UNIDAD 4.

Programación.

Introducción a Maven.

1. Introducción.

El desarrollo de software no solo se centra en la escritura de código, incluyen otras tareas que son necesarias, como es el proceso de realización de test de forma automática, la compilación, el empaquetado, la gestión de dependencias o en ocasiones el desplegado de la aplicación (por ejemplo web, para llevarlo al servidor), entre muchas otras

En prácticamente todos los lenguajes mayoritarios existen herramientas para liberar al desarrollador de las tareas anteriores, o al menos mitigar el tiempo, los recursos y los posibles problemas ocasionados, por ejemplo la compilación de programas con múltiples librerías externas y muchos ficheros fuentes, siendo su funcionalidad muy similar entre estos:

- Definición de proyecto en fichero con formato estructurado: XML y/o JSON.
- Gestión de repositorios externos para librerías de terceros.
- Gestión con línea de comandos.
- Posibilidad de integrar otras herramientas o fases como ejecución de pruebas o desplegado de aplicaciones.

Para Java se tienen 3 herramientas principales:

- **Ant**: Heredera de makefile, para C. Su uso principal era y es la compiliación y construcción. Se basa en la definición de ficheros XML, por defecto build.xml. Posee un comando ant al que se le indica el objetivo a ejecutar, estando dicho objetivo definido en el fichero build.xml. Un ejemplo de build.xml:

```
<project>
    <target name="clean">
        <delete dir="build"/>
    </target>
    <target name="compile">
        <mkdir dir="build/classes"/>
        <javac srcdir="src" destdir="build/classes"/>
    </target>
    <target name="jar">
        <mkdir dir="build/jar"/>
        <jar destfile="build/jar/HelloWorld.jar" basedir="build/classes">
            <manifest>
                <attribute name="Main-Class" value="oata.HelloWorld"/>
            </manifest>
        </jar>
    </target>
    <target name="run">
```

```
        <java jar="build/jar/HelloWorld.jar" fork="true"/>
    </target>
</project>
```

Y un ejemplo de ejecución:

```
ant compile
ant jar
ant run
```

Ant ha caido en desuso, aunque se puede encontrar en proyectos antiguos.

- **Gradle**: El más reciente, evolución de Ant y Maven, define su propio lenguaje para definir cómo construir el proyecto. Diseñado para grandes proyectos, en especial el desarrollo y despliegue. Usado por AndroidStudio, inicialmente centrado en Java y lenguajes relacionados como Google o Scala aunque se ha ampliado a otros como C++ o Swift.

Se tienen comando para interactuar, por ejemplom **gradle init** crea un proyecto a partir de diferentes pregruntas que se realiza como el tipo de proyecto, el lenguaje o el lenguaje SDL de los script a ejecutar (Groovy o Kotlin). Otro ejemplos

  - **gradle run.**
  - **gradle build**
  - **gradle test.**

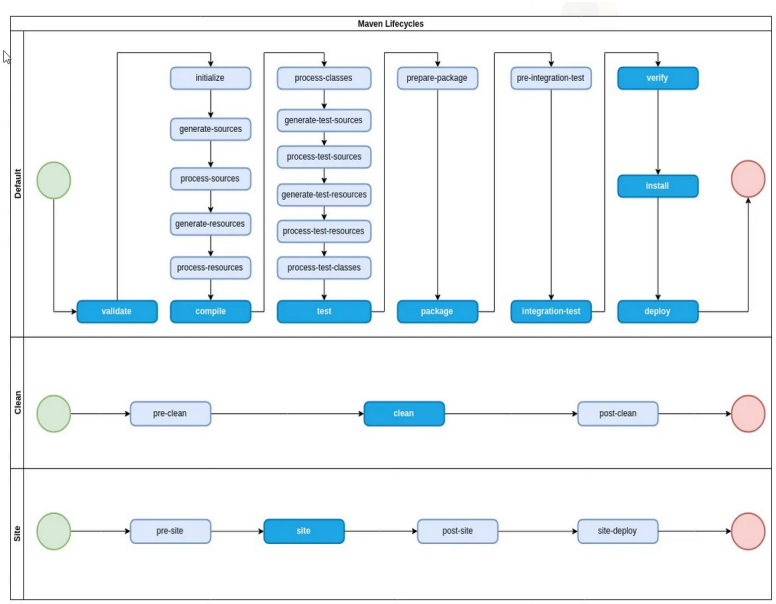
Maven: Junto con Gradle, la herramienta más usada en el mundo Java para la gestión y construcción de proyectos, permite:

- Gestionar librerías de terceros utilizando repositorios.
- Definición y uso de plantillas de proyectos.
- Configuración con fichero xml.
- Integración de “plugins” que amplia la funcionalidad, por ejemplo creación de ejecutables.

Se basa en la ejecución de comandos en el ciclo de vida del desarrollo del sofware que posee diferentes fases entre las que destacan:

- validate. Valida el proyecto.
- initialize. Configura propiedades y crea directorios.
- compile. Compila el código fuente del proyecto.
- test. Ejecuta las pruebas.

- package. Genera el artefacto del proyecto.
- verify. Verifica el artefacto generado.
- install. Instala el artefacto en el repositorio local.
- deploy. Sube el artefacto a un repositorio Maven en la red.

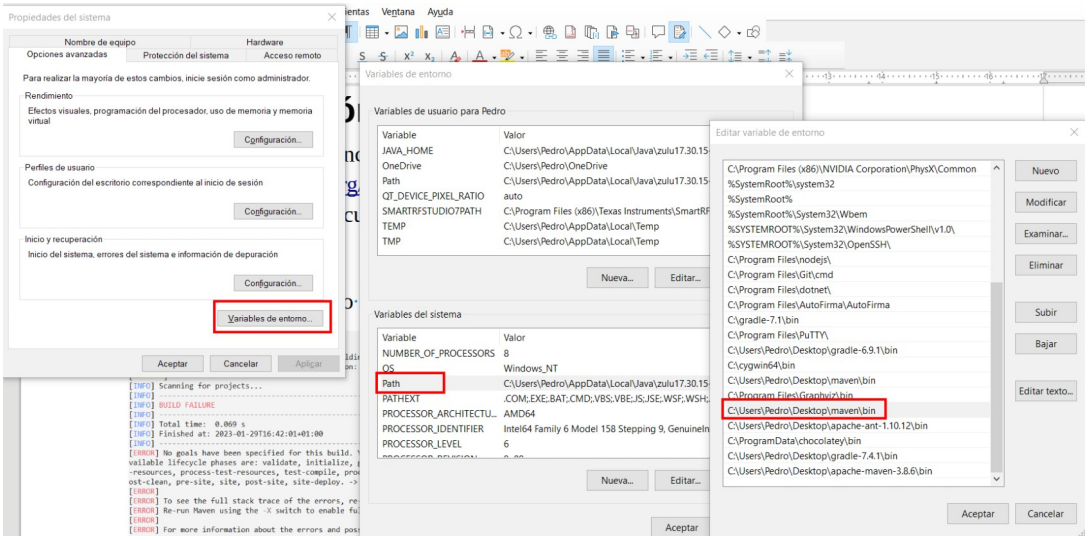


Se puede ejecutar cada una de las fases desde la línea de comandos con

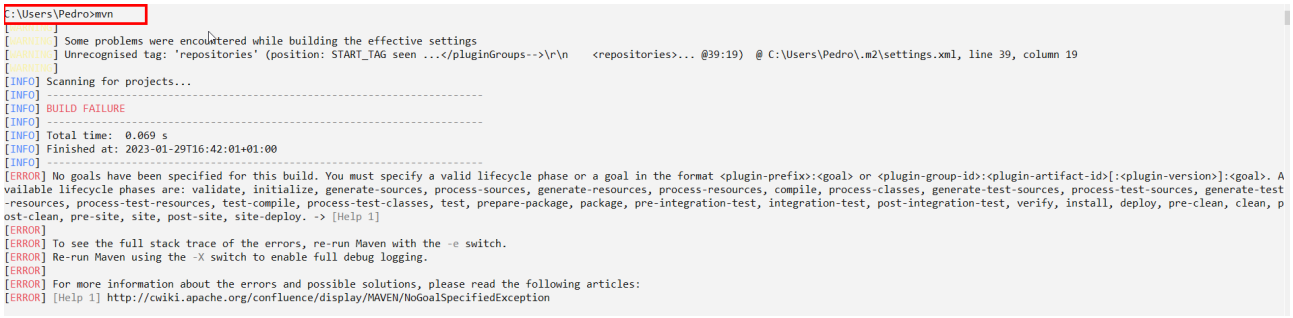
```
mvn fase
```

2. Instalación.

Instalar Maven es sencillo, solo se ha de descargar los binarios desde la página oficial: https://maven.apache.org/download.cgi, descomprimirlo en el directorio seleccionado y añadir el directorio bin que se encuentra en la carpeta descomprimidad a la variable de entorno PATH.



A partir de ese momento es posible ejecutar desde la línea de comandos el comando mvn.



