Maven

Introducción

Maven

- Es una utilidad para crear y administrar proyectos basados en Java
- Como objetivos se propone:
  - Proporcionar un sistema de construcción uniforme
  - Proporcionar información de calidad del proyecto
  - Proporcionar pautas para el desarrollo de mejores prácticas
  - Permitir la migración transparente a nuevas funcionalidades
- Permite declarar **dependencias** para utilizar librerías externas (o nuestras)

https://maven.apache.org

Cómo me ayuda Maven

- Maven, compilá el proyecto
- Maven, armá un .jar con lo que ya compilé
- Maven, compilá y armá un .jar
- Maven, corré los tests
- Maven, guardá el proyecto localmente así lo uso como dependencia en otros proyectos

Más info: https://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html

Goals & Build Phases

Goals: tareas específicas dentro del build

- mvn jar:jar → armar un .jar desde el código ya compilado
- mvn dependency:tree → muestra las dependencias
- mvn exec: java → corre el proyecto

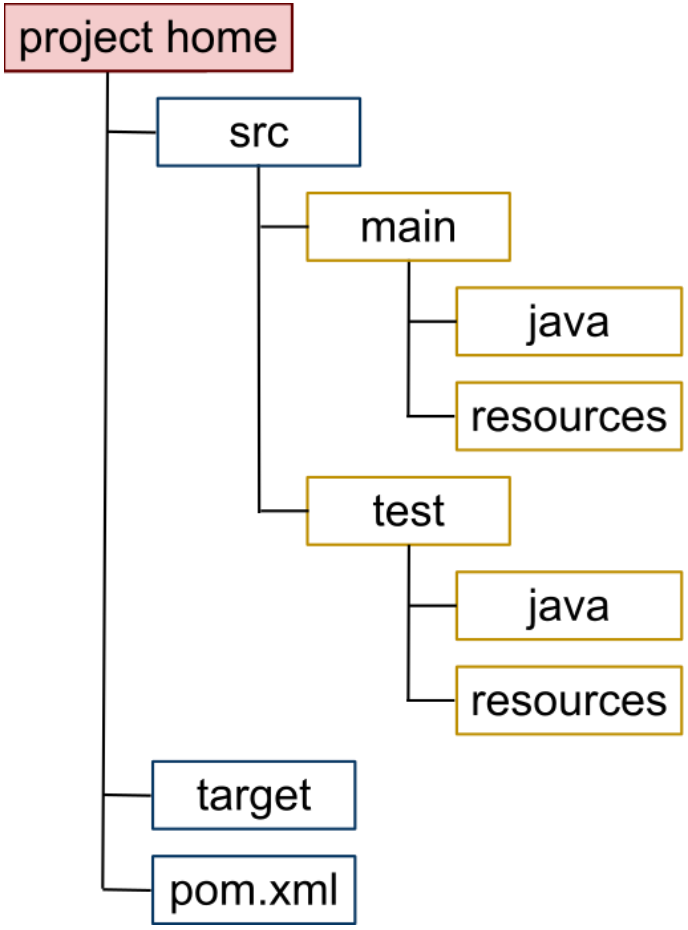
Identifica al plugin **Identifica al goal**

Build Phases: etapas del armado del proyecto

- mvn compile → compila el código
- mvn test→ corre los tests
- mvn package → arma el .jar
- mvn install → guarda el proyecto en el repo local

Las Build Phases ejecutan todo lo de las etapas anteriores!

Estructura Proyecto



pom.xml mínimo

```
<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>ar.edu.itba.eda</groupId>
  <artifactId>Timer</artifactId>
  <version>1.0</version>
</project>
```

Versión de Java

Si queremos estar seguros de que estamos compilando con Java 11 (sobre todo cuando usamos generics donde queremos garantizar cierta versión), agregamos:

Opción 1:

```
<project … >
…
        <properties>
              <maven.compiler.source>11</maven.compiler.source>
             <maven.compiler.target>11</maven.compiler.target>
             <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
         </properties>
…
</project>
```

Versión de Java

Opción 2:

```
<project … >
…
        <build>
                <plugins>
                        <plugin>
                                <groupId>org.apache.maven.plugins</groupId>
                                <artifactId>maven-compiler-plugin</artifactId>
                                <version>3.8.0</version>
                                <configuration>
                                        <release>11</release>
                                        <encoding>UTF-8</encoding>
                                </configuration>
                        </plugin>
                </plugins>
        </build>
…
</project>
```

Dependencias

En el pom.xml se declaran, además, las dependencias a utilizar.

