene la versión del modelo de POM

```
<modelVersion>4.0.0</modelVersion>
```

2.groupId

- Define el dominio del proyecto que estamos realizando, se divide en varios segmentos, el primer segmento es el dominio, generalmente en org,com,cn, etc, depende el fin de la organización, básicamente es el proyecto al cual pertenece el proyecto maven actual, los artifacts tienen un groupId

```
<groupId>com.Wedoll</groupId>
```

3.artifactId

- Define el módulo maven, es el nombre del jar sin incluir la versión.

```
<artifactId>Wedoll</artifactId>
```

4.versionID

- La versión del artifact.

```
<version>0.0.1-SNAPSHOT</version>
```

- Los anteriores nombrados en conjunto forman el artifact.
-

Repositorios Local, Central y Remotos

<http://maven.apache.org/guides/introduction/introduction-to-repositories.html>

- **Repositorio Local:** los repositorios locales son directorios que están en la computadora donde se corre maven, cachea las descargas remotas y contiene los artifacts de compilación temporales

- **Repositorio remoto:** Cualquier otro repositorio que se accede por protocolos como file:// y https://. Cuando se necesita un artifact de estos repos, se descarga el repositorio local del desarrollador y después se utiliza.
- **Repositorio Central:** esta localizado en <http://repo.maven.apache.org/maven2/>, cuando este compila, maven encuentra la dependencia en el repo local y si no se encuentra ahí, lo descarga desde este repositorio central.
- **Entender Ciclos de vida de build**
 - default:** Maneja el despliegue del proyecto, ejecuta una secuencia de fases para completar el ciclo default.
 - clean:** realiza una limpieza del proyecto, elimina archivos generados y luego hace el despliegue
 - site:** Crea la web del proyecto

- Comprender las fases de un ciclo de vida, por ejemplo, default:

Fase de build	Descripción
validate	valida si el proyecto está correcto y toda la información está disponible
compile	compila el código fuente del proyecto
test	prueba el código fuente compilado utilizando un marco de prueba de unidad adecuado. Estas pruebas no deberían requerir que el código se empaquete o implemente
package	toma el código compilado y lo empaqueta en su formato distribuible, como un JAR.
verify	ejecuta cualquier verificación de los resultados de las pruebas de integración para garantizar que se cumplan los criterios de calidad
install	instal1 el paquete en el repositorio local, para usarlo como dependencia en otros proyectos localmente
deploy	hecho en el entorno de compilación, copia el paquete final en el repositorio remoto para compartirlo con otros desarrolladores y proyectos.

- Copiar el siguiente contenido a un archivo, por ejemplo `./trabajo-practico-02/maven/vacio/pom.xml`

```

Abrir  pom.xml  Guardar
~/Escritorio/ing se 3/IsW3-JuanPabloRojas/TPN4

1 <project xmlns="http://maven.apache.org/POM/4.0.0"
2   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
4     http://maven.apache.org/xsd/maven-4.0.0.xsd">
5   <modelVersion>4.0.0</modelVersion>
6
7   <groupId>ar.edu.ucc</groupId>
8   <artifactId>proyecto-01</artifactId>
9   <version>0.1-SNAPSHOT</version>
10 </project>

```

- Ejecutar el siguiente comando en el directorio donde se encuentra el archivo pom.xml

```

juan-pablo@juanpablo-OMEN-Laptop-15-ek0xxx:~/Escritorio/ing se 3/Isw3-JuanPablo
ojas/TPN4$ mvn clean install
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by com.google.inject.internal.cglib.core.$Re
lectUtils$1 (file:/usr/share/maven/lib/guice.jar) to method java.lang.ClassLoad
r.defineClass(java.lang.String,byte[],int,int,java.security.ProtectionDomain)
WARNING: Please consider reporting this to the maintainers of com.google.inject
internal.cglib.core.$ReflectUtils$1
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflec
ive access operations
WARNING: All illegal access operations will be denied in a future release
[INFO] Scanning for projects...
[INFO]
[INFO] -----< ar.edu.ucc:proyecto-01 >-----
[INFO] Building proyecto-01 0.1-SNAPSHOT
[INFO] -----[ jar ]-----
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven
plugins/maven-clean-plugin/2.5/maven-clean-plugin-2.5.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/

```

```

Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven
plugins/maven-plugins/23/maven-plugins-23.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/
plugins/maven-plugins/23/maven-plugins-23.pom (9.2 kB at 9.0 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven
plugins/maven-resources-plugin/2.6/maven-resources-plugin-2.6.jar
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/
plugins/maven-resources-plugin/2.6/maven-resources-plugin-2.6.jar (30 kB at 21 k
/s)

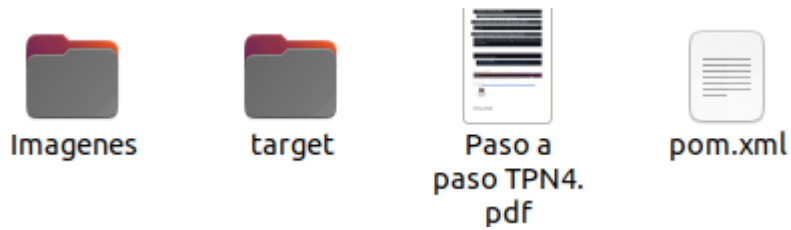
```

```

[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 02:03 min
[INFO] Finished at: 2022-09-07T14:51:15-03:00
[INFO] -----

```

- **Sacar conclusiones del resultado:**
Vemos que se ejecuta una fase de clean al proyecto y después comienza la compilación del mismo, desde 0, luego desde el repo central se descargaron una serie de archivos y se crea el proyecto de maven.