3. Fichero POM.xml.

El fichero pom.xml contiene la información necesaria para la gestión y construcción de la aplicación, se encuentra en la raíz del proyecto. Escrito en XML, contiene diferentes secciones destacando:

- Raíz.
  - Información del proyecto. Con etiquetas como versión, identificador de grupo y artefacto, versión, tipo de empaquetador entre otros.
  - Dependencias.
    - Diferentes librerías necesarias, para cada una de ellas se especifica el grupo, el artefacto, la versión y el ámbito.
  - Plugins.
    - Amplian la funcionalidad

Un ejemplo de POM.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.mycompany</groupId>
    <artifactId>InmoDaw</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>war</packaging>
    <name>InmoDaw-1.0-SNAPSHOT</name>

   <properties>
        <maven.compiler.release>11</maven.compiler.release>
        <endorsed.dir>${project.build.directory}/endorsed</endorsed.dir>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
```

```
        <failOnMissingWebXml>false</failOnMissingWebXml>
        <jakartaee>9.1.0</jakartaee>
    </properties>

    <dependencies>
        <dependency>
            <groupId>jakarta.platform</groupId>
            <artifactId>jakarta.jakartaee-api</artifactId>
            <version>${jakartaee}</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.primefaces</groupId>
            <artifactId>primefaces</artifactId>
            <version>12.0.0</version>
            <classifier>jakarta</classifier>
        </dependency>
        <dependency>
            <groupId>org.webjars</groupId>
            <artifactId>font-awesome</artifactId>
            <version>6.2.0</version>
        </dependency>
        <dependency>
            <groupId>org.glassfish.metro</groupId>
            <artifactId>webservices-rt</artifactId>
            <version>2.3</version>
        </dependency>
        <dependency>
            <groupId>jakarta.ws.rs</groupId>
            <artifactId>jakarta.ws.rs-api</artifactId>
            <version>3.1.0</version>
        </dependency>
        <dependency>
            <groupId>org.eclipse.persistence</groupId>
            <artifactId>org.eclipse.persistence.core</artifactId>
            <version>2.7.10</version>
            <scope>provided</scope>
        </dependency>
```

```
        <dependency>
            <groupId>org.eclipse.persistence</groupId>
            <artifactId>org.eclipse.persistence.asm</artifactId>
            <version>9.2.0</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.eclipse.persistence</groupId>
            <artifactId>org.eclipse.persistence.antlr</artifactId>
            <version>2.7.10</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.eclipse.persistence</groupId>
            <artifactId>org.eclipse.persistence.jpa</artifactId>
            <version>2.7.10</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.eclipse.persistence</groupId>
            <artifactId>org.eclipse.persistence.jpa.jpql</artifactId>
            <version>2.7.10</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.eclipse.persistence</groupId>
            <artifactId>org.eclipse.persistence.moxy</artifactId>
            <version>2.7.10</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.eclipse.persistence</groupId>
            <artifactId>jakarta.persistence</artifactId>
            <version>2.2.3</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.apache.derby</groupId>
```

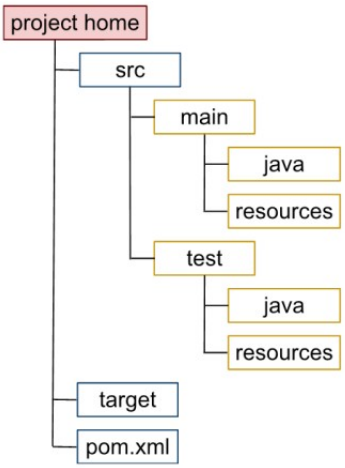
```
            <artifactId>derbyclient</artifactId>
            <version>10.14.2.0</version>
        </dependency>
        <dependency>
            <groupId>org.eclipse.persistence</groupId>
<artifactId>org.eclipse.persistence.jpa.modelgen.processor</artifactId>
            <version>2.7.10</version>
            <scope>provided</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.10.1</version>
                <configuration>
                    <compilerArgs>
                        <args>-Djava.endorsed.dirs=${endorsed.dir}</args>
                    </compilerArgs>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-war-plugin</artifactId>
                <version>3.3.2</version>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-dependency-plugin</artifactId>
                <version>3.3.0</version>
                <executions>
                    <execution>
                        <phase>validate</phase>
                        <goals>
                            <goal>copy</goal>
```

```
                            </goals>
                            <configuration>
                                <outputDirectory>${endorsed.dir}</outputDirectory>
                                <silent>true</silent>
                                <artifactItems>
                                    <artifactItem>
                                        <groupId>jakarta.platform</groupId>
<artifactId>jakarta.jakartaee-api</artifactId>
                                        <version>${jakartaee}</version>
                                        <type>jar</type>
                                    </artifactItem>
                                </artifactItems>
                            </configuration>
                        </execution>
                    </executions>
                </plugin>
            </plugins>
        </build>
</project>
```

4. Estructura de un proyecto.

Maven posee por defecto una estructura con los diferentes elementos del proyecto, en la raíz se encuentran:

- pom.xml. Fichero con la configuración de Maven.
- Src. Directorio que contiene el código fuente, tanto el de la aplicación como el de los test.Se tiene a su vez la carpeta main dividida en 2:
  - Carpeta main, con los ficheros del proyecto, dividio a su vez en:
    - Java: Contiene el código propiamente dicho, los paquetes con los ficheros .java.
    - Resources: Otros elementos necearios para el proyecto como son imágenes, fuentes, css...
  - Carpeta test: Similar a la anterior, pero con los casos de prueba.
- Target. Carpeta en el que se encuentran los archivos generados en las fases del proyecto, como son las clases compiladas o el artefacto fina, normalmente un jar.



5. Dependencias.

Una de las ventajas de usar Maven es la de no necesitar descargar las librerías externas necesarias para el proyecto, encargándose de forma automática de la descarga, actualización del "classpath" y enlazandolo al ejecutable de ser necesario de forma automática.

En el fichero POM.xml se tiene la etiqueta dependencies, y dentro de esta, etiquetas dependencie que definen una dependencia concreta usando a su vez las etiquetas:

- groupId: Clasificación, un grupo puede tener muchos artefactos (librerías)
- artifactId: Dentro de un grupo un artefacto (librería) concreto.
- Version: Versión del artefacto.
- Scope: Ámbito en el que el artefacto está disponible en el ciclo de vida.
- Classifier: Clasificador del artefacto además de la versión
- type: Por defecto jar, si es de otro tipo: war, ear… se ha de especificar.

Un ejemplo sencillo de declaración de dependencias para Maven:

```
<dependencies>
  <dependency>
    <groupId>org.openjfx</groupId>
    <artifactId>javafx-graphics</artifactId>
    <version>16</version>
    <type>jar</type>
  </dependency>
  <dependency>
    <groupId>com.jfoenix</groupId>
    <artifactId>jfoenix</artifactId>
    <version>9.0.10</version>
```

