UNIVERSITÀ DEGLI STUDI DELL'AQUILA

Laboratorio di Programmazione ad Oggetti

Ph.D. Juri Di Rocco
juri.dirocco@univaq.it
http://jdirocco.github.io

Sommario

- File jar
- Cosa è Maven
- Scaricare, installare e configurare
- Project Object Model (POM)
- Concetti base
- POM e super POM
- Struttura del progetto
- Esecuzione
- Dipendenze
- Repository
- Lifecycle di build, fasi e goal

Risorse

- Sito ufficiale
  - http://maven.apache.org
- Tutorial
  http://tutorials.jenkov.com/maven/maven-tutorial.html
- Libro di riferimento
  https://books.sonatype.com/mvnref-book/reference/index.html

Tool: jar (1)

- File con estensione `.jar` (Java Archive) indica un archivio dati compresso (ZIP) usato per distribuire raccolte di classi Java
- Viene utilizzato per fornire librerie e/o framework che non sono presenti all'interno di Java
- Si utilizza un jar sia in fase di compilazione che di esecuzione
- Compilazione (`cp` oppure `classpath`)

  ```
  javac -cp nomefilejar.jar Test.java
  ```

  Opzione `cp` viene utilizzata durante la compilazione per cercare le classi all'interno dei files jar
- Esecuzione

  ```
  java -cp nomefilejar.jar Test
  ```

  Opzione `cp` viene utilizzata durante l'esecuzione per cercare e caricare in memoria le classi necessarie
- Per creare un jar si utilizza il comando `jar`
- Documentazione https://docs.oracle.com/en/java/javase/11/tools/tools-and-command-reference.html

Tool: jar (1)

- File con estensione `.jar` (Java Archive) indica un archivio dati compresso (ZIP) usato per distribuire raccolte di classi Java
- Viene utilizzato per fornire librerie e/o framework che non sono presenti all'interno di Java
- Si utilizza un jar sia in fase di compilazione che di esecuzione
- **Compilazione** (`cp` oppure `classpath`)

  ```
  javac –cp nomefilejar.jar Test.java
  ```

  Opzione `cp` viene utilizzata durante la compilazione per cercare le classi all'interno dei files jar
- **Esecuzione**

  ```
  java –cp nomefilejar.jar Test
  ```

  Opzione `cp` viene utilizzata durante l'esecuzione per cercare e caricare in memoria le classi necessarie

Cosa è Maven (1)

› Per la grande maggioranza degli utenti, Maven è
  – Un tool utilizzato per costruire artefatti (package) partendo dal codice sorgente

› Gli ingegneri e i responsabili di progetto potrebbero riferirsi a
  – Maven come uno strumento di **gestione** del progetto

› Maven fornisce capacità di
  – pre-elaborazione,
  – compilazione,
  – packaging,
  – test e
  – deployment.

Cosa è Maven (2)