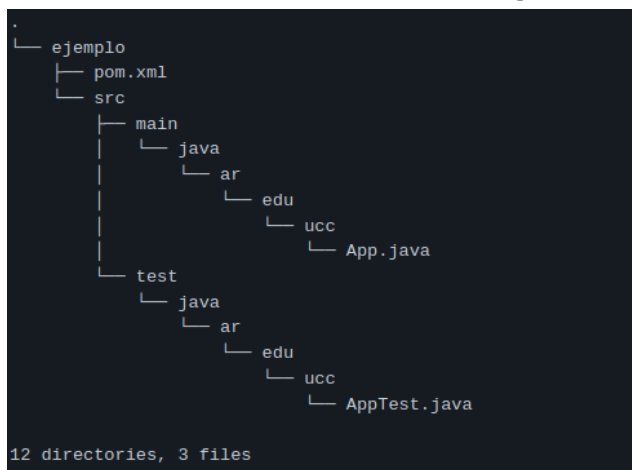
en el screen de arriba podemos ver los archivos que se crearon apartir del clean ejecutado hacia el pom.xml, donde apartir del artifact que se almacena en un repo, si tiramos el clean, despues me va a hacer un build del proyecto con esa configuración del pom.xml y me lo va a descargar en mi directorio.

Entonces,El siguiente comando **mvn clean install** hace que maven haga un clean de todos los módulos antes de ejecutar el install action para los módulos, entonces limpia cualquier archivo compilado que tengas asi te aseguras que realmente estas compilando de 0. Luego despues de la limpieza, me instalo

4- Maven Continuación

- Generar un proyecto con una estructura inicial:
`mvn archetype:generate -DgroupId=ar.edu.ucc -DartifactId=ejemplo -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false`

Analizar la estructura de directorios generada:



- **Compilar el proyecto**
mvn clean package

```
<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/maven-
v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>ar.edu.ucc</groupId>
  <artifactId>ejemplo</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>ejemplo</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
  <build>
    <plugins>
      <plugin> <groupId>org.apache.maven.plugins</groupId> <artifactId>maven-compiler-plugin</-
artifactId> <version>3.8.0</version> </plugin>
    </plugins>
  </build>
</project>
```

tuve que hacer un downgrade en la versión de jdk, sacado de un foro de git issues

- You are using JDK9 or later
- Your project uses **maven-compiler-plugin** with an old version which defaults to Java 5.

You have three options to solve this

1. Downgrade to **JDK7** or **JDK8** (meh)
2. Use **maven-compiler-plugin** version or later, because

NOTE: Since 3.8.0 the default value has changed from 1.5 to 1.6
 See <https://maven.apache.org/plugins/maven-compiler-plugin/compile-mojo.html#target>

```
<plugin> <groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-compiler-plugin</artifactId>
<version>3.8.0</version> </plugin>
```

output de compilar el proyecto:

```
-----
T E S T S
-----
Running ar.edu.ucc.AppTest
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.006 sec
Results :
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
```

- **Analizar la salida del comando anterior y luego ejecutar el programa**

Se corren los tests, se descargaron algunos archivos del repo central, se creo un target

java -cp target/ejemplo-1.0-SNAPSHOT.jar ar.edu.ucc.App

```
juan-pablo@juanpablo-OMEN-Laptop-15-ek0xxx:~/Escritorio/Proyecto-maven/ejemplo$  
java -cp target/ejemplo-1.0-SNAPSHOT.jar ar.edu.ucc.App  
Hello World!
```

Este comando imprime "Hello world!"

6- Manejo de dependencias

- **Crear un nuevo proyecto con artifactId ejemplo-uber-jar**

**mvn archetype:generate -DgroupId=ar.edu.ucc -DartifactId=ejemplo-uber.jar
-DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false**

- **Modificar el código de App.java para agregar utilizar una librería de logging:**

- **Compilar el código e identificar el problema.**

mvn clean package

```
[ERROR] Failed to execute goal org.apache.maven.plugins:maven-compiler-plugin:3.8.0:compile (default-compile) on project ejemplo-uber.jar: Compilation failure:
Compilation failure:
[ERROR] /home/juan-pablo/Escritorio/Proyecto-Maven2/ejemplo-uber.jar/src/main/java/ar/edu/ucc/App.java:[3,17] package org.slf4j does not exist
[ERROR] /home/juan-pablo/Escritorio/Proyecto-Maven2/ejemplo-uber.jar/src/main/java/ar/edu/ucc/App.java:[4,17] package org.slf4j does not exist
[ERROR] /home/juan-pablo/Escritorio/Proyecto-Maven2/ejemplo-uber.jar/src/main/java/ar/edu/ucc/App.java:[14,9] cannot find symbol
[ERROR]   symbol:   class Logger
[ERROR]   location: class ar.edu.ucc.App
[ERROR] /home/juan-pablo/Escritorio/Proyecto-Maven2/ejemplo-uber.jar/src/main/java/ar/edu/ucc/App.java:[14,22] cannot find symbol
[ERROR]   symbol:   variable LoggerFactory
[ERROR]   location: class ar.edu.ucc.App
[ERROR] -> [Help 1]
[ERROR]
[ERROR] To see the full stack trace of the errors, re-run Maven with the -e switch.
[ERROR] Re-run Maven using the -X switch to enable full debug logging.
[ERROR]
[ERROR] For more information about the errors and possible solutions, please read the following articles:
```

