Build automation

- Strumenti che automatizzano task comuni nello sviluppo software, come
  - compilazione del sorgente, packaging dell'eseguibile, esecuzione dei test, rilascio dell’applicazione
- UNIX make
  - 1976, Stuart Feldman @ Bell Labs, pensato per lo sviluppo in C su UNIX
- Apache Ant
  - ~2000, James Duncan Davidson @ Sun, pensato per lo sviluppo Java (di Tomcat)
- Apache **Maven**
  - 2004, Apache Software Foundation, semplifica Ant e gestisce le dipendenze del progetto
- Gradle
  - 2007, uso di uno script Groovy, invece di un documento XML, per la configurazione
- …

Maven

- Supportato da tutti i principali IDE per Java
  - https://maven.apache.org/
- Per usarlo via CLI
  - https://maven.apache.org/download.cgi
    - Verifica installazione (version): mvn -v
- Creazione di un nuovo progetto
  - mvn -B archetype:generate -DarchetypeGroupId=org.apache.maven.archetypes -DgroupId=com.example -DartifactId=hello
  - Nel nuovo folder, nome artifactId
    - folder src per il codice sorgente per il progetto, main e test, Java e risorse aggiuntive
    - **pom.xml** (POM: Project Object Model)

Project Object Model

- I processi seguono convenzioni stabilite, solo le eccezioni vanno indicate
  - Ad esempio, la versione Java di default è la obsoleta 5
- Nel POM, all'interno dell'elemento project, specifichiamo le nostre variazioni
  - Properties
    - Costanti relative al POM
      - Charset utilizzato
      - Versione Java da usare
        - Per interpretare il codice sorgente
        - Per generare il bytecode
    - …
  - Dependencies
    - Implicano il download automatico delle librerie richieste

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

Aggiungere una dependency

- Ogni nuova dipendenza va in project, nell'elemento dependencies
- Occorre indicare groupId, artifactId e version
- Ricerca su repository Maven (central e altri)
  - https://search.maven.org/, https://mvnrepository.com/
- Esempio:
  - JUnit (4.xx) o JUnit Jupiter engine (5.x.x)

Passare a Jupiter implica refactoring

```
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.13</version>
  <scope>test</scope>
</dependency>
```

```
<dependency>
  <groupId>org.junit.jupiter</groupId>
  <artifactId>junit-jupiter-engine</artifactId>
  <version>5.7.1</version>
  <scope>test</scope>
</dependency>
```

Tra le <dependencies>

Diciamo a Maven che vogliamo usare JUnit solo in test

Compilazione e packaging

- Compilazione del progetto: mvn compile
  - I file risultanti vengono messi nel folder “target”
  - Esecuzione da target/classes:
    - java com.example.App
- Generazione di jar (war, …): mvn package
  - Esecuzione da target:
    - java -cp hello[…].jar com.example.App
    - java -jar hello[…].jar per i jar eseguibili 
- Per ripulire la build: mvn clean
  - Rimuove il folder “target”

Maven per executable jar

- In project – build – plugins
  - Configurazione ed esecuzione del plugin maven-assembly
  - Disabilitazione dell'esecuzione del plugin maven-jar

```
<artifactId>maven-assembly-plugin</artifactId>
<configuration>
   <descriptorRefs>
      <descriptorRef>jar-with-dependencies</descriptorRef>
   </descriptorRefs>
   <archive>
      <manifest><mainClass>com.example.App</mainClass></manifest>
   </archive>
</configuration>
<executions><execution>
   <id>executable-jar</id>
   <phase>package</phase>
   <goals><goal>single</goal></goals>
</execution></executions>
```

```
<artifactId>maven-jar-plugin</artifactId>
<version>3.2.0</version>
<executions>
   <execution>
      <id>default-jar</id>
      <phase>none</phase>
   </execution>
</executions>
```