

IES Enrique Tierno Galván de Madrid

Introducción a Maven

Proyectos con Maven en Netbeans

Introducción a MAVEN

Contenido

1. Introducción	3
1.1. ¿Qué es Maven?	3
1.2. Configuración	3
2. Creación de un proyecto en Netbeans con Maven	3
3. Gestión de dependencias	5
3.1. ¿Qué son las dependencias en Maven?	5
3.2. Añadir dependencias al archivo <code>pom.xml</code>	5
3.3. Verificar las dependencias descargadas	6
3.4. Usar dependencias en el código	6
3.5. Resolver conflictos de dependencias	7
3.6. Repositorios adicionales (opcional)	7
4. Ejecución de Fases del Ciclo de Vida de Maven en NetBeans	7
4.1. ¿Qué es el ciclo de vida de Maven?	7
4.2. Ejecutar fases del ciclo de vida en NetBeans	8
5. Personalización del archivo <code>pom.xml</code>	9
5.1. ¿Qué es el archivo <code>pom.xml</code> ?	9
5.2. Estructura básica de un archivo <code>pom.xml</code>	9
5.3. Elementos clave del archivo <code>pom.xml</code>	10
Ejemplo práctico: Personalización	11
6. Bibliotecas locales	12
Opción 1: Instalar la biblioteca en tu repositorio local de Maven	12
Opción 2: Referenciar directamente el archivo JAR	13
Anexo 1. Archivo ejecutable con bibliotecas	14
Anexo 2. Imágenes	15
¿Qué consecuencias tiene para las imágenes del proyecto?	15
¿Cómo se soluciona?	15

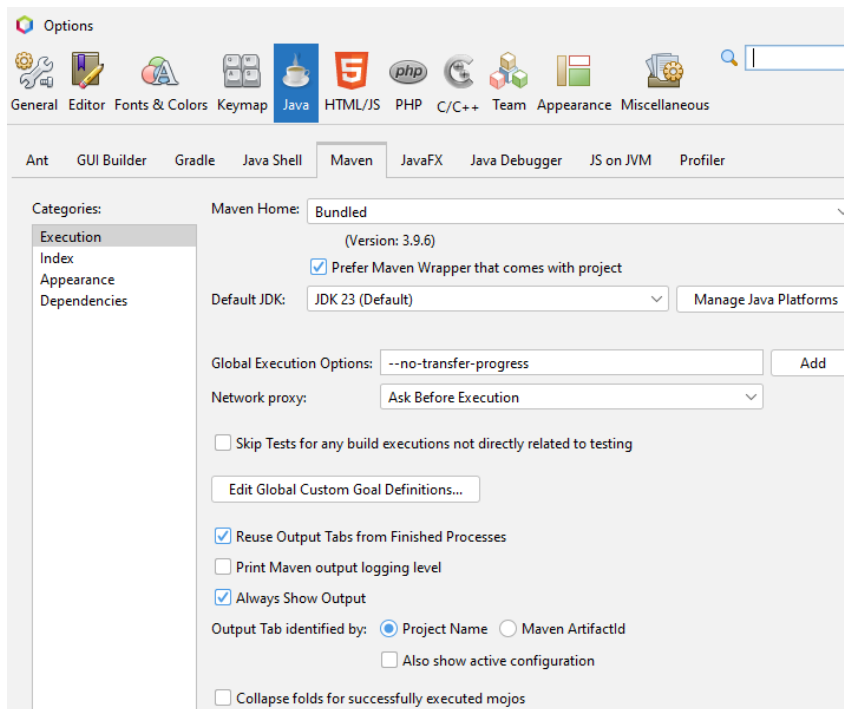
1. Introducción

1.1. ¿Qué es Maven?

Maven es una herramienta de gestión y comprensión de proyectos Java que permite manejar dependencias, construir el proyecto y mucho más. Se basa en el archivo de configuración `pom.xml`.

1.2. Configuración

Para configurar Maven se debe ir a **Tools > Options**.