El problema es que en el output me dice que no se pudo encontrar el simbolo de la clase Logger ni LoggerFactory y la localización.

el problema es que no se agrego la dependencia al pom.xml

Podemos ver que en el directorio con el jar en su lugar

```

uber.jar$ tree
.
├── pom.xml
├── src
│   ├── main
│   │   └── java
│   │       ├── ar
│   │       │   ├── edu
│   │       │   │   └── ucc
│   │       │   │       App.java
│   └── test
│       ├── java
│       │   ├── ar
│       │   │   ├── edu
│       │   │   │   └── ucc
│       │   │   │       AppTest.java
│       └── resources
│           └── log4j.properties
└── target
    ├── classes
    │   ├── ar
    │   │   ├── edu
    │   │   │   └── ucc
    │   │   │       App.class
    ├── ejemplo-uber.jar-1.0-SNAPSHOT.jar
    ├── generated-sources
    │   └── annotations
    ├── generated-test-sources
    │   └── test-annotations
    ├── maven-archiver
    │   └── pom.properties
    ├── maven-status
    │   └── maven-compiler-plugin
    │       ├── compile
    │       │   ├── default-compile
    │       │   │   ├── createdFiles.lst
    │       │   │   └── inputFiles.lst
    │       └── testCompile
    │           ├── default-testCompile
    │           │   ├── createdFiles.lst
    │           │   └── inputFiles.lst
    ├── surefire-reports
    │   ├── ar.edu.ucc.AppTest.txt
    │   └── TEST-ar.edu.ucc.AppTest.xml
    └── test-classes
        ├── ar
        │   ├── edu
        │   │   └── ucc
        │   │       AppTest.class
        └── resources
            └── log4j.properties
  
```

java -cp target/ejemplo-uber-jar-1.0-SNAPSHOT.jar ar.edu.ucc.App

- **Sacar conclusiones y analizar la salida:**
java -cp ejemplo-uber.jar-1.0-SNAPSHOT.jar ar.edu.ucc.App


```

juan-pablo@juanpablo-OMEN-Laptop-15-ek0xxx:~/Escritorio/Proyecto-Maven2/ejemplo-uber.jar/target$ java -cp ejemplo-uber.jar-1.0-SNAPSHOT.jar ar.edu.ucc.App
Exception in thread "main" java.lang.NoClassDefFoundError: org.slf4j/LoggerFactory
    at ar.edu.ucc.App.main(App.java:14)
Caused by: java.lang.ClassNotFoundException: org.slf4j.LoggerFactory
    at java.base/jdk.internal.loader.BuiltinClassLoader.loadClass(BuiltinClassLoader.java:581)
    at java.base/jdk.internal.loader.ClassLoaders$AppClassLoader.loadClass(ClassLoaders.java:178)
    at java.base/java.lang.ClassLoader.loadClass(ClassLoader.java:522)
    ... 1 more

```

Podemos observar que por mas que hayamos añadido la dependencia en el `pom.xml`, se intenta importar `LoggerFactory` que no se encuentra porque la especificación va a ser útil cuando se compile, entonces habria que buscar la forma de vincular la clase. Una forma puede ser una forma de ejecutar no solo la app sino librerías y build del `pom.xml` que muchas veces nos aconseja el ide.

Ahora, ejecutar la clase con el siguiente comando (en windows reemplazar `$HOME` por `%USERPROFILE%`, y separar por `;` en lugar de `:`)

```

java -cp
target/ejemplo-uber-jar-1.0-SNAPSHOT.jar:$HOME/.m2/repository/org/slf4j/slf4j-api/1.7.22/slf4j-api-1.7.22.jar:$HOME/.m2/repository/ch/qos/logback/logback-classic/1.2.1/logback-classic-1.2.1.jar:$HOME/.m2/repository/ch/qos/logback/logback-core/1.2.1/logback-core-1.2.1.jar ar.edu.ucc.App

```

en el comando de arriba cambie el nombre del jar que se crea

salida:

```

juan-pablo@juanpablo-OMEN-Laptop-15-ek0xxx:~/Escritorio/Proyecto-Maven2/ejemplo-uber.jar$ java -cp target/ejemplo-uber-jar-1.0-SNAPSHOT.jar:$HOME/.m2/repository/org/slf4j/slf4j-api/1.7.22/slf4j-api-1.7.22.jar:$HOME/.m2/repository/ch/qos/logback/logback-classic/1.2.1/logback-classic-1.2.1.jar:$HOME/.m2/repository/ch/qos/logback/logback-core/1.2.1/logback-core-1.2.1.jar ar.edu.ucc.App
20:38:18.099 [main] INFO ar.edu.ucc.App - Hola Mundo!