```
  </dependency>
  <dependency>
    <groupId>org.openjfx</groupId>
    <artifactId>javafx-controls</artifactId>
    <version>17.0.0.1</version>
    <type>jar</type>
  </dependency>
  <dependency>
    <groupId>jakarta.xml.bind</groupId>
    <artifactId>jakarta.xml.bind-api</artifactId>
    <version>3.0.1</version>
    <type>jar</type>
  </dependency>
  <dependency>
    <groupId>com.sun.xml.bind</groupId>
    <artifactId>jaxb-impl</artifactId>
    <version>3.0.2</version>
  </dependency>
  <dependency>
    <groupId>com.google.code.gson</groupId>
    <artifactId>gson</artifactId>
    <version>2.9.0</version>
    <type>jar</type>
  </dependency>
</dependencies>
```

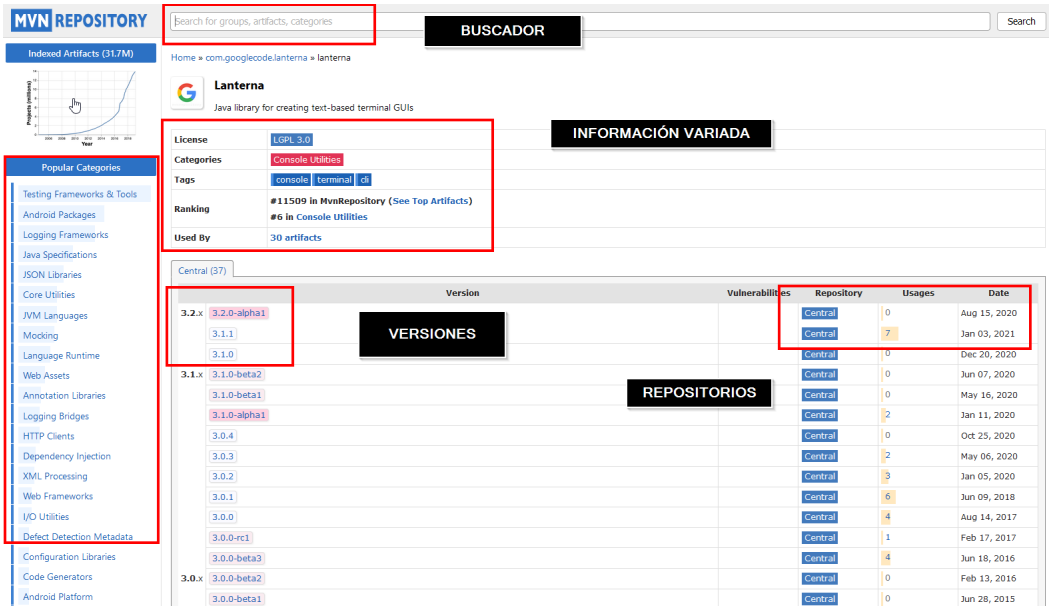
El ámbito o “scope” determina cuando la dependencia es incluida en el “classpath”, pudiendo tener los valores:

- Compile: Este es el ámbito predeterminado, que se utiliza si no se especifica ninguno. Las dependencias de compilación están disponibles en todos los classpaths de un proyecto. Además, esas dependencias se propagan a proyectos dependientes.
- Provided: Muy parecido a compilar, pero indica que espera que el JDK o un contenedor proporcione la dependencia en tiempo de ejecución. Por ejemplo, al crear una aplicación web para Java Enterprise Edition, establecería la dependencia de la API Servlet y las API Java EE relacionadas con el alcance proporcionado porque el contenedor web proporciona esas clases. Se agrega una dependencia con este ámbito a la ruta de clase utilizada para la compilación y la prueba, pero no a la ruta de clase en tiempo de ejecución. No es transitivo.

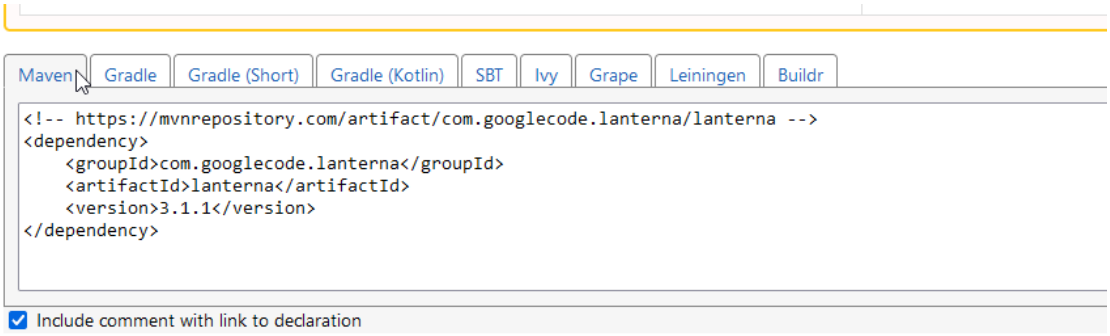
- Runtime: Este ámbito indica que la dependencia no es necesaria para la compilación, pero sí para la ejecución. Maven incluye una dependencia con este alcance en el tiempo de ejecución y en las rutas de clase de prueba, pero no en la ruta de clase de compilación.
- Test: Este alcance indica que la dependencia no es necesaria para el uso normal de la aplicación y solo está disponible para las fases de compilación y ejecución de prueba. Este alcance no es transitivo. Normalmente, este alcance se usa para bibliotecas de prueba como JUnit y Mockito. También se usa para bibliotecas que no son de prueba, como Apache Commons IO, si esas bibliotecas se usan en pruebas unitarias (src/test/java) pero no en el código del modelo (src/main/java).
- System: Este alcance es similar al providedo, excepto que debe proporcionar el JAR que lo contiene explícitamente. El artefacto siempre está disponible y no se busca en un repositorio.
- Import: Este ámbito solo se admite en una dependencia de tipo pom en la sección <dependencyManagement>. Indica que la dependencia se reemplazará con la lista efectiva de dependencias en la sección <dependencyManagement> del POM especificado. Dado que se reemplazan, las dependencias con un alcance de importación en realidad no participan en la limitación de la transitividad de una dependencia.

	compile	provided	runtime	test
compile	compile(*)	-	runtime	-
provided	provided	-	provided	-
runtime	runtime	-	runtime	-
test	test	-	test	-

Para obtener los artefactos o librerías se recure al uso repositorios en los que se encuentran estos con sus diferentes versiones. Es posible definir repositorios externos, aunque por defecto y más conocido es https://mvnrepository.com/, que posee un buscador.



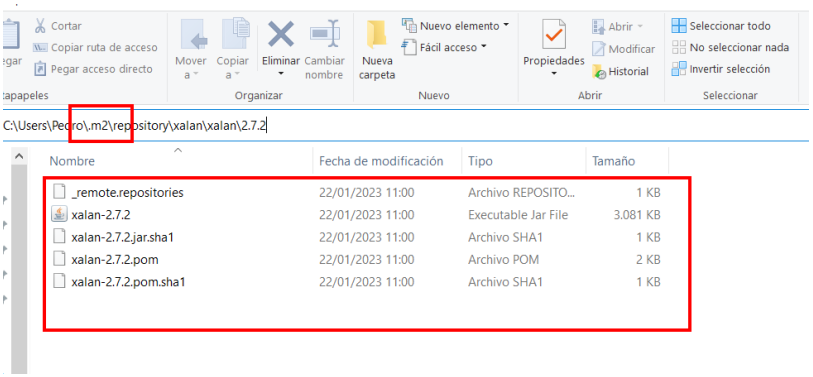
Una vez localizado el artefacto y la versión deseada, pulsar sobre el, apareciendo una página con el xml a insertar en el fichero. Observar que se tiene ofrecen opciones para otros sistemas como Gradle:



Ahora solo es necesario incluir el código dentro de <depencies>.

Para optimizar el uso de los repositorios, Maven crea un repositorio en local, en el que se almacen los artefactos (librerías) usadan en alguno de los proyectos, de forma que se puedan reutilizar en posteriores proyectos.

El repositorio se define para cada usuario, y se encuentra en el directorio oculot .m2:



*** Si NetBeans tarda mucho y se ralentiza, es posible que tenga problemas al indexar este directorio, borrarlo para solucionar el problema.**

6. Plugins.

Se puede ampliar la funcionalidad de Maven para conseguir objetivos específicios, por ejemplo la generación de programas con JavaFX para diseño de interfaces gráficas. Existen 2 tipos de "plugins":

Compilación (build): Se ejecutarán durante la compilación y deben configurarse en el **<build/>** del POM.

Informes (reporting): Se ejecutarán durante la generación del sitio y deben configurarse en el **<reporting/>** del POM.

La lista de plugins oficial se puede consultar en https://maven.apache.org/plugins/, entre los soportados por de forma oficial se encuentra:

NOMBRE	TIPO	DESCRIPCIÓN
Clean	Construcción	Limpia después de la construcción

Compiler	Construcción	Compila las fuentes.
Deploy	Construcción	Implementa el artefacto en un repositorio remoto.
Install	Construcción	Instala en el repositorio local
Resources	Construcción	Copia los recursos en el directorio de salida para incluirlos en el JAR (imágenes, sonidos, ficheros...)
Verifier	Construcción	Pruebas de integración.
ChangeLog	Informe	Genera un informe de los cambios.
Javadoc	Informe	Genera un Javadoc del proyecto.
Archetype	Construcción	Genera la estructura de un proyecto a partir de un arquetipo (plantilla).
Assembly	Construcción	Construye una distribución de fuentes y/o binarios.
Pdf	Construcción	Genera una versión en PDF de la documentación del proyecto.

En https://www.mojohaus.org/plugins.html se dispone de otros muchos plugins, pudiendo incluso crear cada desarrollador los suyos propios en función de las necesidades.

Los plugings de construcción (build) se incluyen dentro de la etiqueta **build** del fichero pom.xml y los de informes (**reporting)** en la etiqueta **reporting**, definiendose cada uno de ellos a su vez dentro de la etiqueta **plugins.**

Cada plugin tiene su propia configuración y es necesario consultar la documentación para conocer las diferentes opciones que ofrece. Por ejemplo en plugin maven-compiler-plugin se encuentra documentado en: https://maven.apache.org/plugins/maven-compiler-plugin/index.html. Permite definir el proceso de compilación (principalmente opciones de javac). Un ejemplo de uso en la que se configura las opciones del compilador:

```
<plugins>
  <plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-compiler-plugin</artifactId>
    <version>3.10.1</version>
    <configuration>
      <compilerArgs>
        <arg>-verbose</arg>
        <arg>-Xlint:all,-options,-path</arg>
```

```
      </compilerArgs>
    </configuration>
  </plugin>
</plugins>
```

Un plugin interesante, que se usará en el curso es open.fx-maven-plugin, que facilita el uso y construcción de aplicaciones para escritorio usando JavaFX, la documentación se encuentra en GitHub, https://github.com/openjfx/javafx-maven-plugin:

```
<plugin>
    <groupId>org.openjfx</groupId>
    <artifactId>javafx-maven-plugin</artifactId>
    <version>0.0.8</version>
    <configuration>
        <mainClass>hellofx/org.openjfx.App</mainClass>
    </configuration>
</plugin>
```

Los “plugins” añaden “Goals” o tareas, por ejemplo el plugin de compilación tiene dos “goals” el de compilación y el de compilación de los test, cada uno asociaciado a elementos del ciclo de vida diferente: fase compilación y fase de testeo.

Para ver los “goals” o tareas de un “plugin” utilizar el comando:

```
mvn help:describe -DgroupId=org.apache.maven.plugins  -DartifactId=maven-compiler-plugin
```

Si se añade el parámetro -Ddetails aparece toda la información del “plugins”. En el caso de JavaFX:

```
mvn help:describe -DgroupId=org.openjfx  -DartifactId=javafx-maven-plugin -Ddetails
```

Obteniendo la salida:

```
Name: javafx-maven-plugin Maven Mojo
Description: The JavaFX Plugin is used to run JavaFX 11+ projects
Group Id: org.openjfx
Artifact Id: javafx-maven-plugin
Version: 0.0.8
Goal Prefix: javafx

This plugin has 2 goals:

javafx:jlink
  Description: (no description available)
```

```
Implementation: org.openjfx.JavaFXJLinkMojo
Language: java
Before this goal executes, it will call:
  Phase: 'process-classes'
Available parameters:
  async (Default: false)
    User property: javafx.async
    If set to true the child process executes asynchronously and build
    execution continues in parallel.

  asyncDestroyOnShutdown (Default: true)
    User property: javafx.asyncDestroyOnShutdown
    If set to true, the asynchronous child process is destroyed upon JVM
    shutdown. If set to false, asynchronous child process continues execution
    after JVM shutdown. Applies only to asynchronous processes; ignored for
    synchronous processes.

  bindServices (Default: false)
    User property: javafx.bindServices
    Add the option --bind-services or not, default false.

  commandlineArgs
    User property: javafx.args
    Arguments separated by space for the executed program. For example: '-j
    20'

  compress (Default: 0)
    User property: javafx.compress
    Compression level of the resources being used, equivalent to: -c,
    --compress=level. Valid values: 0, 1, 2, default 0

  ignoreSigningInformation (Default: false)
    User property: javafx.ignoreSigningInformation
    --ignore-signing-information, default false

  includePathExceptionsInClasspath (Default: false)
    User property: javafx.includePathExceptionsInClasspath
    If set to true, it will include the dependencies that generate path
```

```
    exceptions in the classpath. Default is false.

  jlinkExecutable (Default: jlink)
    User property: javafx.jlinkExecutable
    The executable. Can be a full path or the name of the executable. In the
    latter case, the executable must be in the PATH for the execution to
    work.

  jlinkImageName (Default: image)
    User property: javafx.jlinkImageName
    The name of the folder with the resulting runtime image, equivalent to
    --output <path>

  jlinkVerbose (Default: false)
    User property: javafx.jlinkVerbose
    Turn on verbose mode, equivalent to: --verbose, default false

  jlinkZipName
    User property: javafx.jlinkZipName
    When set, creates a zip of the resulting runtime image.

  jmodsPath
    User property: javafx.jmodsPath
    Optional jmodsPath path for local builds.

  launcher
    User property: javafx.launcher
    Add a launcher script, equivalent to: --launcher
    <name>=<module>[/<mainclass>].

  mainClass
    Required: true
    User property: javafx.mainClass
    (no description available)

  noHeaderFiles (Default: false)
    User property: javafx.noHeaderFiles
    Remove the includes directory in the resulting runtime image, equivalent
```

```
      to: --no-header-files, default false

    noManPages (Default: false)
      User property: javafx.noManPages
      Remove the man directory in the resulting Java runtime image, equivalent
      to: --no-man-pages, default false

    options
      A list of vm options passed to the executable.

    outputFile
      User property: javafx.outputFile
      (no description available)

    runtimePathOption
      User property: javafx.runtimePathOption
      Type of RuntimePathOption to run the application.

    skip (Default: false)
      User property: javafx.skip
      Skip the execution.

    stripDebug (Default: false)
      User property: javafx.stripDebug
      Strips debug information out, equivalent to -G, --strip-debug, default
      false

    stripJavaDebugAttributes (Default: false)
      User property: javafx.stripJavaDebugAttributes
      Strip Java debug attributes out, equivalent to
      --strip-java-debug-attributes, default false

    workingDirectory
      User property: javafx.workingDirectory
      The current working directory. Optional. If not specified, basedir will
      be used.

javafx:run
```

```
Description: (no description available)
Implementation: org.openjfx.JavaFXRunMojo
Language: java
Before this goal executes, it will call:
  Phase: 'process-classes'

Available parameters:
  async (Default: false)
    User property: javafx.async
    If set to true the child process executes asynchronously and build
    execution continues in parallel.

  asyncDestroyOnShutdown (Default: true)
    User property: javafx.asyncDestroyOnShutdown
    If set to true, the asynchronous child process is destroyed upon JVM
    shutdown. If set to false, asynchronous child process continues execution
    after JVM shutdown. Applies only to asynchronous processes; ignored for
    synchronous processes.

  commandlineArgs
    User property: javafx.args
    Arguments separated by space for the executed program. For example: '-j
    20'

  executable (Default: java)
    User property: javafx.executable
    The executable. Can be a full path or the name of the executable. In the
    latter case, the executable must be in the PATH for the execution to
    work.

  includePathExceptionsInClasspath (Default: false)
    User property: javafx.includePathExceptionsInClasspath
    If set to true, it will include the dependencies that generate path
    exceptions in the classpath. Default is false.

  mainClass
    Required: true
    User property: javafx.mainClass
```

```
      (no description available)

    options
      A list of vm options passed to the executable.

    outputFile
      User property: javafx.outputFile
      (no description available)

    runtimePathOption
      User property: javafx.runtimePathOption
      Type of RuntimePathOption to run the application.

    skip (Default: false)
      User property: javafx.skip
      Skip the execution.

    workingDirectory
      User property: javafx.workingDirectory
      The current working directory. Optional. If not specified, basedir will
      be used.
```

Para ejecutar una tarea desde la línea de comandos:

```
mvn goal parámetros
```

Siguiendo con JavaFX, sobre un proyecto, para iniciar la aplicación:

```
mvn javafx:run
```

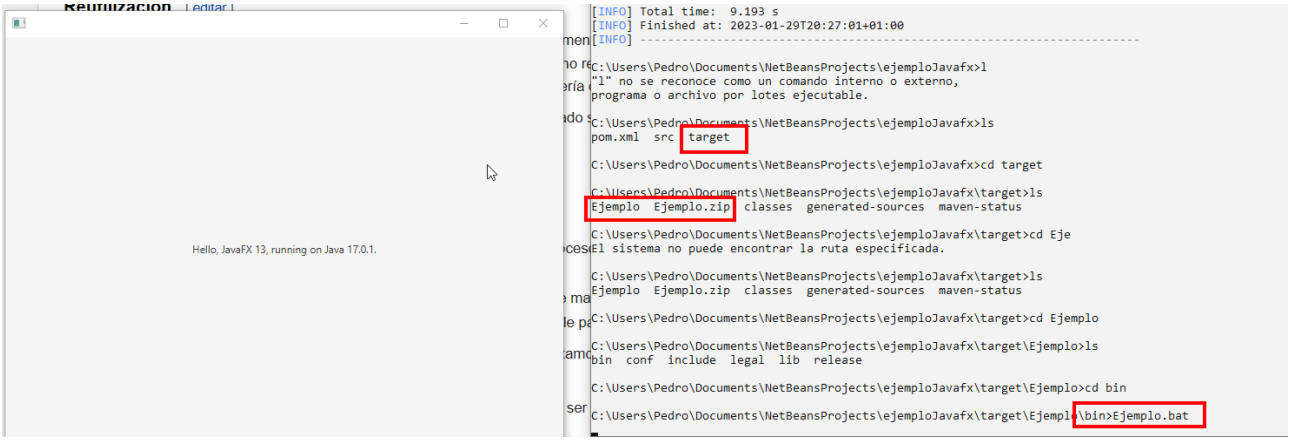
Para crear una aplicación portable (no necesita tener el JRE instalado en el cliente) se utiliza la tarea Jlink, junto con la configuración del “plugin”:

```
<configuration>
  <mainClass>com.mycompany.ejemplojavafx.App</mainClass>
  <launcher>Ejemplo</launcher>
  <jlinkImageName>Ejemplo</jlinkImageName>
  <jlinkZipName>Ejemplo</jlinkZipName>
</configuration>
```

Ejecución:

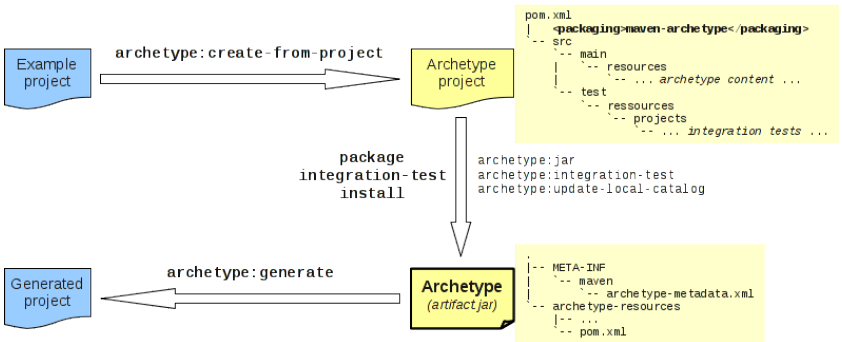
```
mvn javafx:jlink
```

Generando una carpeta y un comprimido con todo lo necesario para ejecutar la aplicación JavaFX



7. Archetypes.

Los “archetypes” o arquetipos en castellano, son plantillas que construyen el esqueleto de una aplicación, generando un pom.xml con la configuración necesaria, la estructura de directorios e incluso parte del código necesario.



El comando para generar un proyecto es sencillo:

```
mvn archetype:generate
```

Indicando además el groupId el id del artefacto y la versión del mismo.

Existe al igual que para las dependencias, existe un repositorio de “plantillas”: https://mvnrepository.com/search?q=archetype.

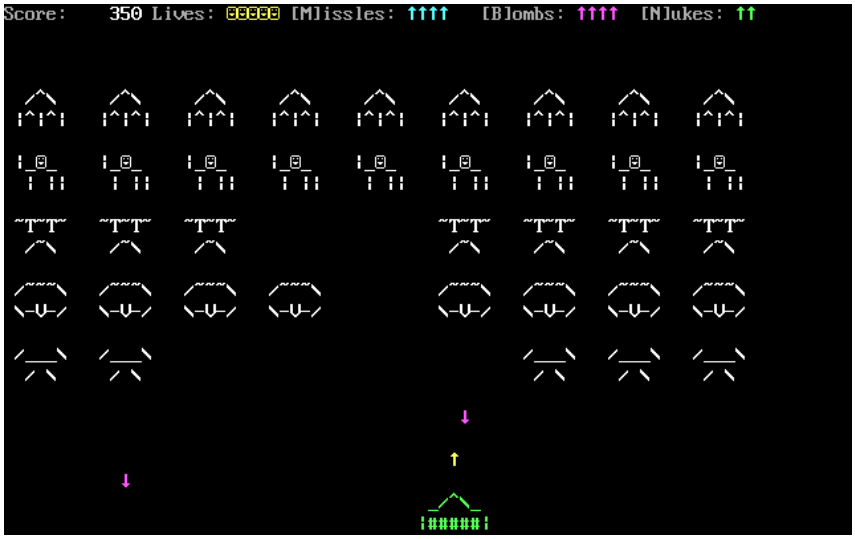
Por ejemplo un arquetipo para una aplicación para IOS: https://mvnrepository.com/artifact/org.robovm/robovm-templates-ios-single-view.

Desde la línea de comandos con archetype:generate, se pueden realizar búsquedas en los repositorios:

```
C:\Users\Pedro\Documents\NetBeansProjects>mvn archetype:generate
[WARNING]
[WARNING] Some problems were encountered while building the effective settings
[WARNING] Unrecognised tag: 'repositories' (position: START_TAG seen ...</pluginGroups-->\r\n    <repositories>... @39:19)  @ C:\Users\Pedro\.m2\settings.xml, lir
[WARNING]
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------< org.apache.maven:standalone-pom >-------------------
[INFO] Building Maven Stub Project (No POM) 1
[INFO] --------------------------------[ pom ]---------------------------------
[INFO]
[INFO] >>> maven-archetype-plugin:3.2.1:generate (default-cli) > generate-sources @ standalone-pom >>>
[INFO]
[INFO] <<< maven-archetype-plugin:3.2.1:generate (default-cli) < generate-sources @ standalone-pom <<<
[INFO]
[INFO]
[INFO] --- maven-archetype-plugin:3.2.1:generate (default-cli) @ standalone-pom ---
[INFO] Generating project in Interactive mode
[INFO] No archetype defined. Using maven-archetype-quickstart (org.apache.maven.archetypes:maven-archetype-quickstart:1.0)
Choose archetype:
1: remote -> am.ik.archetype:elm-spring-boot-blank-archetype (Blank multi project for Spring Boot + Elm)
2: remote -> am.ik.archetype:graalvm-blank-archetype (Blank project for GraalVM)
3: remote -> am.ik.archetype:graalvm-springmvc-blank-archetype (Blank project for GraalVM + Spring MVC)
4: remote -> am.ik.archetype:graalvm-springwebflux-blank-archetype (Blank project for GraalVM + Spring MVC)
5: remote -> am.ik.archetype:maven-reactjs-blank-archetype (Blank Project for React.js)
6: remote -> am.ik.archetype:msgpack-rpc-jersey-blank-archetype (Blank Project for Spring Boot + Jersey)
7: remote -> am.ik.archetype:mvc-1.0-blank-archetype (MVC 1.0 Blank Project)
8: remote -> am.ik.archetype:spring-boot-blank-archetype (Blank Project for Spring Boot)
9: remote -> am.ik.archetype:spring-boot-docker-blank-archetype (Docker Blank Project for Spring Boot)
10: remote -> am.ik.archetype:spring-boot-gae-blank-archetype (GAE Blank Project for Spring Boot)
11: remote -> am.ik.archetype:spring-boot-jersey-blank-archetype (Blank Project for Spring Boot + Jersey)
12: remote -> am.ik.archetype:spring-fu-jafu-blank-archetype (Blank project for Vanilla Spring WebFlux.fn)
13: remote -> am.ik.archetype:vanilla-spring-webflux-fn-blank-archetype (Blank project for Vanilla Spring WebFlux.fn)
14: remote -> at.chrl.archetypes:chrl-spring-sample (Archetype for Spring Vaadin Webapps)
15: remote -> at.stderr:archetype-simple (-)
16: remote -> be.cloudway:gramba-aws-lambda-archetype (-)
17: remote -> biz.turnonline.ecosystem:turnonline-ecosystem-microservice-archetype (TurnOnline.biz Ecosystem: Serverless Microservice Archetype)
18: remote -> br.com.address.archetypes:struts2-archetype (an archetype web 3.0 + struts2 (bootstrap + jquery) + JPA 2.1 with struts2 login system)
19: remote -> br.com.address.archetypes:struts2-base-archetype (An Archetype with JPA 2.1; Struts2 core 2.3.28.1; Jquery struts plugin; Struts BootStrap plugin; J
20: remote -> br.com.anteros:Anteros-Archetype (Anteros Archetype for Java Web projects.)
21: remote -> br.com.codecode:vlocadora-json (Modelos com Anotaþ§es Gson)
22: remote -> br.com.diogoko:maven-doclet-archetype (A Maven archetype to create Doclets for Javadoc)
23: remote -> br.com.ingenieux:elasticbeanstalk-docker-dropwizard-webapp-archetype (A Maven Archetype for Publishing Dropwizard-based Services on AWS' Elastic
       Beanstalk Service)
```

8. Ejemplo.

Se desea crear el juego SpaceInvader en modo consola de texto, para ello se usa la librería Lanterna, similar a la famosa ncurses en C. De forma estándar un terminal en modo texto tiene 80



columas y 24 líneas.

Se utiliza el lenguaje Java y la herramienta Maven. Encargándonos preparar el entorno, con las labores a realizar:

1. Crear el proyecto Maven a partir del arquetipo básico.

2. Añadir la librería Lanterna al proyecto.
3. Crear la clase Game con el método main y un método para las pruebas.
4. Crear casos de prueba para el método anterior y ver que funciona.
5. Ejecutar el programa una vez pasadas las pruebas.

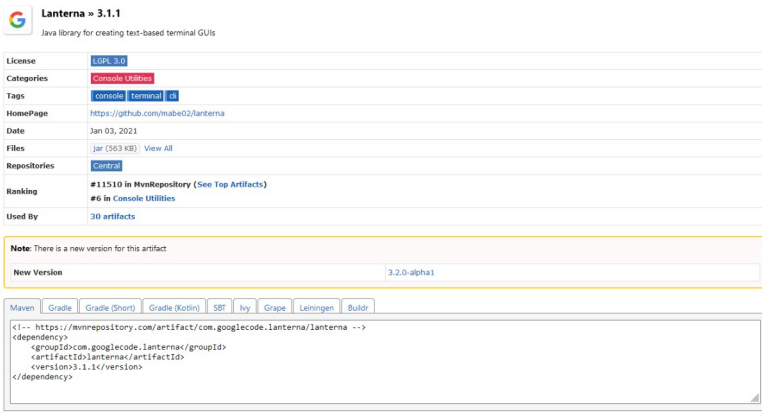
Paso 1.

Crear el proyecto usando el arquetipo maven-archetype-quickstart.