2. Creación de un proyecto en Netbeans con Maven

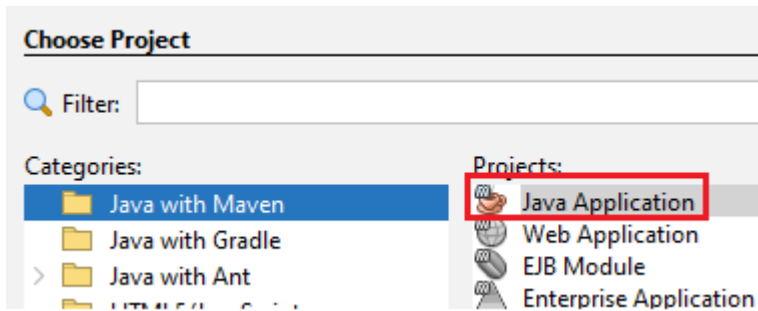
Para crear un proyecto en Netbeans con Maven hay que seguir los siguientes pasos:

1. Abrir NetBeans

Asegúrate de que NetBeans esté correctamente configurado con Maven (si no, revisa el paso anterior).

2. Crear un nuevo proyecto Maven

1. En el menú principal de NetBeans, selecciona:
Archivo > Nuevo Proyecto...
2. En la ventana que aparece:
 - Categoría: Selecciona **Maven**.
 - Proyectos: Selecciona **Aplicación Java con Maven** (o un tipo de proyecto que prefieras).
 - Haz clic en **Siguiente**.



3. Configurar el proyecto

1. Completa los campos requeridos:
 - **Grupo ID:** Un identificador único para tu proyecto, como `com.ejemplo`.
 - **Artifact ID:** El nombre de tu proyecto (por ejemplo, `mi-proyecto`).
 - **Versión:** La versión inicial de tu proyecto (por defecto, `1.0-SNAPSHOT`).
 - **Nombre del paquete:** NetBeans sugerirá uno basado en el Grupo ID. Puedes cambiarlo si deseas.

Name and Location	
Project Name:	ProyectoMaven01
Project Location:	E:\NetbeansProjects_Maven
Project Folder:	E:\NetbeansProjects_Maven\ProyectoMaven01
Artifact Id:	ProyectoMaven01
Group Id:	com.tierno
Version:	1.0-SNAPSHOT
Package:	com.tierno.proyectomaven01

Haz clic en **Siguiente**.

4. Elegir configuración adicional (opcional)

1. Selecciona un arquetipo, si quieres usar un modelo preconfigurado para tu proyecto. Por defecto, se usará el arquetipo más básico.
 - **Arquetipos comunes:**
 - `maven-archetype-quickstart`: Para una aplicación Java sencilla.
 - `maven-archetype-webapp`: Para aplicaciones web.
 - Si no necesitas uno en particular, deja la configuración predeterminada y haz clic en **Finalizar**.

5. Revisar el proyecto creado

NetBeans generará una estructura de proyecto estándar de Maven:

- **src/main/java:** Carpeta principal para tu código fuente.
- **src/test/java:** Carpeta para tus pruebas unitarias.
- **pom.xml:** Archivo de configuración de Maven. Aquí manejarás dependencias y configuraciones de tu proyecto.

6. Ejecutar el proyecto

1. Haz clic derecho en el proyecto desde la ventana de proyectos.
2. Selecciona **Construir o Ejecutar**.
 - Al construir, Maven ejecutará las fases del ciclo de vida (como `compile`, `test`, y `package`).
3. Si todo está bien configurado, deberías ver la salida en la consola integrada.

7. Añadir dependencias al proyecto (opcional)

1. Abre el archivo `pom.xml`.
2. En la sección `<dependencies>`, añade las dependencias que necesites, por ejemplo:

```
<dependencies>
  <dependency>
    <groupId>org.apache.commons</groupId>
    <artifactId>commons-lang3</artifactId>
    <version>3.17.0</version>
  </dependency>
</dependencies>
```

3. Guarda los cambios. Maven descargará automáticamente las dependencias al construir el proyecto.

3. Gestión de dependencias

3.1. ¿Qué son las dependencias en Maven?

Las dependencias son bibliotecas externas necesarias para que un proyecto funcione. Se declaran en el archivo `pom.xml`, y Maven se encarga de descargarlas automáticamente desde repositorios como Maven Central.