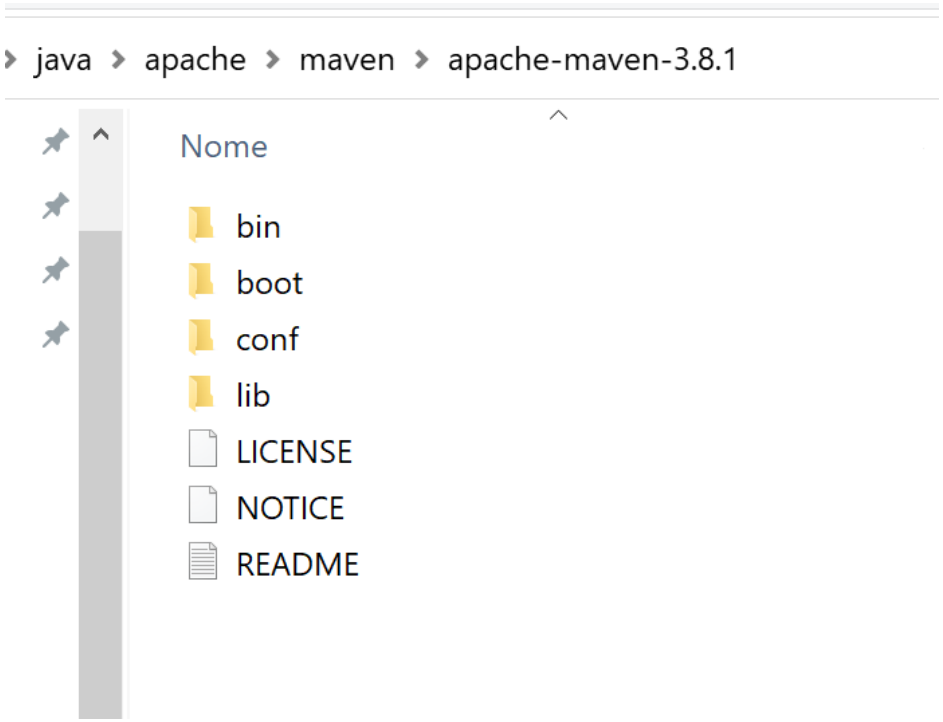
› Per il sito ufficiale di Apache Maven

› Maven è un tool di gestione del progetti che include
- Un modello ad oggetti del progetti (pom)
- Un insieme di standard
- Un ciclo di vita del progetto
- Un sistema di gestione delle dipendenze
- Una logica per eseguire plugins in specifiche fasi del ciclo di vita del progetto

Download e installazione (1)

- Download
  - https://maven.apache.org/download.cgi
  - Requisiti di sistema: JDK 1.7 o superiori
- Installazione
  - Assicurarsi che la variabile d'ambiente `JAVA_HOME` sia configurata
  - Estrarre il file scaricato in una qualsiasi directory
  - Aggiungere alla variabile d'ambiente `PATH` `$INSTALL_HOME/apache-maven-3.8.1/bin` (`apache-maven-3.8.1` dipende dalla versione scaricata)
  - `$INSTALL_HOME` è la cartella dove si è estratto il file
- Gli ambienti di sviluppo come Eclipse hanno già **installato** Maven al suo interno

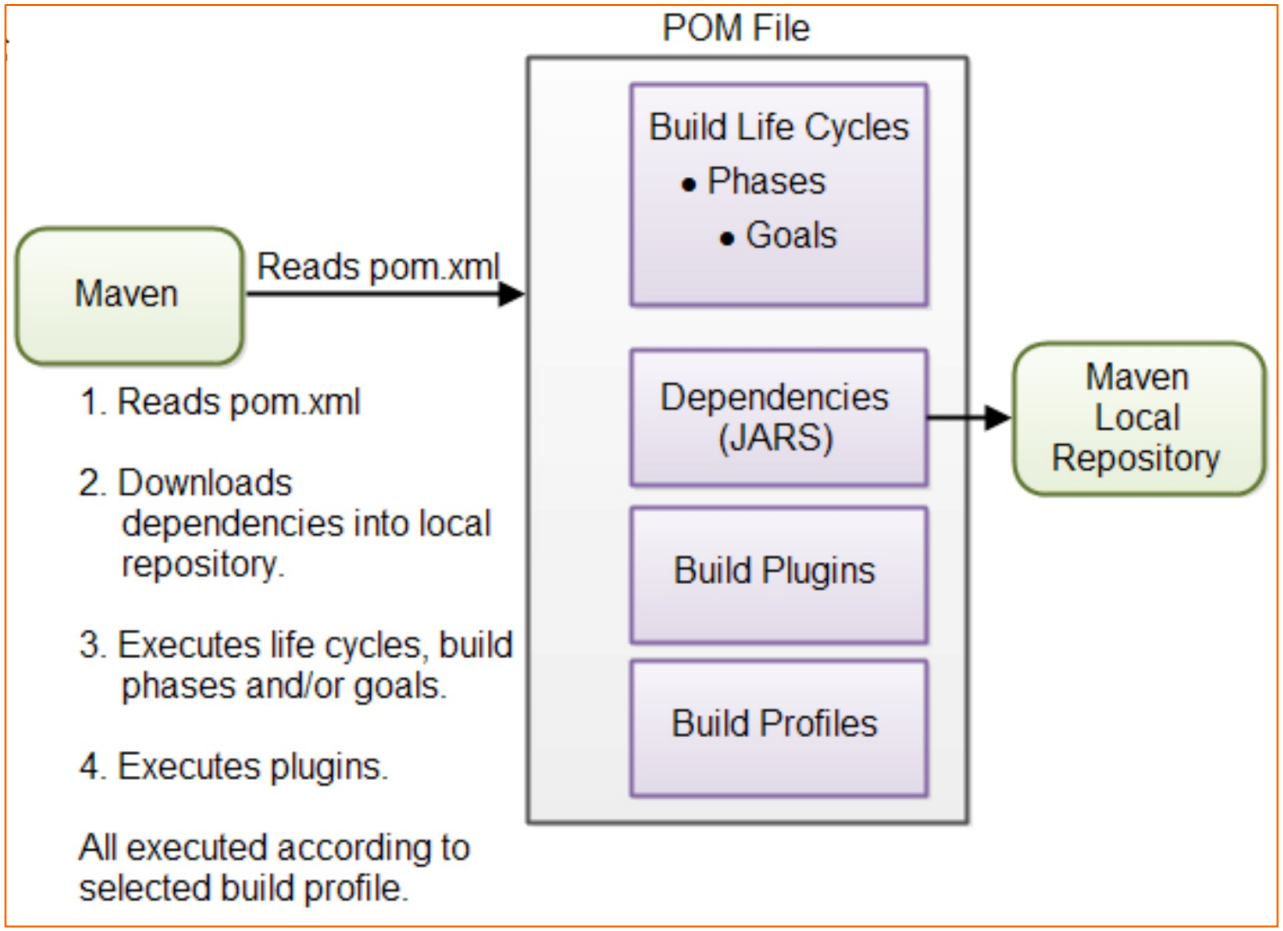
Download e installazione (2)