```

- Implementar la opción de uber-jar:
<https://maven.apache.org/plugins/maven-shade-plugin/>
`java -cp target\ejemplo-uber-jar-1.0-SNAPSHOT.jar ar.edu.ucc.App`

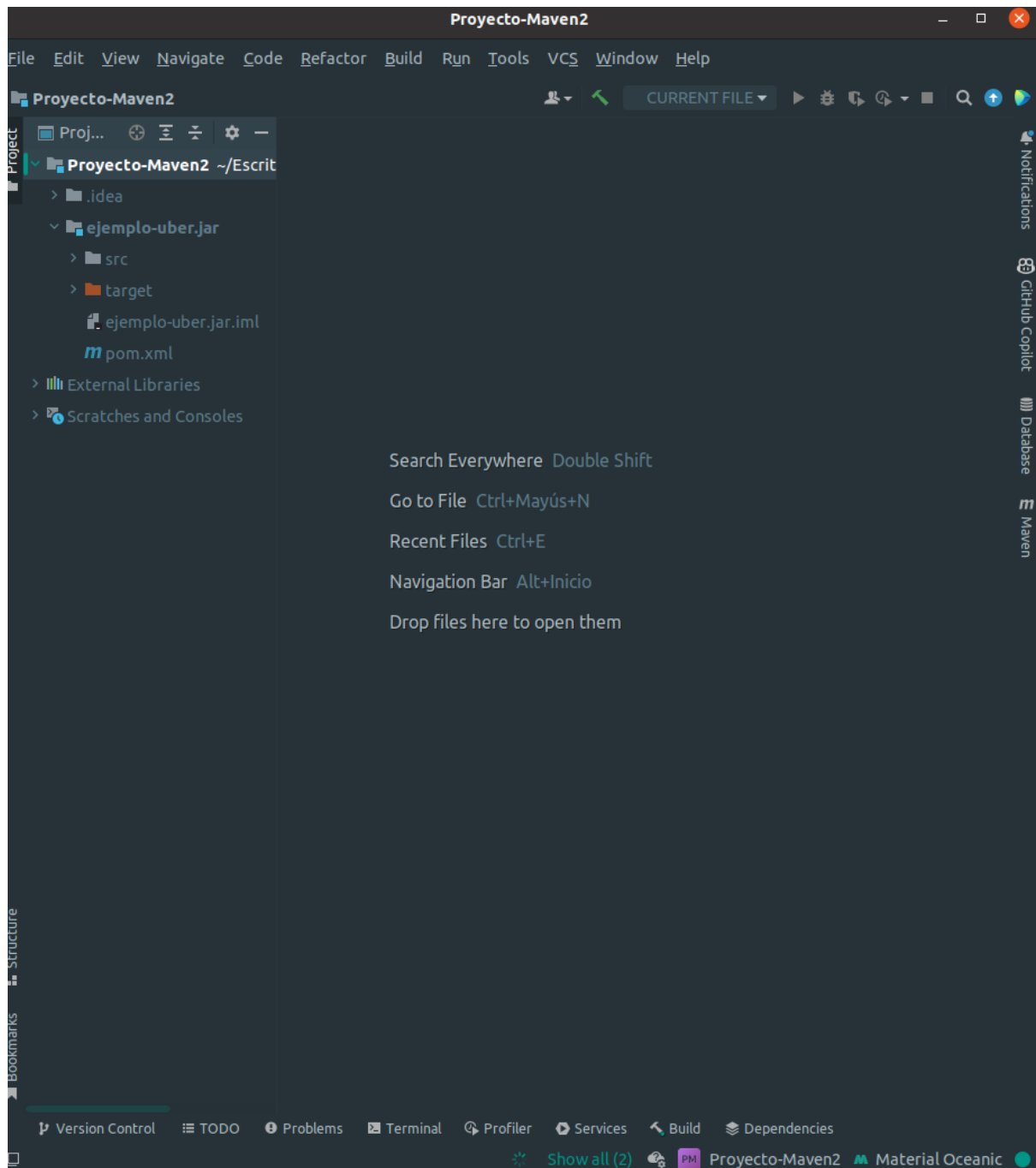
ver porque no anda esta poronga

7- Utilizar una IDE

Importar el proyecto anterior en Eclipse o IntelliJ como maven project:

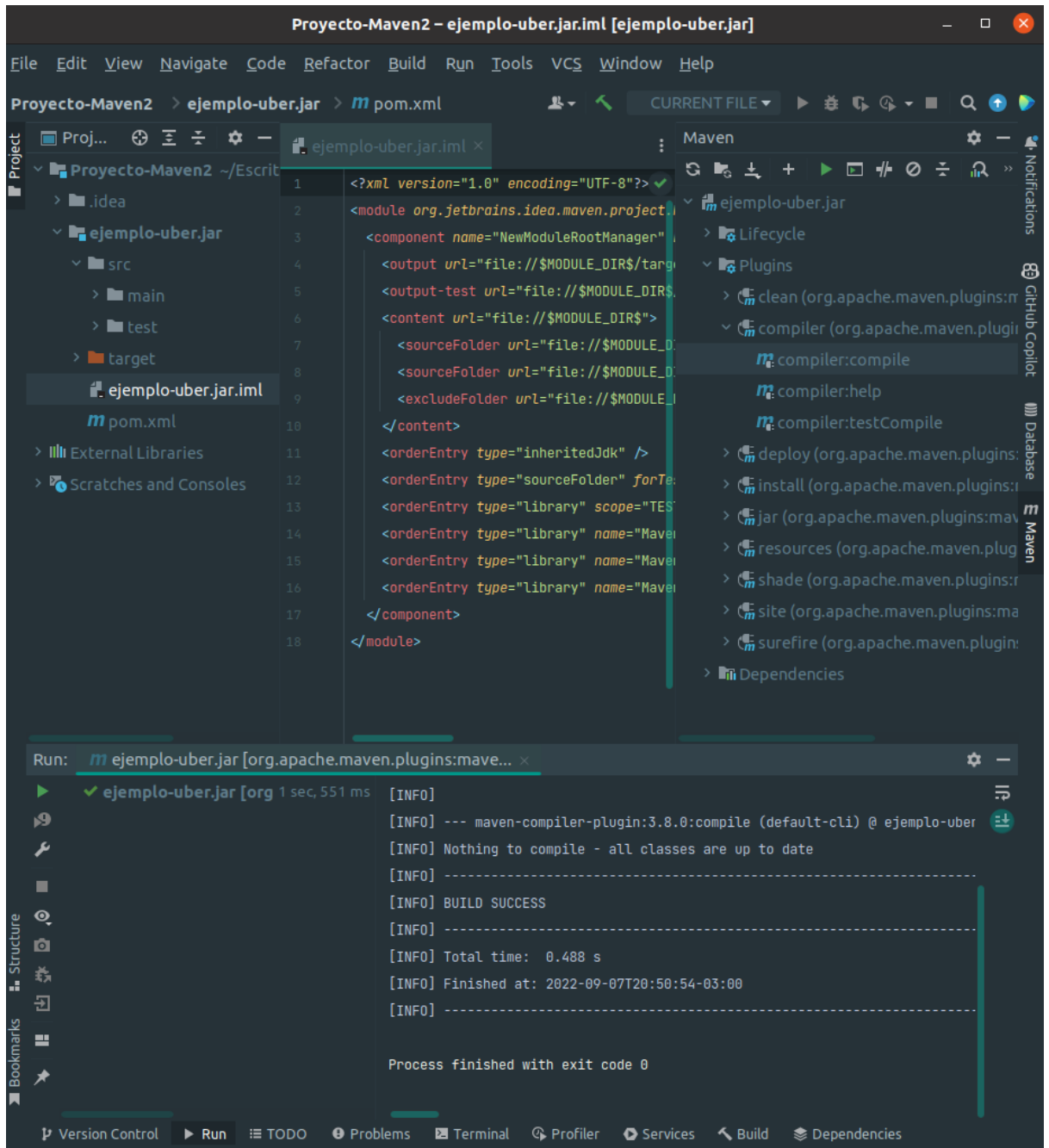
En mi caso intelliJi

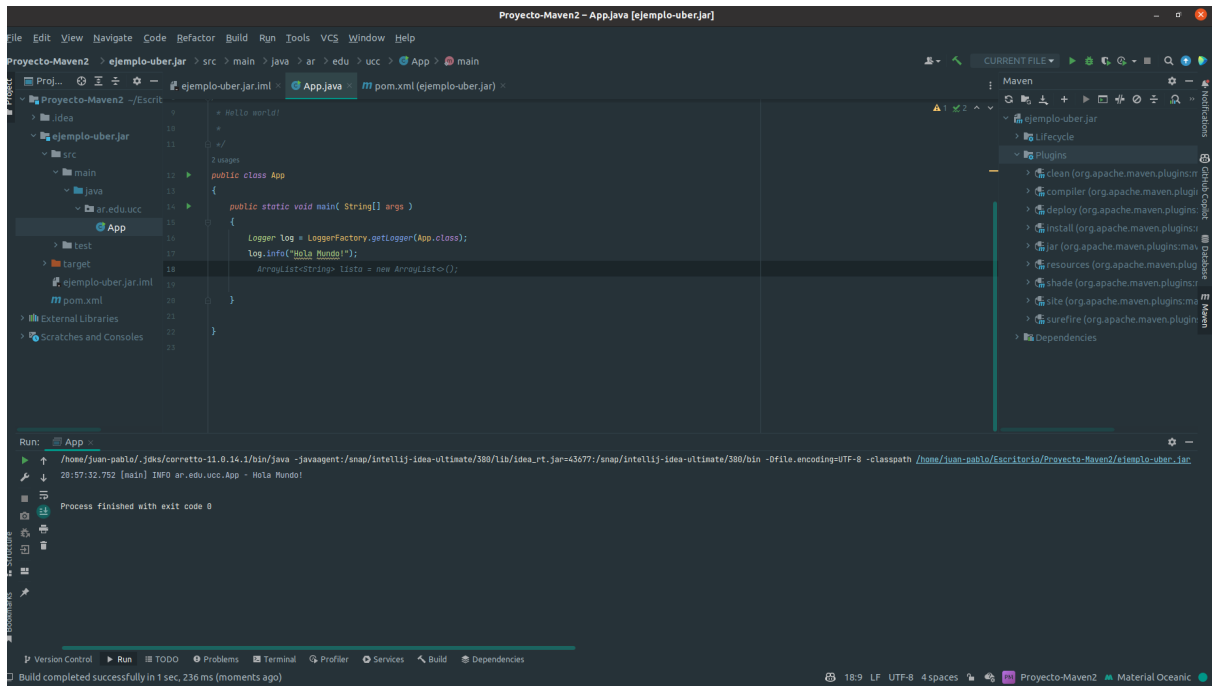
Familiarizarse con la interfaz grafica



- **Ejecutar la aplicación**

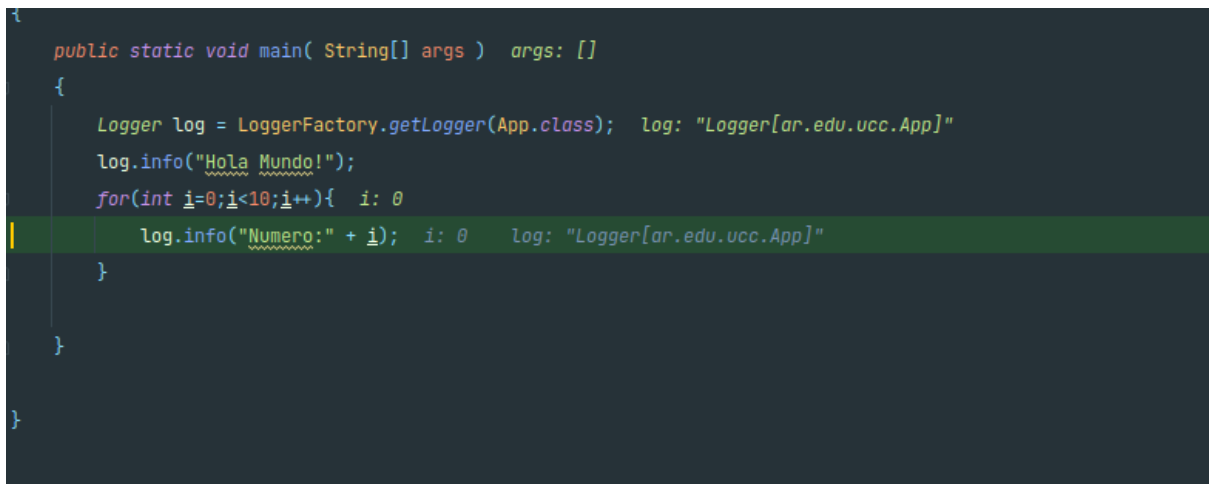
-Para ejecutar la aplicación podemos ver al costado