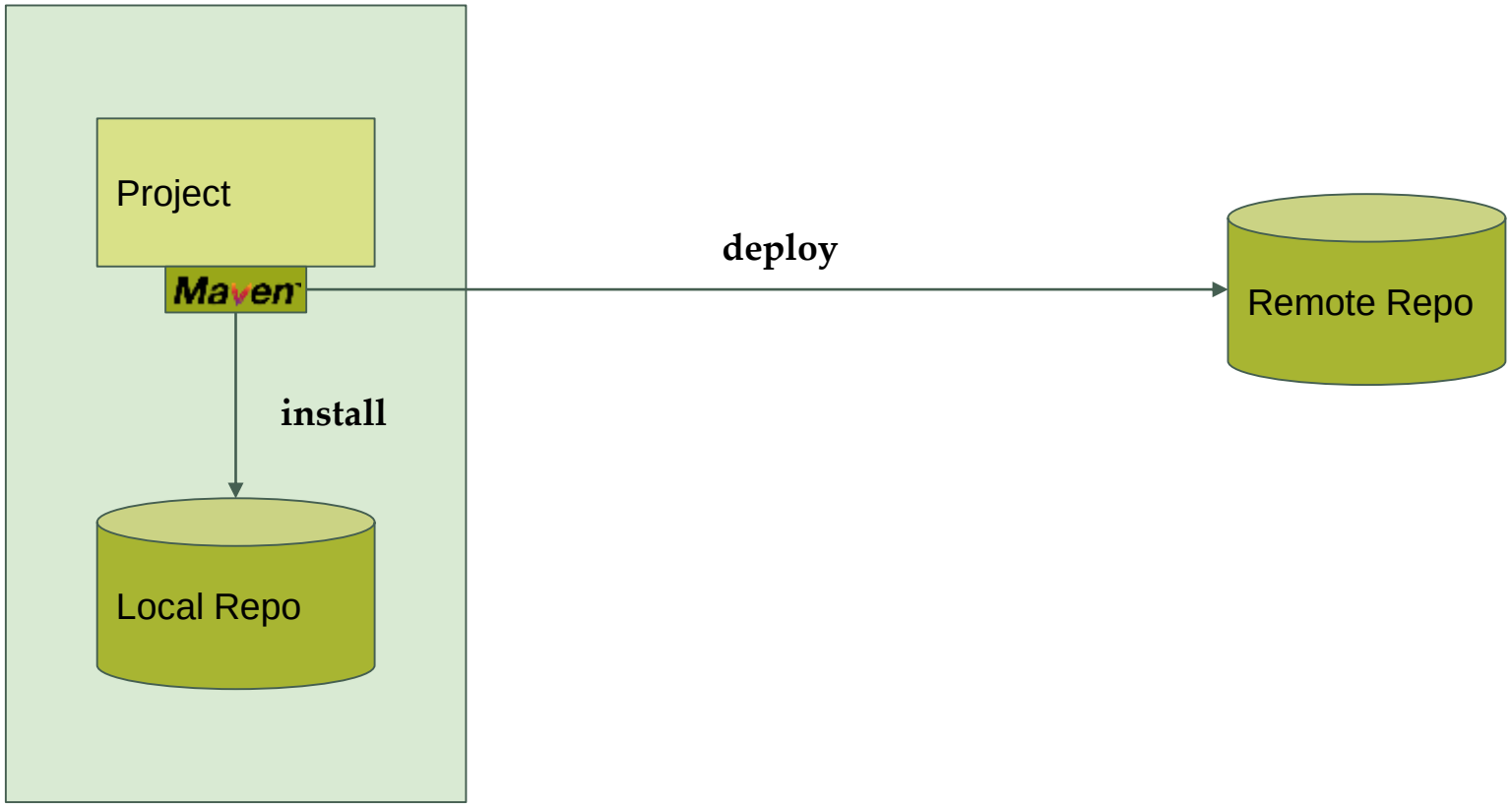
Algoritmo: Maven busca primero localmente a la dependencia en nuestro repositorio local. En caso de no encontrarla la descarga del repositorio correspondiente al repositorio local de nuestra compu.

Ese repositorio local típicamente se encuentra en $HOME/.m2

Ejemplo:

- C:\Users\lgomez\.m2
- /Users/jabu/.m2
- /home/luis/.m2

Repositorios



TP 1-Ejer 3.1

Migrar la clase MyTimer anterior para que sea un proyecto Maven. Será nuestra primera versión.

TP 1-Ejer 3.2

Agregar un nuevo plugin y ver qué genera

Desde el directorio del proyecto donde se encuentra el pom.xml
Ejecutar con **mvn exec:java**

```
<project … >
        …
    <build>
        <plugins>
                …
            <plugin>
                <groupId>org.codehaus.mojo</groupId>
                <artifactId>exec-maven-plugin</artifactId>
                <version>3.0.0</version>
                <configuration>
                    <mainClass>Main</mainClass>
                </configuration>
            </plugin>
            …
        </plugins>
    </build>
    …
</project>
```

TP 1-
Ejer 4.1 y 4.2

Implementar la clase `MyTimer` (`versión 2,` usando la biblioteca Joda y Maven)

El proyecto se llamará TimerJoda

Crear un nuevo proyecto Maven desde el IDE con dicha dependencia.

La funcionalidad a implementar y caso de uso, sigue siendo la misma.