Project Object Model

- `pom.xml` è un file XML
- Contiene informazioni sul progetto, dipendenze, builds, profili, ecc.
- POM comunica a Maven
  - Il tipo del progetto (java o web)
  - Come modificare il comportamento di default per generare l'output a partire dal codice sorgente
- `pom.xml` è analogo ad un `Makefile` o a un file Ant `build.xml`

Concetti chiave



Lifecycles di build, fasi e goals

› Processo di build è suddiviso in **lifecycle** (cicli di vita), **fasi** e **goals**
  – Ogni lifecycle è composto di una sequenza di fasi
  – Ogni fase è composta di una sequenza di goals

› Maven viene eseguito passando come comando o il nome di un **lifecyle** o di una **fase** o di un **goal**
  – Se viene eseguita un lifecycle, **tutte** le fasi contenute all'interno di quel lifecycle vengono eseguite
  – Se viene eseguita una fase di build (che si trova all'interno di un lifecycle) vengono eseguite tutte le fasi **precedenti** e previste nel lifecycle

Dipendenze e repositories

- Maven effettua il controllo delle dipendenze necessarie al progetto
- Dipendenze sono file JAR (librerie Java) che il progetto utilizza
- Se le dipendenze non sono trovate nel repository locale, Maven le scarica da un repository centrale e le pone in quello locale
- E' possibile specificare quale repository remoto utilizzare per scaricare le dipendenze

Build Plugins

- Sono utilizzati per inserire goals extra durante la fase di build
- Sono utili per eseguire un insieme di azioni che non sono previste dalle fasi standard di Maven
  - Esempio: generare stub e skeleton Java partendo dal WSDL
- Vengono aggiunti nel file POM
- Maven ha plugins standard che possono esere usati
- Se ne possono implementare di propri

Build Profiles

- Sono utilizzati per effettuare il build del progetto in diversi ambienti
- Esempio
  - Potrebbe essere necessario effettuare il build del progetto per il proprio computer oppure per l'ambiente di test oppure quello di produzione
- Per abilitare i differenti build è necessario specificarli all'interno del POM
- All'atto dell'esecuzione del commando maven viene scelto il relativo profilo

Struttura progetto Maven

```
- src
  - main
    - java
    - resources
    - webapp
  - test
    - java
    - resources
-target
pom.xml
```