que tenemos una opción de maven, en un momento me pregunto si quería cargar el jar



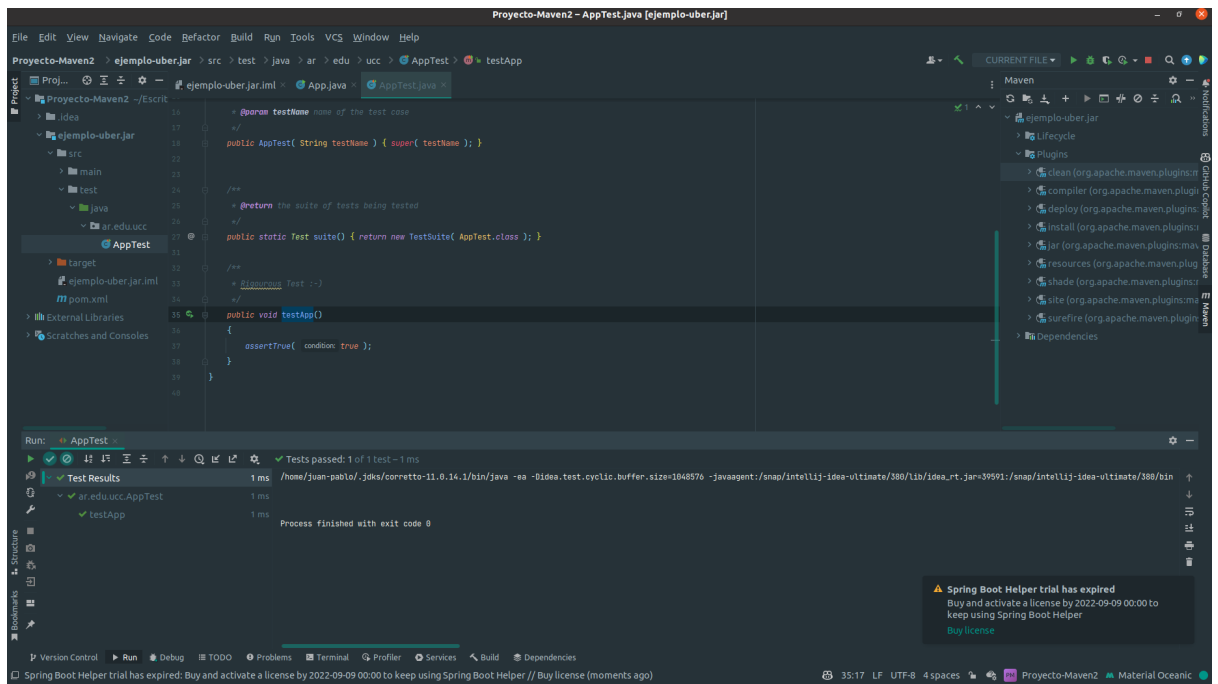


- **Depurar la aplicación**

Podemos ver como debugueo la aplicacion al recorrer un array y puedo ver las variables y sus modificaciones.