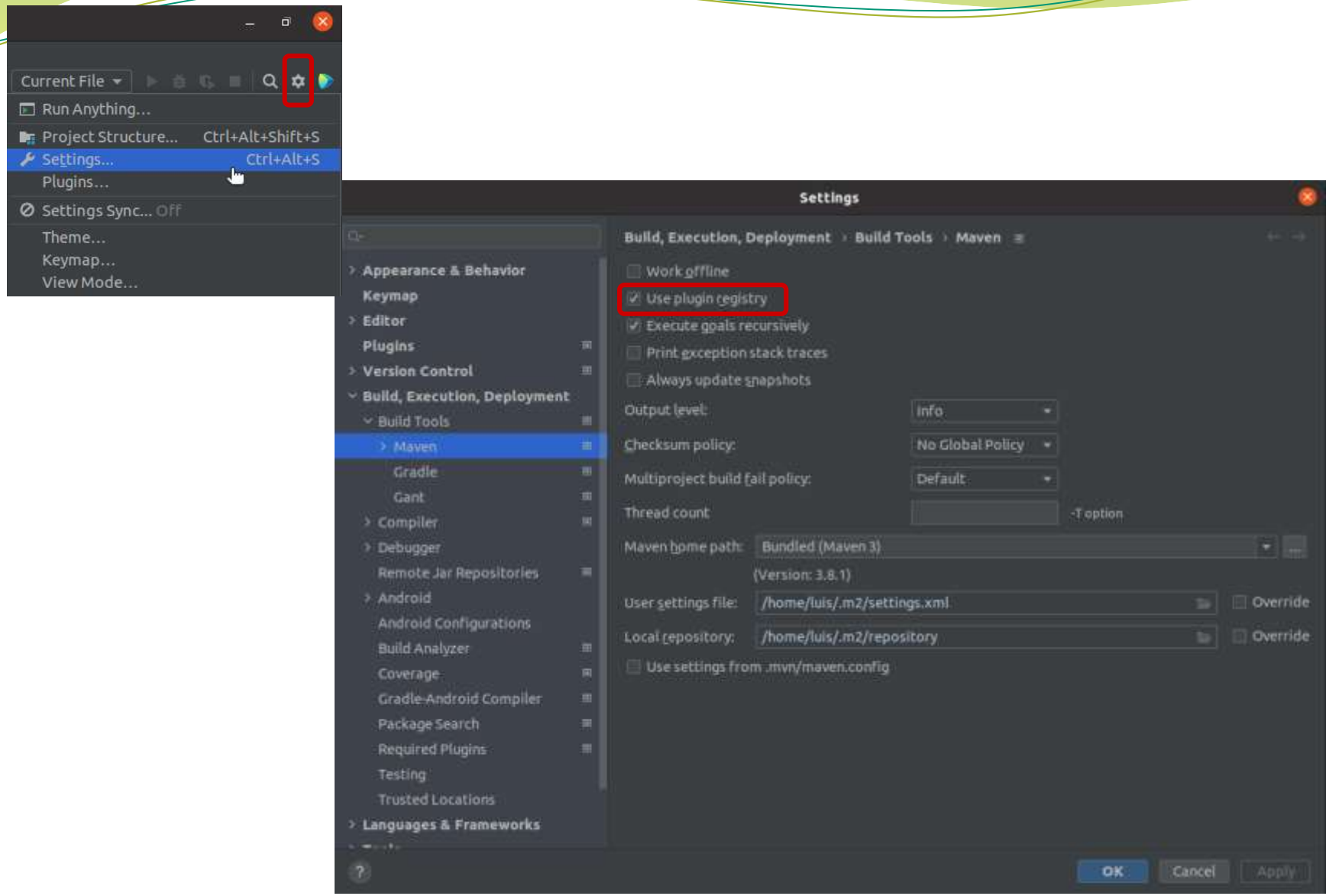
Tip: investigar las clases **`Instant`** y **`Period`**

https://www.joda.org/joda-time/
(leer detenidamente su especificación)

(nuestra clase será un wrapper)

La dependencia de joda-time:

```
<project … >
      …
   <dependencies>
      <!-- https://mvnrepository.com/artifact/joda-
      time/joda-time -->
      <dependency>
          <groupId>joda-time</groupId>
          <artifactId>joda-time</artifactId>
          <version>2.10.10</version>
      </dependency>
   </dependencies>
   …
</project>
```


Current File
Run Anything...
Project Structure... Ctrl+Alt+Shift+S
Settings... Ctrl+Alt+S
Plugins...
Settings Sync... Off
Theme...
Keymap...
View Mode...
Settings
Build, Execution, Deployment › Build Tools › Maven
Appearance & Behavior
Keymap
Editor
Plugins
Version Control
Build, Execution, Deployment
Build Tools
Maven
Gradle
Gant
Compiler
Debugger
Remote Jar Repositories
Android
Android Configurations
Build Analyzer
Coverage
Gradle-Android Compiler
Package Search
Required Plugins
Testing
Trusted Locations
Languages & Frameworks
Work offline
Use plugin registry
Execute goals recursively
Print exception stack traces
Always update snapshots
Output level: Info
Checksum policy: No Global Policy
Multiproject build fail policy: Default
Thread count
-T option
Maven home path: Bundled (Maven 3)
(Version: 3.8.1)
User settings file: /home/luis/.m2/settings.xml
Override
Local repository: /home/luis/.m2/repository
Override
Use settings from .mvn/maven.config
OK
Cancel
Apply