- `src`: cartella di root per il codice sorgente e codice di test
- `main`: cartella di root per il codice sorgente
- `test`: cartella di root per il codice di test
- `java` sotto `main` e `test`: codice Java dell'applicazione e del test rispettivamente
- `resources`: risorse utili al progetto (esempio, file configurazione, ecc.)
- `webapp`: risorse per applicazioni web in Java (opzionali)
- `target`: contiene le classi compilate, file jars prodotte da Maven

Pom file

- Descrive le risorse del progetto
  - Codice sorgente, codice di test, ecc.
  - Dipendenze esterne (files JAR)
- Descrive cosa costruire e non come
  - Come è determinate dalle fasi e goals di Maven
- Un progetto composto da sottoprogetti ha tipicamente un POM per il progetto padre e un POM per ogni sottoprogetto
  - Permette di effettuare il build del progetto in un passo oppure per ogni sottoprogetto di fare il build separamente

POM minimale

```
<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>it.univaq.disim.oop</groupId>
    <artifactId>myunivaq</artifactId>
    <version>1.0.0</version>

</project>
```

Elementi del POM (1)

- `modelVersion` determina la versione del modello del POM utilizzato
  - Versione 4.0.0 per Maven
- `groupId`: ID unico per una organizzazione o per un progetto
  - Tipicamente è simile alla *root* del package java del progetto
  - Se il progetto è open source generalmente il group ID è relativo al progetto
  - Viene usato per creare le cartelle all'interno del repository locale dove il "." è usato come separatore
  - Esempio

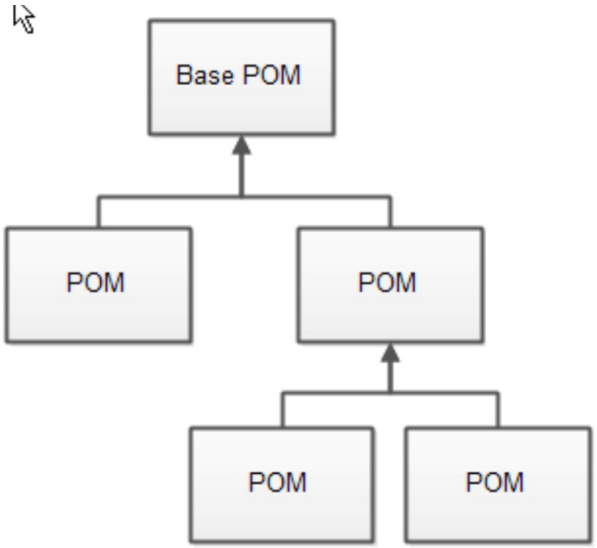
  `it.univaq.disim.oop` → cartella `it/univaq/disim/oop`

Elementi del POM (2)

- `artifactId`: contiene il nome del progetto che si sta costruendo
  - Utilizzato come nome della sottodirectory sotto la directory associata al group ID nel repository
  - Uilizzato anche come nome dell'artefatto (es. JAR) prodotto
- `versionId`: contiene il numero di versione del progetto
  - Utilizzato per riferirsi ad una versione specifica del progetto
  - Utilizzato come nome di una sottodirectory all'interno della sottodirectory relativa all'artifact ID
  - Utilizzato anche nel nome dell'artefatto (es. JAR) prodotto
- `groupId`, `artifactId` e `versionId` compongono il nome dell'artefatto
- Esempio relativo al POM precedente
  `$MAVEN_REPO/it/univaq/disim/oop/my-univaq/1.0.0/myunivaq-1.0.0.jar`

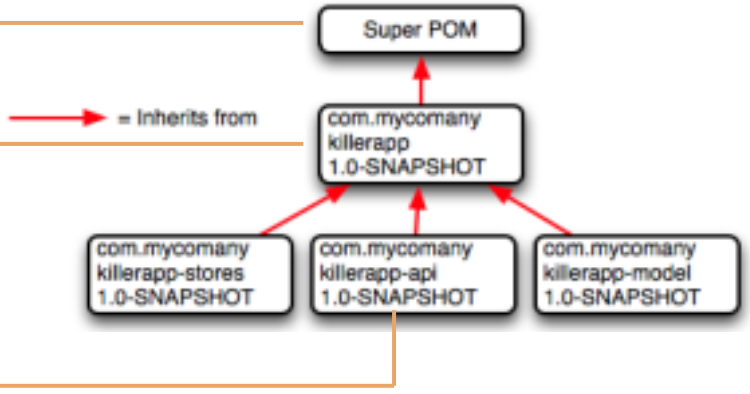
Super POM (1)