```
mvn archetype:generate -DgroupId=pedro.ieslaencanta.com.space -
DartifactId=spaceinvaders -DarchetypeArtifactId=maven-archetype-quickstart -
DarchetypeVersion=1.4 -DinteractiveMode=false
```

```
C:\Users\Pedro\Documents\NetBeansProjects\spaceinvader>tree
Listado de rutas de carpetas para el volumen Windows
El número de serie del volumen es F0C5-93CD
C:.
└───spaceinvaders
    └───src
        ├───main
        │   └───java
        │       └───pedro
        │           └───ieslaencanta
        │               └───com
        │                   └───space
        └───test
            └───java
                └───pedro
                    └───ieslaencanta
                        └───com
                            └───space
```

Paso 2.



Paso 2.1 Ir a Maven repository y buscar la librería.

Paso 2.2. Añadir la librería a la lista de dependencías.

```
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.11</version>
```

```
      <scope>test</scope>
    </dependency>
    <!--
https://mvnrepository.com/artifact/com.googlecode.lanterna/lanterna -->
    <dependency>
      <groupId>com.googlecode.lanterna</groupId>
      <artifactId>lanterna</artifactId>
    <version>3.1.1</version>
    </dependency>
</dependencies>
```

Paso 3.

Crear la clase Game con el método main y un método para las pruebas.

Si bien el arquetipo crea la clase App, en este caso se crea la clase Game y se borra App, definiendo un método para probar los test.

```
public class Game {
    public static void main(String[] args) {
        System.out.println("Prueba inicial");
    }
    public Game(){
    }
    public boolean pruebaJUnit(int numero1,int numero2){
        return numero1<numero2;
    }

}
```

Paso 4.

Crear casos de prueba para el método anterior y ver que funciona. Los casos de prueba no son más que métodos de una clase que el framework Junit se encarga de ejecutar. Dentro de cada método, denominada prueba se crean objetos, llamando a métodos del mismo y se compara el resultado esperado con el real, en caso de ser diferente, no se ha pasado la prueba.

Para indicar a Junit que el método es una prueba se añade en la cabecera del método la anotacion @Test y @Before para inicializar la clase (en temas posteriores se tratan las anotaciones).

```
public class GameTest
{
    Game game;
    @Before
    public void init(){
```

```
        this.game= new Game();
    }
    @Test
    public void prueba1()
    {
        assertTrue( this.game.pruebaJUnit(5,6) );
    }
     @Test
    public void prueba2()
    {
        assertTrue( this.game.pruebaJUnit(5,4) );
    }
}
```

Para ejecutar los test desde la línea de comandos (la primera vez tarda al descargar los artefactos al repositorio local):

```
mvn test
```

El resultado:

```
[INFO]
[INFO] Results:
[INFO]
[ERROR] Failures:
[ERROR]   GameTest.prueba2:27
[INFO]
[ERROR] Tests run: 2, Failures: 1, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD FAILURE
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  2.048 s
[INFO] Finished at: 2023-01-29T22:04:44+01:00
[INFO] ------------------------------------------------------------------------
[ERROR] Failed to execute goal org.apache.maven.plugins:maven-surefire-plugin:2.22.1:test (default-test)
```

Se ha probocado un fallo para ver que no pasa los test, si se corrige el resultado es:

```
[INFO] --- maven-resources-plugin:3.0.2:resources (default-resources) @ spaceinvaders ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory C:\Users\Pedro\Documents\NetBeansProjects\spaceinvader\spaceinvaders\src\main\resources
[INFO]
[INFO] --- maven-compiler-plugin:3.8.0:compile (default-compile) @ spaceinvaders ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-resources-plugin:3.0.2:testResources (default-testResources) @ spaceinvaders ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory C:\Users\Pedro\Documents\NetBeansProjects\spaceinvader\spaceinvaders\src\test\resources
[INFO]
[INFO] --- maven-compiler-plugin:3.8.0:testCompile (default-testCompile) @ spaceinvaders ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-surefire-plugin:2.22.1:test (default-test) @ spaceinvaders ---
[INFO]
[INFO] -------------------------------------------------------
[INFO]  T E S T S
[INFO] -------------------------------------------------------
[INFO] Running pedro.ieslaencanta.com.space.GameTest
[INFO] Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.044 s - in pedro.ieslaencanta.com.space.GameTest
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 2, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
```

Paso 5. Ejecutar el programa una vez pasadas las pruebas.

Se puede usar maven para ejecutar comandos, en el caso de Java:

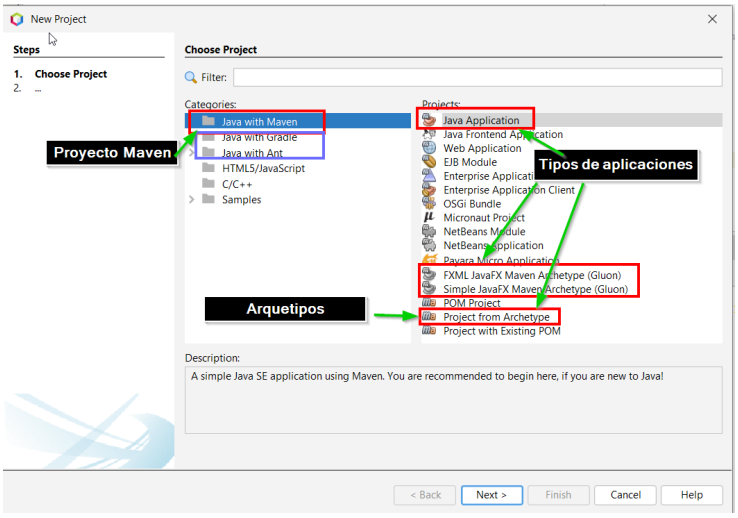
```
C:\Users\Pedro\Documents\NetBeansProjects\spaceinvader\spaceinvaders>mvn exec:java -Dexec.mainClass="pedro.ieslaencanta.com.space.Game"
[WARNING]
[WARNING] Some problems were encountered while building the effective settings
[WARNING] Unrecognised tag: 'repositories' (position: START_TAG seen ...</pluginGroups-->\r\n    <repositories>... @39:19)  @ C:\Users\Pedro\.m2\settings.
xml, line 39, column 19
[WARNING]
[INFO] Scanning for projects...
[INFO]
[INFO] ------------< pedro.ieslaencanta.com.space:spaceinvaders >-------------
[INFO] Building spaceinvaders 1.0-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- exec-maven-plugin:3.1.0:java (default-cli) @ spaceinvaders ---
Prueba inicial
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  0.478 s
[INFO] Finished at: 2023-01-29T22:48:36+01:00
[INFO] ------------------------------------------------------------------------
```

Es posible incluir plugins que incluyan tareas para no necesitar ejecutar el exec.

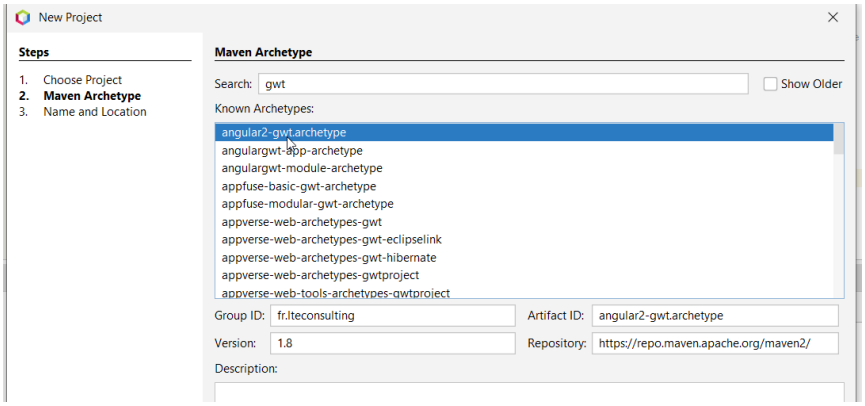
9. Integración con NetBeans.

La mayoría de los entornos de desarrollo para Java integran Maven de forma que se puedan realizar casi todas las acciones de línea de comandos desde el IDE, no siendo NetBeans una excepción.

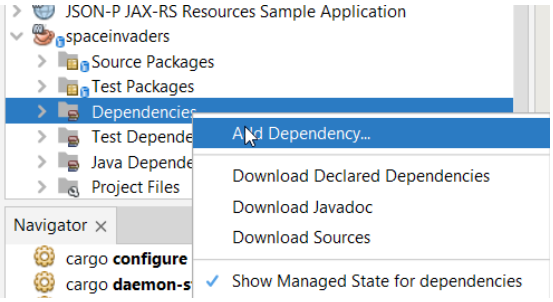
Cuando se crea un proyecto, se da la opción de crearlo usando Ant, Gradle y Maven, en el caso de Maven se tienen diferentes opciones (internamente son arquetipos).



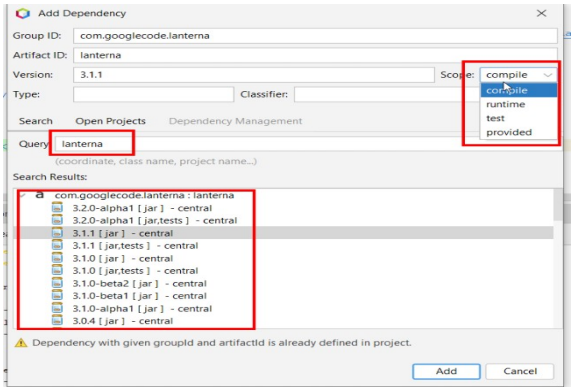
Se dispone incluso de un buscador de arquetipos:



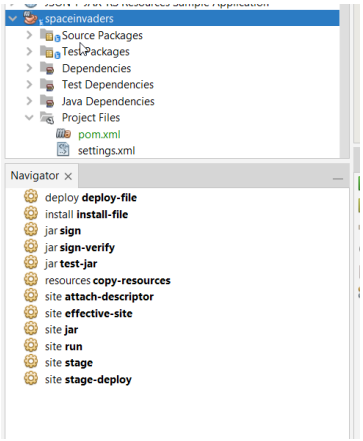
Una vez creado el proyecto, es posible añadir buscar y añadir dependencias de forma sencilla, en la ventala de “Proyect” sobre la carpeta “Dependencies” pulsar botón derecho y seleccionar “Add Dependency”:



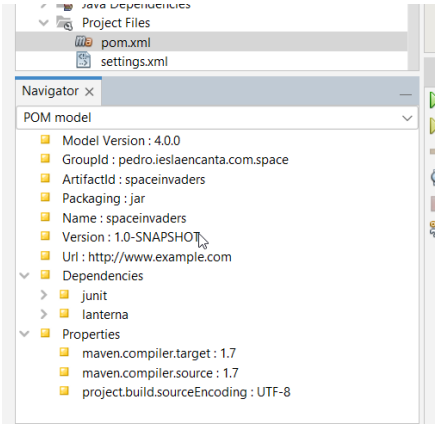
Apareciendo un buscador, pudiendo seleccionar entre otros la versión y el ámbito, al añadir modifica el fichero pom.xml:



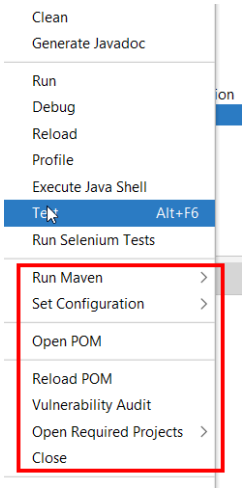
También se puede ver los diferentes “Goals” de los “plugins” en la ventana “Navigator” y ejecutarlos:



Si se hace “click” sobre el fichero pom.xml, la ventana navigator cambia, mostrando las características del fichero.

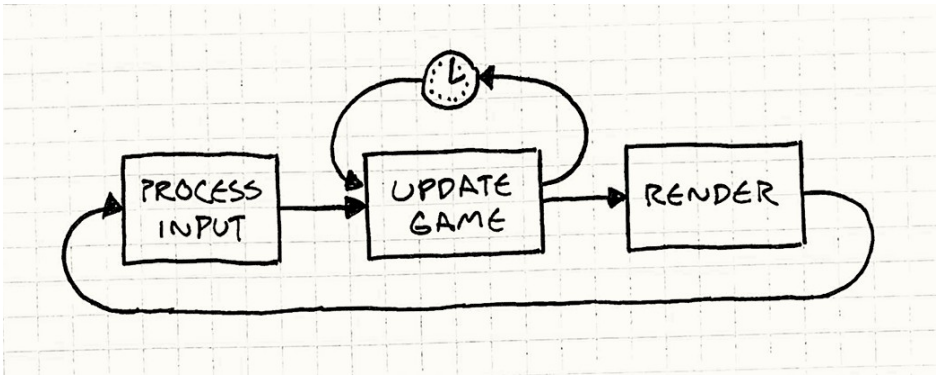


Por último, desde el menú contextual también es posible acceder a otras opciones de Maven.



Para probar que todo es correcto se crea la nave espacial y se utiliza Lanterna para dibujar el juego. El programa principal de un juego es un bucle que realiza 3 tareas principales, pudiendo existir variantes en cuanto al momento de realizar alguna de las tareas:

- Procesar las entradas.
- Actualizar el juego al siguiente estado a partir de las entradas y su estado actual.
- Renderizar



10. Código de ejemplo.

En el juego se tienen principalmente 5 clases básicas:

- Jugador: Posee una puntuación y un número de vidas.
- **Nave. Se pordrá mover en el eje horizontal y disparar.**
- Defensa. Entre los enemigos y la nave, a medida que soporta disparos se va destruyendo, existen varias.
- **Disparo. Producido tanto por los “marcianos” como por la nave.**

- Enemigo: Se podrá mover de forma horizontal y vertical, además de disparar de forma aleatorioa hacia la nave.

Entre otras clases también se ha creado la de Point2D, que representa una coordenada en pantalla, teniendo en cuenta que un terminal tiene 24 filas x 80 columnas.

Se desarrolla la clase Point2D, Ship y Shoot, teniendo en cuenta que la nave solo puede tener n balas en un instante de tiempo en el juego.

Además se crean casos de pruebas unitarias para las clases anteriores.

Clase Point2D:

```
public class Point2D {
    private int x;
    private int y;
    public Point2D(){
        this.x=-1;
        this.y=-1;
    }
    public Point2D(int x,int y){
        this.x=x;
        this.y=y;
    }

    public int getX() {
        return x;
    }

    public void setX(int x) {
        this.x = x;
    }

    public int getY() {
        return y;
    }

    public void setY(int y) {
        this.y = y;
    }
    public void addX(int incx){
        this.x+=incx;
```

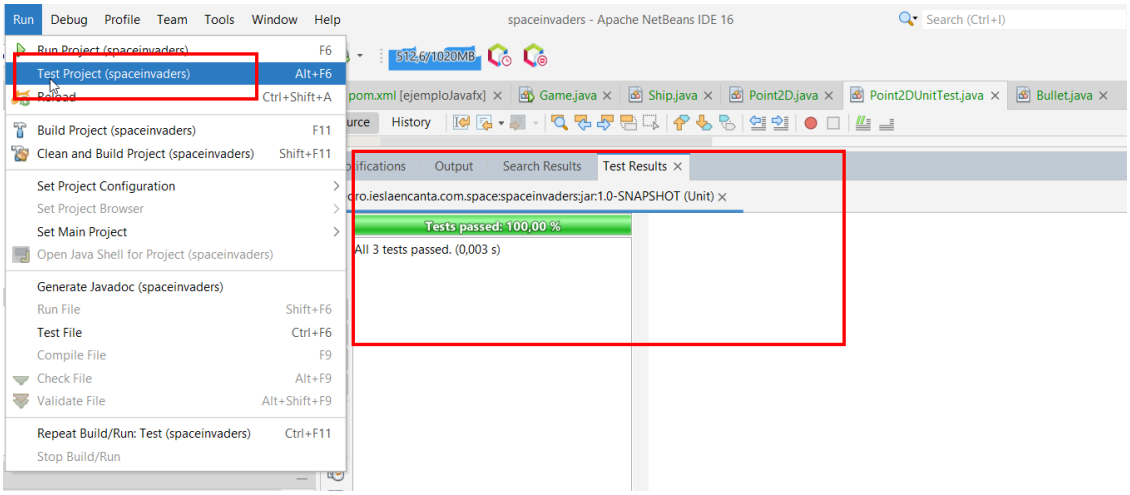
```
    }
      public void addY(int incy){
        this.y+=incy;
    }
}
```

Además se tiene una clase de prueba unitaria par esta clase:

```
public class Point2DUnitTest {

    public Point2DUnitTest() {
    }

    @BeforeClass
    public static void setUpClass() {

    }

    @AfterClass
    public static void tearDownClass() {
    }

    @Before
    public void setUp() {
    }

    @After
    public void tearDown() {
    }
    @Test
    public void defaultConstructorTest(){
        Point2D point= new Point2D();
        assertTrue(point.getX()==-1 && point.getY()==-1);
    }
    @Test
    public void overrideConstructorTest(){
        Point2D point= new Point2D(5,4);
        assertTrue(point.getX()==5 && point.getY()==4);
    }
```

```
    @Test
    public void addTest(){
        Point2D point= new Point2D(5,4);
        point.addX(5);
        point.addY(1);
        assertTrue(point.getX()==10 && point.getY()==5);
    }

}
```

Pudiendo ejecutar desde Maven o con Netbeans:



Clase Bullet.