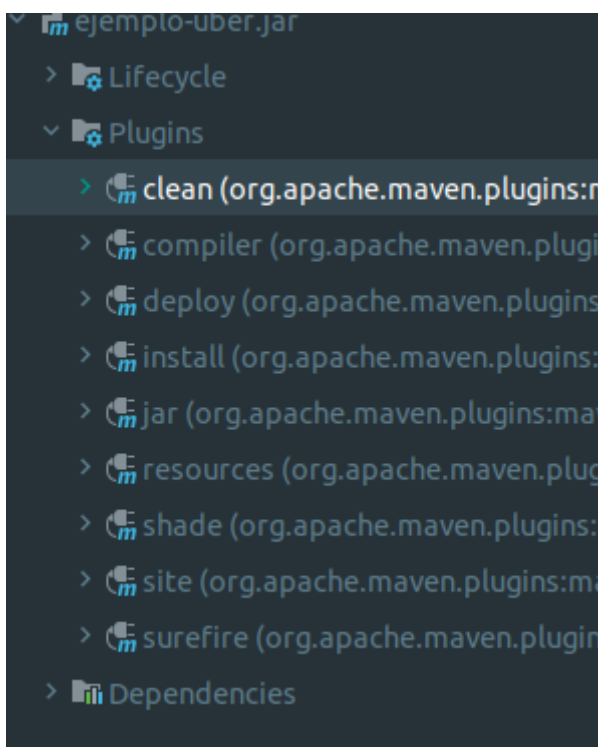


- **Correr unit tests y coverage**



- Ejecutar los goals de maven

Un Maven plugin se encarga de centralizar una serie de tareas que son comunes a la construcción del software y están fuertemente relacionadas . Así pues cada plugin dispone de varias tareas a ejecutar. Por ejemplo el plugin de Compile dispone de dos tareas a cada tarea se la denomina Maven Goal. En este caso la tarea de Compile que compila el código y la tarea o goal de TestCompile que compila la parte de Testting.



- Encontrar donde se puede cambiar la configuración de Maven.

- etc.

8- Ejemplo con nodejs

Instalar el componente para generar aplicaciones Express

`npx create-react-app my-app`

Crear una nueva aplicación

`cd my-app`

`npm start`

Inside that directory, you can run several commands:

`npm start`
Starts the development server.

`npm run build`
Bundles the app into static files for production.

`npm test`
Starts the test runner.

`npm eject`
Removes this tool and copies build dependencies, configuration files and scripts into the app directory. If you do this, you can't go back!

We suggest that you begin by typing:

`cd my-app`
`npm start`

Happy hacking!

`juan-pablo@juanpablo-OMEN-Laptop-15-ek0xxx:~$ cd my-app`
`juan-pablo@juanpablo-OMEN-Laptop-15-ek0xxx:~/my-app$`

La aplicación web estará disponible en <http://localhost:3000>

Inside that directory, you can run several commands:

```
npm start
  Starts the development server.
```

```
npm run build
  Bundles the app into static files for production.
```

```
npm test
  Starts the test runner.
```

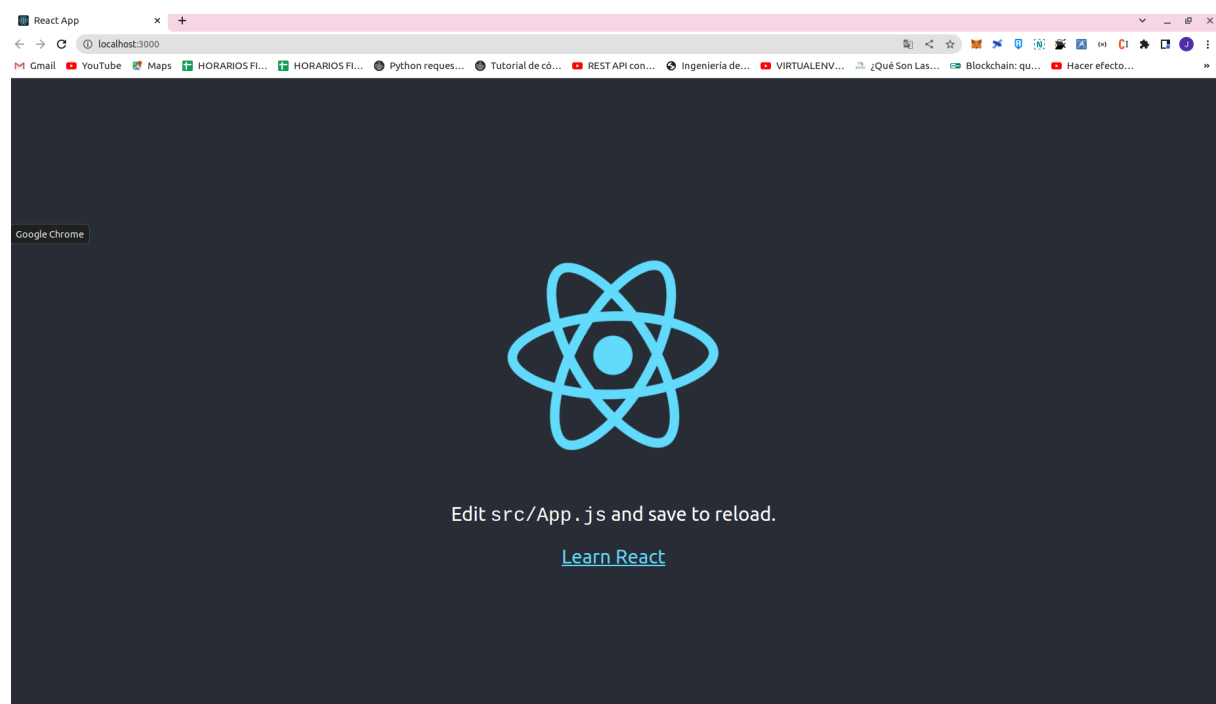
```
npm eject
  Removes this tool and copies build dependencies, configuration files
  and scripts into the app directory. If you do this, you can't go back!
```