3.2. Añadir dependencias al archivo `pom.xml`

1. **Abrir el archivo `pom.xml`:**
 - Ve al proyecto en NetBeans.
 - Haz doble clic en el archivo `pom.xml` (ubicado en la raíz del proyecto).
2. **Localizar la sección `<dependencies>`:**
 - Si no existe, añade el siguiente bloque básico:

```
<dependencies>
  <!-- Aquí irán tus dependencias -->
</dependencies>
```

3. **Añadir una dependencia:** Cada dependencia necesita tres elementos principales:
 - **groupId:** Identificador del grupo o la organización que proporciona la biblioteca.

- **artifactId**: Nombre específico de la biblioteca.
- **version**: Versión de la biblioteca que deseas usar.
- **scope** (Opcional): Especifica el alcance de la dependencia. Algunos valores comunes son:
 1. **compile** (por defecto): La dependencia se necesita en todas las fases (compilación, pruebas, ejecución).
 2. **test**: Solo se usa en pruebas (por ejemplo, JUnit).
 3. **Otras**: **provided**, **runtime** y **system**.

Por ejemplo, para añadir la biblioteca `Apache Commons Lang`:

```
<dependency>
  <groupId>org.apache.commons</groupId>
  <artifactId>commons-lang3</artifactId>
  <version>3.17.0</version>
</dependency>
```

4. **Guardar el archivo `pom.xml`**: NetBeans detectará automáticamente el cambio y descargará la dependencia.

3.3. Verificar las dependencias descargadas

1. Abre la pestaña **Archivos** o **Dependencias** (en la vista del proyecto).
2. Busca las dependencias añadidas dentro de **Dependencias**.
 - Maven descarga las dependencias en tu repositorio local (generalmente en `~/.m2/repository`).

3.4. Usar dependencias en el código

1. Importa las clases de la dependencia en tus archivos `.java`. Por ejemplo:

```
import org.apache.commons.lang3.StringUtils;

public class Main {
    public static void main(String[] args) {
        System.out.println(StringUtils.toUpperCase("hola maven"));
    }
}
```

2. Ejecuta el proyecto para verificar que la dependencia funciona correctamente.

3.5. Resolver conflictos de dependencias

A veces, dos bibliotecas tienen dependencias transitorias que entran en conflicto (distintas versiones del mismo archivo). Para resolverlo:

1. Usa la herramienta **Dependencias** en NetBeans:
 - Haz clic derecho en el proyecto y selecciona **Resolución de dependencias**.
 - NetBeans mostrará posibles conflictos y te sugerirá exclusiones.
2. Manualmente, puedes excluir dependencias en el `pom.xml`:

```
<dependency>
  <groupId>org.example</groupId>
  <artifactId>library</artifactId>
  <version>1.0</version>
  <exclusions>
    <exclusion>
      <groupId>org.conflict</groupId>
      <artifactId>conflicting-library</artifactId>
    </exclusion>
  </exclusions>
</dependency>
```

3.6. Repositorios adicionales (opcional)

Si necesitas bibliotecas que no están en Maven Central:

1. Añade un repositorio en el archivo `pom.xml`:

```
<repositories>
  <repository>
    <id>repositorio-ejemplo</id>
    <url>https://ejemplo.com/maven</url>
  </repository>
</repositories>
```

2. Declara la dependencia como de costumbre.

4. Ejecución de Fases del Ciclo de Vida de Maven en NetBeans

4.1. ¿Qué es el ciclo de vida de Maven?

Maven organiza el proceso de construcción de un proyecto en una serie de **fases**. Estas fases son pasos predefinidos que forman parte del ciclo de vida de construcción. Al ejecutar una fase, Maven automáticamente ejecutará todas las fases anteriores en orden.

Esto significa que las fases del ciclo de vida de Maven están diseñadas para ser incrementales. Cuando ejecutas una fase, Maven se asegura de que las fases anteriores también se hayan completado (si no se han ejecutado aún).

Además, no es necesario ejecutar todas las fases, solamente aquellas que se necesiten. Por ejemplo, puede ser solamente necesario compilar.

Ciclos de vida principales de Maven

1. **Ciclo de vida de construcción estándar (default):**

Maneja tareas como compilación, pruebas y empaquetado del proyecto.

- Fases principales en orden:
 1. **validate:** Verifica que el proyecto esté correcto y que toda la información esté disponible.
 2. **compile:** Compila el código fuente del proyecto.
 3. **test:** Ejecuta las pruebas unitarias.
 4. **package:** Empaqueta el proyecto (por ejemplo, genera un archivo JAR o WAR).
 5. **verify:** Ejecuta verificaciones adicionales para asegurar que el paquete es válido y cumple con los estándares de calidad requeridos.
 6. **install:** Instala el paquete en el repositorio local para su uso en otros proyectos.
 7. **deploy:** Despliega el paquete a un repositorio remoto.