- Tutti i progetti Maven estendono da un Super POM, che definisce una serie di proprietà condivise
- Esempio
  - Struttura del progetto
- Fa parte dell'installazione Maven
- E' simile alla classe `java.lang.Object` per Java



Super POM (2)

```
<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>
    <parent>
        <groupId>com.mycomany</groupId>
        <artifactId>killerapp</artifactId>
        <version>1.0-SNAPSHOT</version>
        <relativePath>../pom.xml</relativePath>
    </parent>
    <artifactId>killerapp-api</artifactId>
    <version>1.0-SNAPSHOT</version>
</project>
```



Un POM *figlio* può sovrascrivere delle proprietà presenti nel super POM

Eseguire Maven

- All'interno della root del progetto

```
$ mvn command
```

- Command contiene il nome di un lifecycle di build, fase o goal che Maven deve eseguire
- Esempio

```
$ mvn package
```

  - Esegue la fase detta `package` (che fa parte del lifecycle di build di **default**)
  - Esegue tutte le fasi di build prima di `package` e poi la esegue
  - Risultato di `package` è compilare e creare il file JAR

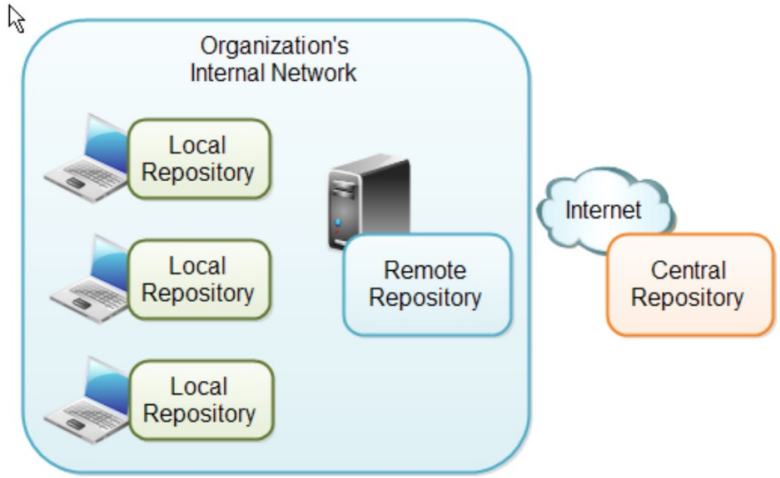
Dipendenze

- E' possible specificare nel POM librerie esterne (con la versione)
- Maven scarica tali librerie e le pone nel repository locale
- Se tale libreria necessita di altre, anche queste vengono scaricate (dipendenza transitiva)

```
<dependencies>
        <dependency>
                <groupId>org.openjfx</groupId>
                <artifactId>javafx-controls</artifactId>
                <version>14</version>
        </dependency>
        <dependency>
                <groupId>org.openjfx</groupId>
                <artifactId>javafx-fxml</artifactId>
                <version>14</version>
        </dependency>
</dependencies>
```

Repositories Maven

- Sono directory che contengono files JAR con dati extra (metadata)
- I metadata sono files POM che descrivono i progetti con le relative dipendenze esterne
- Tali metadata permettono di scaricare le dipendenze in modo ricorsivo e porle nel repository locale
- Tre tipi
  - Repository Locale
  - Repository Centrale
  - Repository Remoto
- Maven cerca all'interno di tali repository nella sequenza sopra descritta



Repository Locale (1)

› E' una directory che si trova sul proprio computer

› Nome di default della cartella `repository`

› Contiene tutte le dipendenze che Maven scarica

› Per default Maven pone tale cartella all'interno

`$USER_HOME/.m2/`

Mio caso `/Users/juri/.m2/repository`

› E' possible porre all'interno del repository locale i propri progetti se si usa il commando

```
$ mvn install
```