```
public class Bullet {


    private Point2D position;


    private TextColor color;
    private TextColor backgroundcolor;
    private int width = 1;
    private int height = 1;

    public Bullet() {
        this.position = new Point2D();
        this.init();
    }

    public Bullet(Point2D p) {
```

```
        this.position = p;
        this.init();
    }

    public Bullet(int x, int y) {
        this.position = new Point2D(x, y);
        this.init();
    }

    public Point2D getPosition() {
        return position;
    }

    public void setPosition(Point2D position) {
        this.position = position;
    }

    private void init() {
        this.color = TextColor.ANSI.GREEN;
        this.backgroundcolor = TextColor.ANSI.GREEN;

    }

    public void moveVertical(int incy, int min_y, int max_y) {
        if (this.getPosition().getY() + incy >= min_y &&
this.getPosition().getY() + incy < max_y) {
            this.getPosition().addY(incy);
        } else {
            Toolkit.getDefaultToolkit().beep();
        }
    }
    public void paint(Screen s) {
        s.setCharacter(this.getPosition().getX(), this.getPosition().getY(),
new TextCharacter(' ', color, this.backgroundcolor));

    }
}
```

Clase Ship:

```
public class Ship {


    private Point2D position;
    private TextColor color;
    private TextColor backgroundcolor;
    private int width = 7;
    private int height = 4;
    private static int bullets_size = 2;
    private Bullet[] bullets;
    //por la frecuencia
    private static int max_paint_counter = 35;
    private int paint_counter = 0;


    public Ship() {
        this.position = new Point2D();
        this.init();
    }


    public Ship(Point2D p) {
        this.position = p;
        this.init();
    }


    public Ship(int x, int y) {
        this.position = new Point2D(x, y);
        this.init();
    }


    private void init() {
        this.color = TextColor.ANSI.GREEN;
        this.backgroundcolor = TextColor.ANSI.BLACK;
        this.bullets = new Bullet[Ship.bullets_size];


    }


    public void moveHorizontal(int intx, int min_x, int max_x) {
        if (this.position.getX() + intx - this.width / 2 >= min_x &&
this.position.getX() + intx + this.width / 2 < max_x) {
```

```
            this.position.addX(intx);
        } else {
            Toolkit.getDefaultToolkit().beep();
        }
    }

    public void moveBullets(int min_y, int max_y) {
        this.paint_counter++;
        //para que se pueda ver el disparo
        if (this.paint_counter >= Ship.max_paint_counter) {
            this.paint_counter = 0;

            for (int i = 0; i < this.bullets.length; i++) {
                if (this.bullets[i] != null) {
                    this.bullets[i].moveVertical(-1, min_y, max_y);
                    //en caso de llegar a la parte superior se elimina
                    if (this.bullets[i].getPosition().getY() <= min_y) {
                        this.bullets[i] = null;
                    }
                }
            }
        }
    }

    /**
     * Dibuja ^ _/ \_ !#####!
     *
     * @param s
     */
    public void paint(Screen s) {

        s.setCharacter(this.position.getX(), this.position.getY() - 1, new
TextCharacter('^', color, this.backgroundcolor));
        //_/ \_
        s.setCharacter(this.position.getX() - 2, this.position.getY(), new
TextCharacter('_', color, this.backgroundcolor));
        s.setCharacter(this.position.getX() - 1, this.position.getY(), new
TextCharacter('/', color, this.backgroundcolor));
```

```
        s.setCharacter(this.position.getX(), this.position.getY(), new
TextCharacter(' ', color, this.backgroundcolor));

        s.setCharacter(this.position.getX() + 1, this.position.getY(), new
TextCharacter('\\', color, this.backgroundcolor));

        s.setCharacter(this.position.getX() + 2, this.position.getY(), new
TextCharacter('_', color, this.backgroundcolor));


        // !#####!

        s.setCharacter(this.position.getX() - 3, this.position.getY() + 1, new
TextCharacter('|', color, this.backgroundcolor));

        s.setCharacter(this.position.getX() - 2, this.position.getY() + 1, new
TextCharacter('#', color, this.backgroundcolor));

        s.setCharacter(this.position.getX() - 1, this.position.getY() + 1, new
TextCharacter('#', color, this.backgroundcolor));

        s.setCharacter(this.position.getX(), this.position.getY() + 1, new
TextCharacter('#', color, this.backgroundcolor));

        s.setCharacter(this.position.getX() + 3, this.position.getY() + 1, new
TextCharacter('|', color, this.backgroundcolor));

        s.setCharacter(this.position.getX() + 2, this.position.getY() + 1, new
TextCharacter('#', color, this.backgroundcolor));

        s.setCharacter(this.position.getX() + 1, this.position.getY() + 1, new
TextCharacter('#', color, this.backgroundcolor));

        for (int i = 0; i < this.bullets.length; i++) {

            if (this.bullets[i] != null) {

                this.bullets[i].paint(s);

            }

        }

    }


    public void shoot() {

        Bullet tempo;

        boolean shooted = false;

        //solo  dispara si tiene un disparo libre

        for (int i = 0; i < this.bullets.length && !shooted; i++) {

            if (this.bullets[i] == null) {

                tempo = new Bullet(this.position.getX(), this.position.getY() -
2);

                this.bullets[i] = tempo;

                shooted = true;

            }

        }

    }
```

```
}
```

Clase Game, encargada del bucle del juego:

```
public class Game {

    //dimensiones de un terminal
    private static int COLUMNS = 80;
    private static int ROWS = 24;
    //20 MHz
    private static int frecuency = 2;
    private Terminal terminal;
    private Screen screen;
    private TextColor background;
    private boolean key_left_pressed;
    private boolean key_right_pressed;
    private boolean key_exit;
    private boolean key_shoot;
    private Ship ship;

    public Game() {
        this.key_left_pressed = false;
        this.key_right_pressed = false;
        this.key_exit = false;
        this.key_shoot=false;
        //se crea la nave
        this.background=TextColor.ANSI.BLACK;
        this.ship = new Ship(Game.COLUMNS / 2, ROWS - 3);
        try {
            this.terminal = new DefaultTerminalFactory().createTerminal();
            this.screen = new TerminalScreen(this.terminal);
            //no se muestra el cursor
            screen.setCursorPosition(null);
        } catch (IOException ex) {
            Logger.getLogger(Game.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    public void loop() {
        try {
```

```
            screen.startScreen();
            screen.clear();
            this.terminal.setBackgroundColor(TextColor.ANSI.CYAN);

            while (!this.key_exit) {
                try {
                    //se procesa la entrada
                    this.process_input();
                    //se actualiza el juego
                    this.update();
                    //se pinta
                    this.paint(this.screen);
                    //1000 es un segundo, frecuenca de 10 Hz son 10 veces por
segundo
                    //frecuenca de 20 Hz son 20 veces por segundo, una vez cada
0,05 segundos
                    Thread.sleep((1 / Game.frecuency) * 1000);
                } catch (InterruptedException ex) {
                    Logger.getLogger(Game.class.getName()).log(Level.SEVERE,
null, ex);
                }
            }
            //fin del bucle
            screen.stopScreen();
        } catch (IOException ex) {
            Logger.getLogger(Game.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    private void paint(Screen s) {
        try {
            TerminalSize terminalSize = s.getTerminalSize();
            for (int column = 0; column < terminalSize.getColumns(); column++)
{
                for (int row = 0; row < terminalSize.getRows(); row++) {
                    s.setCharacter(column, row, new TextCharacter(
                            ' ',
                            TextColor.ANSI.DEFAULT,
                            this.background));
```

```
                }
            }
            this.ship.paint(s);
            screen.refresh();
        } catch (IOException ex) {
            Logger.getLogger(Game.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    /**
     * Borrar el buffer de teclado para evitar saltos en el movimiento
     *
     */
    private void clear_keyboard_input() {
        KeyStroke keyStroke = null;
        do {
            try {
                keyStroke = screen.pollInput();
            } catch (IOException ex) {
                Logger.getLogger(Game.class.getName()).log(Level.SEVERE, null,
ex);
            }
        } while (keyStroke != null);
    }

    private void process_input() {
        this.key_left_pressed = false;
        this.key_right_pressed = false;
        this.key_shoot=false;
        try {
            //la lectura es no bloqueante
            KeyStroke keyStroke = screen.pollInput();

            if (keyStroke != null) {
                if (keyStroke.getKeyType() == KeyType.Escape) {
                    this.key_exit = true;
                }
```

```
                if (keyStroke.getKeyType() == KeyType.ArrowLeft) {
                    this.key_left_pressed = true;
                }
                if (keyStroke.getKeyType() == KeyType.ArrowRight) {
                    this.key_right_pressed = true;
                }
                if(keyStroke.getKeyType() ==KeyType.Enter)
                    this.key_shoot=true;
                //se borra el buffer
                this.clear_keyboard_input();
            }

        } catch (IOException ex) {
            Logger.getLogger(Game.class.getName()).log(Level.SEVERE, null, ex);
        }

    }

    private void update() {
        if (this.key_left_pressed) {
            this.ship.moveHorizontal(-1, 0, COLUMNS - 1);
        }
        if (this.key_right_pressed) {
            this.ship.moveHorizontal(1, 0, COLUMNS - 1);
        }
        //se mueven las balas
        this.ship.moveBullets(0, ROWS);
        //se dispara si se ha pulsado la tecla
        if(this.key_shoot)
            this.ship.shoot();
    }

    public boolean isKey_left_pressed() {
        return key_left_pressed;
    }

    public void setKey_left_pressed(boolean key_left_pressed) {
        this.key_left_pressed = key_left_pressed;
```

```
    }

    public boolean isKey_right_pressed() {
        return key_right_pressed;
    }

    public void setKey_right_pressed(boolean key_right_pressed) {
        this.key_right_pressed = key_right_pressed;
    }

    public static void main(String[] args) {
        Game game = new Game();
        game.loop();

    }

}
```