2. **Ciclo de vida de limpieza (clean):**

Limpia los archivos generados en compilaciones previas.

- Fase importante:
 - **clean:** Elimina los directorios de salida (target).

3. **Ciclo de vida de generación de sitio (site):**

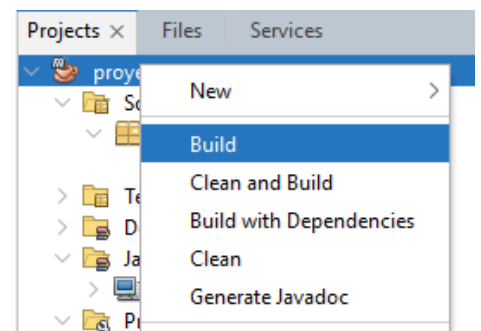
Genera documentación del proyecto.

- Fase importante:
 - **site:** Genera la documentación del proyecto.

4.2. Ejecutar fases del ciclo de vida en NetBeans

4.2.1. Desde el menú contextual del proyecto

1. Haz clic derecho en tu proyecto Maven en la ventana de proyectos.
2. Selecciona **Construir** para ejecutar la fase `package` (o el equivalente configurado como predeterminado).
3. O selecciona directamente una de las siguientes opciones:
 - **Limpiar:** Ejecuta la fase `clean`.
 - **Limpiar y Construir:** Ejecuta `clean` seguido de `package`.



5. Personalización del archivo pom.xml

5.1. ¿Qué es el archivo pom.xml?

El archivo `pom.xml` (**Project Object Model**) es el núcleo de un proyecto Maven. Define la configuración del proyecto, incluyendo:

- Identificación del proyecto.
- Dependencias requeridas.
- Configuración de plugins.
- Configuración del ciclo de vida de Maven.
- Propiedades específicas del proyecto.

5.2. Estructura básica de un archivo pom.xml

Aquí tienes un ejemplo básico con los elementos más comunes:

```
<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>

  <!-- Información básica del proyecto -->
  <groupId>com.ejemplo</groupId>
  <artifactId>mi-proyecto</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>jar</packaging> <!-- jar, war, pom, etc. -->

  <name>Mi Proyecto</name>
  <description>Descripción del proyecto</description>

  <!-- Dependencias del proyecto -->
  <dependencies>
    <dependency>
      <groupId>org.apache.commons</groupId>
      <artifactId>commons-lang3</artifactId>
      <version>3.12.0</version>
    </dependency>
  </dependencies>

  <!-- Configuración de plugins -->
  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
```

```

        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.8.1</version>
        <configuration>
            <source>1.8</source> <!-- Versión del código
fuente -->
            <target>1.8</target> <!-- Versión del bytecode
generado -->
        </configuration>
    </plugin>
</plugins>
</build>
</project>

```

5.3. Elementos clave del archivo pom.xml

1. Identificación del proyecto

Define cómo Maven identifica tu proyecto:

- <groupId>: Identifica tu organización o dominio único. Ejemplo: com.miempresa.
- <artifactId>: Nombre del proyecto. Ejemplo: mi-proyecto.
- <version>: Versión del proyecto. Ejemplo: 1.0.0.

2. Dependencias

Declaras bibliotecas externas que tu proyecto necesita:

```

<dependencies>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-core</artifactId>
    <version>5.3.10</version>
  </dependency>
</dependencies>

```

3. Repositorios adicionales (opcional)

Si necesitas descargar dependencias desde un repositorio no estándar:

```

<repositories>
  <repository>
    <id>repositorio-ejemplo</id>
    <url>https://ejemplo.com/maven</url>
  </repository>
</repositories>

```

4. Configuración del ciclo de vida y plugins

Puedes personalizar cómo se compila, empaqueta o prueba tu proyecto:

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.8.1</version>
      <configuration>
        <source>11</source>
        <target>11</target>
      </configuration>
    </plugin>
  </plugins>
</build>
```

5. Propiedades

Define valores reutilizables en todo el `pom.xml`:

```
<properties>
  <java.version>11</java.version>
</properties>
```

Usa estas propiedades en otros lugares:

```
<configuration>
  <source>${java.version}</source>
  <target>${java.version}</target>
</configuration>
```

Ejemplo práctico: Personalización

1. Añadir una dependencia específica:

```
<dependency>
  <groupId>org.junit.jupiter</groupId>
  <artifactId>junit-jupiter</artifactId>
  <version>5.8.2</version>
  <scope>test</scope>
</dependency>
```

2. Configurar compilación para Java 17:

```

<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.8.1</version>
      <configuration>
        <source>17</source>
        <target>17</target>
      </configuration>
    </plugin>
  </plugins>
</build>

```

6. Bibliotecas locales

Si tenemos una biblioteca local que queremos utilizar en nuestro proyecto, podemos hacerlo de varias formas. A continuación mostrados dos formas:

Opción 1: Instalar la biblioteca en tu repositorio local de Maven

Maven organiza las dependencias en repositorios locales, remotos o centrales. Puedes instalar manualmente tu archivo JAR en el repositorio local.

Pasos:

1. Abre una terminal y ejecuta el siguiente comando:

```

mvn install:install-file -Dfile=/ruta/a/tu/archivo.jar
-DgroupId=mi.grupo -DartifactId=mi-biblioteca -Dversion=1.0
-Dpackaging=jar

```

Parámetros explicados:

- -Dfile: Ruta completa al archivo JAR.
 - -DgroupId: Identificador del grupo (puedes elegir uno como com.miempresa).
 - -DartifactId: Nombre del archivo o biblioteca.
 - -Dversion: La versión de la biblioteca.
 - -Dpackaging: El tipo de archivo (en este caso, jar).
2. Verifica que el archivo JAR se haya instalado en el repositorio local. Normalmente se encuentra en:

```

~/ .m2/repository/<groupId>/<artifactId>/<version>/

```

3. Declara esta dependencia en tu archivo `pom.xml`:

```
<dependency>
  <groupId>mi.grupo</groupId>
  <artifactId>mi-biblioteca</artifactId>
  <version>1.0</version>
</dependency>
```

Ejemplo (para Windows)

```
"C:\Program Files\NetBeans-20\netbeans\java\maven\bin\mvn.cmd"
org.apache.maven.plugins:maven-install-plugin:2.3.1:install-file
-Dfile="D:\\Bibliotecas\\isiHSQL-1.0.jar"
-DgroupId=com.tierno
-DartifactId=Tierno
-Dversion=1.0
-Dpackaging=jar
-DlocalRepositoryPath="C:\\Users\\Profesor\\.m2\\repository\\"
```

Observaciones: No es conveniente copiar el ejemplo anterior y pegarlo en el terminal. Es mejor pegarlo en un editor de texto, modificarlo y luego, sí, pegarlo en el terminal para su ejecución.

Opción 2: Referenciar directamente el archivo JAR

Si no deseas instalar la biblioteca en el repositorio local, puedes referenciar el archivo directamente en el `pom.xml`.

Pasos:

1. Coloca el archivo JAR en un directorio dentro de tu proyecto, por ejemplo, `libs/`.
2. Modifica el archivo `pom.xml` para incluir el JAR como una dependencia utilizando el plugin `system`:

```
<dependency>
  <groupId>mi.grupo</groupId>
  <artifactId>mi-biblioteca</artifactId>
  <version>1.0</version>
  <scope>system</scope>
  <systemPath>${project.basedir}/libs/archivo.jar</systemPath>
</dependency>
```

Nota: El uso de `systemPath` no es recomendado para proyectos grandes o en equipos, ya que puede generar problemas al compartir el proyecto.

Anexo 1. Archivo ejecutable con bibliotecas

Para que genere un archivo ejecutable con las bibliotecas incluidas es necesario añadir el siguiente trozo de XML a pom.xml. Este código se debe adaptar al proyecto en el que se utilice (**debe modificarse lo que está con color de fondo amarillo**).

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-assembly-plugin</artifactId>
      <version>3.4.0</version>
      <configuration>
        <descriptorRefs>
<descriptorRef>jar-with-dependencies</descriptorRef>
        </descriptorRefs>
        <archive>
          <manifest>
<mainClass>com.tierno.proyecto01.Proyecto01</mainClass>
          </manifest>
        </archive>
      </configuration>
      <executions>
        <execution>
          <phase>package</phase>
          <goals>
            <goal>single</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>
```