Repository Centrale

- E' un repository fornito dalla comunità di Maven
- Per default Maven cerca nel repository centrale le dipendenze necessarie che non sono state trovate in quello locale
- `https://mvnrepository.com/repos/central`
- `http://central.maven.org/maven2/`

Repository Remoto

- E' un repository in rete dal quale Maven può scaricare le dipendenze
- Può essere localizzato ovunque su internet oppure sulla rete dell'organizzazione per cui si lavora
- Repository remoto contiene spesso progetti utilizzati internamente ad una organizzazione che sono condivisi
- Per utilizzare un repository remoto è necessario porre nel file `pom.xml` subito dopo le dipendenze (`<dependencies>`)

```
<repositories>
 <repository>
  <id>disim.code</id>
  <url>https://oss.sonatype.org/content/repositories/releases/</url>
 </repository>
</repositories>
```

Properties

› Le proprietà sono l'ultimo elemento necessario per comprendere le basi del POM. Le proprietà Maven sono segnaposto di valore, come le proprietà in Ant.

```
<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>org.mdeforge</groupId>
        <artifactId>mdeforge</artifactId>
        <packaging>war</packaging>
        <version>1.0-SNAPSHOT</version>
        <name>mdeforge</name>
        <url>https://github.com/MDEGroup/MDEForge</url>
        <properties>
                <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
                <spring.version>4.0.6</spring.version>
                <lucene.version>6.5.0</lucene.version>
        </properties>
```

Properties (2)

› I relativi valori sono accessibili ovunque all'interno di un POM utilizzando la notazione `${X}`, dove X è la proprietà. Oppure possono essere utilizzati dai plugin come valori predefiniti, ad esempio:

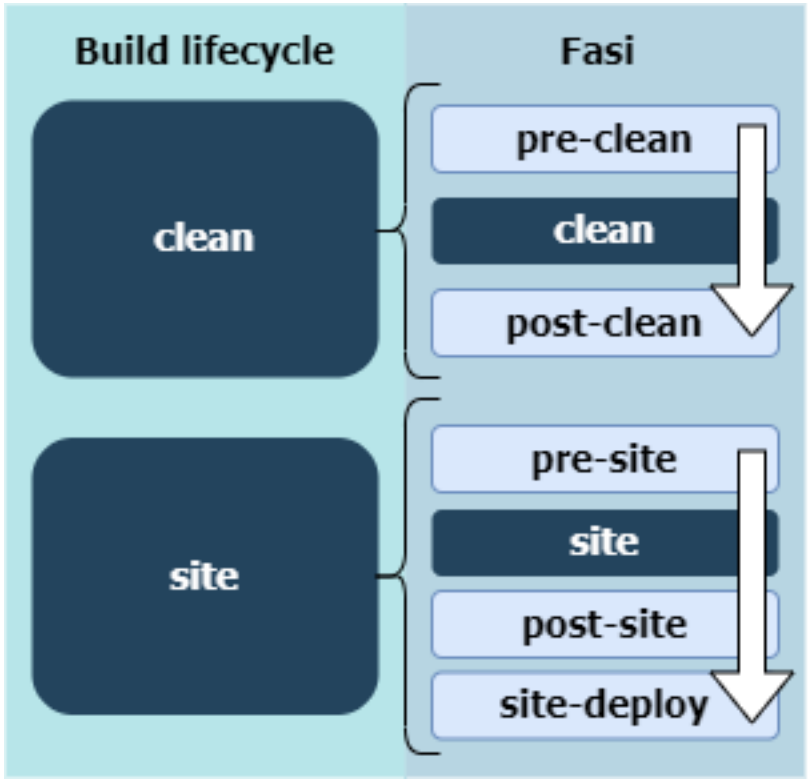
```
<dependency>
        <groupId>org.apache.lucene</groupId>
        <artifactId>lucene-core</artifactId>
        <version>${lucene.version}</version>
</dependency>
<dependency>
        <groupId>org.apache.lucene</groupId>
        <artifactId>lucene-queryparser</artifactId>
        <version>${lucene.version}</version>
</dependency>
```