We suggest that you begin by typing:

```
cd my-app
npm start
```

Happy hacking!

```
juan-pablo@juanpablo-OMEN-Laptop-15-ek0xxx:~$ cd my-app
juan-pablo@juanpablo-OMEN-Laptop-15-ek0xxx:~/my-app$
```



- **Analizar el manejo de paquetes y dependencias realizado por npm.**

NPM es un manejador de paquetes de node, tiene dos partes principales,

- Un repo online para publicar paquetes de software libre para ser utilizados en proyectos node.js.
- Una herramienta para la terminal para interactuar con el repositorio que permita la instalación de utilidades, manejo de dependencias y la publicación de paquetes

En cada proyecto que utilizamos npm, vamos a tener un archivo package.json que contendrá la información del proyecto, este se genera cuando se empieza a tratar con npm.

entonces tenemos Repo central → autores de paquetes y consumidores de paquetes, en el medio de esta esta el npm CLI o consola encargada de instalar, publicar y manejar dependencias, etc.

en nuestro caso en el proyecto que se utilizo NPM tenemos lo siguiente:


```

{
  "name": "my-app",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@testing-library/jest-dom": "^5.16.5",
    "@testing-library/react": "^13.4.0",
    "@testing-library/user-event": "^13.5.0",
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-scripts": "5.0.1",
    "web-vitals": "^2.1.4"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "eslintConfig": {
    "extends": [
      "react-app",
      "react-app/jest"
    ]
  },
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
      "last 1 chrome version",
      "last 1 firefox version",
      "last 1 safari version"
    ]
  }
}

```

9- Ejemplo con python

Instalar dependencias (Ejemplo Ubuntu) varía según el OS:

```
sudo apt install build-essential python3-dev
pip3 install cookiecutter
```

Correr el scaffold

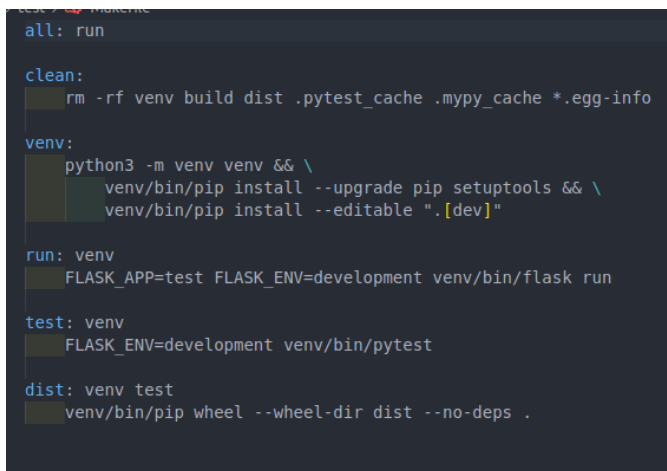
```
$ cookiecutter https://github.com/candidtim/cookiecutter-flask-minimal.git
application_name [Your Application]: test
package_name [yourapplication]: test
$
```

Ejecutar la aplicación

```
cd test
make run
```

- **Acceder a la aplicación en:** <http://localhost:5000/>
- **Explicar que hace una tool como cookiecutter, make y pip.**
- **cookiecutter:** Este nos proporciona una interfaz grafica de usuario para descubrir plantillas, opciones de plantilla de entrada y crear proyectos y archivos.
- **Make:** Make sirve para construir programas, este nos muestra que partes del programa necesitan ser compiladas para levantarse, es decir contiene las ordenes que debe ejecutar la utilidad make, como las dependencias entre los distintos modulos.

en nuestro proyecto de test podemos ver el contenido del mismo:



```
all: run

clean:
    rm -rf venv build dist .pytest_cache .mypy_cache *.egg-info

venv:
    python3 -m venv venv && \
        venv/bin/pip install --upgrade pip setuptools && \
        venv/bin/pip install --editable ".[dev]"

run: venv
    FLASK_APP=test FLASK_ENV=development venv/bin/flask run

test: venv
    FLASK_ENV=development venv/bin/pytest

dist: venv test
    venv/bin/pip wheel --wheel-dir dist --no-deps .
```

- **pip:** es una herramienta de linea de comando que permite administrar paquetes, instalar, reinstalar y desinstalar paquetes de python.

10- Build tools para otros lenguajes

- Hacer una lista de herramientas de build (una o varias) para distintos lenguajes, por ejemplo (Rust -> cargo)
- Elegir al menos 10 lenguajes de la lista de top 20 o top 50 de tiobe:
<https://www.tiobe.com/tiobe-index/>

C# → Essential MSbuild

C++ → Make, Cmake

JavaScript → WebPack

Go → go build, Task

Php → Phing

java → Ant, Maven, Gradle

Perl → Perl Build System

Swift → Swift Package Manager

C → Cheesmake

.Net Framework → NAnt

Groovy → Gradle