Algunas opciones comunes para **<descriptorRef>** son:

- **jar-with-dependencies:** Incluye el archivo JAR principal junto con todas las dependencias necesarias en un único archivo JAR ejecutable.
- **jar-with-dependencies-and-sources:** Similar al anterior, pero también incluye los archivos fuente en el JAR.
- **project-descriptor:** Usa una configuración personalizada definida dentro del proyecto, en lugar de las predefinidas.
- **bin:** Empaqueta los archivos binarios (clases compiladas) en un archivo ZIP.
- **src:** Empaqueta únicamente los archivos fuente en un archivo ZIP o tarball.
- **dist:** Empaqueta una distribución completa de la aplicación, que puede incluir binarios, dependencias y otros recursos necesarios.
- **repository:** Crea un repositorio Maven completo, incluyendo dependencias, en un directorio empaquetado.

- **runtime:** Similar a jar-with-dependencies, pero orientado a empaquetar solo las dependencias necesarias para la ejecución en tiempo de ejecución (excluye dependencias de prueba y compilación).
- **assembly-descriptor:** Te permite especificar un descriptor completamente personalizado en el archivo de configuración del plugin assembly.

Anexo 2. Imágenes

Maven usa el directorio `src/main/java` (mapeado a Source Packages en NetBeans) solo para el código fuente, por lo que cualquier archivo cuya extensión no sea `.java` será excluido al momento de la compilación y por lo tanto no se depositará en el directorio `target\classes` ni en el JAR que se genere.

¿Qué consecuencias tiene para las imágenes del proyecto?

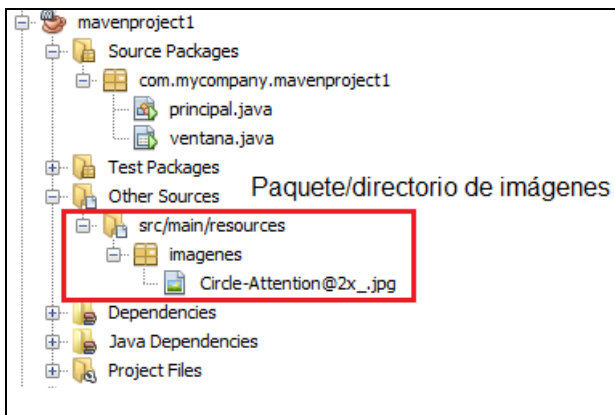
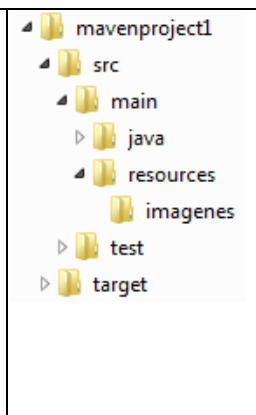
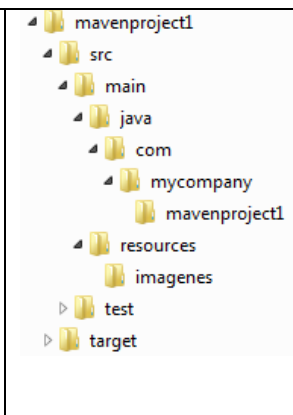
Esto significa que las imágenes de nuestro proyecto no se incluyen en el directorio desde el cual se ejecuta la aplicación (`target/classes`).

¿Cómo se soluciona?

Se deben colocar los archivos que no son `.java` (sean imágenes o cualquier otro tipo de archivo) en el directorio `src/main/resources`.

A continuación se muestran algunas imágenes sobre dónde deberían estar los paquetes/directorios. Al tratarse de imágenes, se ha creado una carpeta con nombre "imagenes".

```
src/
└─ main/
   ├── java/
   │   └── com/
   │       └── ejemplo/
   │           └── MainApp.java
   └── resources/
       └── images/
           ├── logo.png
           └── fondo.jpg
```

		
<p>En Netbeans, rodeado de un cuadrado rojo, aparece dónde deberían estar las imágenes (y otros archivos que no son código fuente o <code>.java</code>).</p>	<p>En estas imágenes se ve cómo quedaría la estructura de directorios de nuestro proyecto.</p>	