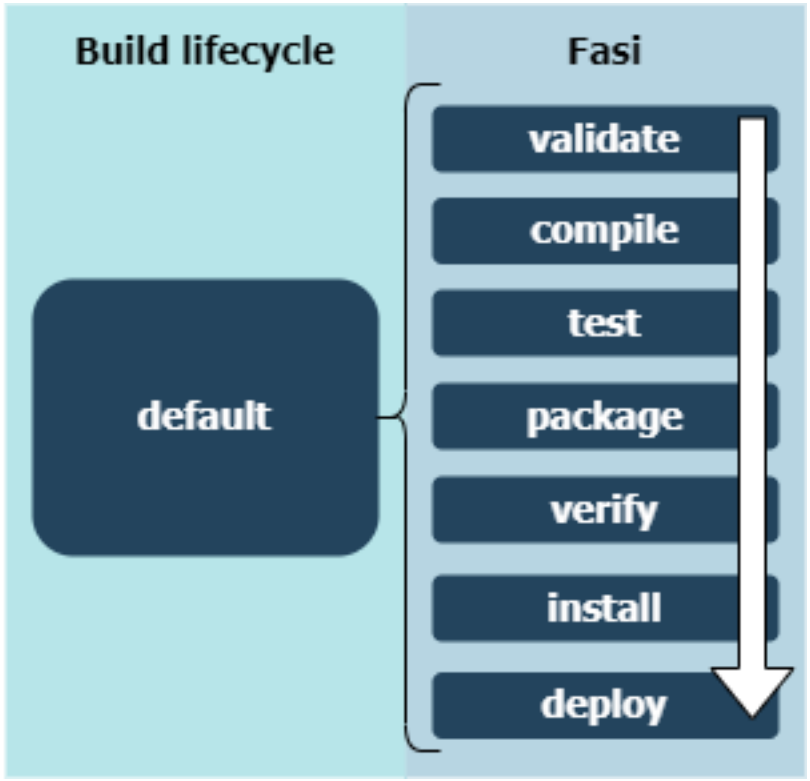
Lifecycles di build, fasi e goals (1)

- Quando Maven costruisce un progetto segue un lifecycle di build
- Come detto in precedenza un lifecyle è diviso in **fasi** di build e ogni fase è divisa in **goal**
- Sono presenti tre lifecycle di build predefiniti
  - `default`: utilizzato per compilare e costruire l'artefatto del progetto
  - `clean`: utilizzato per rimuovere i file temporanei dalla cartella di output
  - `site`: utilizzato per generare la documentazione del progetti (poco usato)
- Ognuno di questi lifecycle sono eseguiti indipendentemente uno dall'altro
- E' possible eseguire più di un lifecycle di build ma devono essere eseguiti in sequenze e indipendentemente uno dall'altro

```
$ mvn clean compile
```

Lifecycles di build, fasi e goals (2)





Lifecycles di build, fasi e goals (3)

- E' possibile eseguire un intero lifecycle di build (esempio `clean` o `site`), un fase di build come `compile` che è parte del lifecycle di build di `default` oppure uno specifico goal come `dependency:copy-dependencies`
- **Non è possibile** eseguire il lifecycle di `default` direttamente
  - E' necessario specificare una **fase** di build o **goal**
- Quando viene eseguita una fase di build tutte le fasi precedenti nella sequenza standard sono eseguite

Lifecycles di build, fasi e goals (4)

› Fasi di build più comuni del lifecycle di `default`

- `validate`: Verifica che il progetto è corretto e tutte le informazioni sono disponibili. Si assicura che anche tutte le dipendenze sono scaricate
- `compile`: Compila il codice sorgente del progetto
- `test`: Esegue i test (se specificati) utilizzando un particolare framewok di test
- `package`: Impacchetta il codice compilato in un formato distribuibile (esempio file JAR)
- `install`: Installa il package nel repository locale per essere utilizzato come dipendenza in altri progetti locali
- `deploy`: Copia il package finale al repository remoto per condividerlo con gli altri sviluppatori

Lifecycles di build, fasi e goals (5)