



Maven

Software project management and comprehension tool





Roadmap



Maven
basics



Maven
coding





Project

Un progetto è un insieme ben definito di attività in modo che:

- ha un inizio
- ha una fine
- realizza un obiettivo
- è realizzato da un insieme di persone
- utilizza un certo insieme di risorse



Project Management

Insieme delle attività necessarie ad assicurare che un prodotto software sia sviluppato rispettando le scadenze fissate e risponda a determinati standard

Interazione di aspetti economici e tecnici

**“Un progetto diretto bene qualche volta fallisce ...
... uno diretto male fallisce sicuramente”**

L'importanza dell'esperienza



Alcuni vantaggi derivanti dal Project Management

Distribuzione organizzata del lavoro su varie persone

Gestire in modo organizzato i **cambiamenti** dei requisiti

Valutazione del **rischio**:

- A che punto del progetto mi trovo?
- Se devo aggiungere una funzionalità posso farlo rispettando budget e deadline?

Gestione organizzata:

- Dell'alternanza di persone nel team
- Risoluzione dipendenze in base ad una strutturazione dichiarativa del progetto
- Scaffolding predefinito



Maven Framework

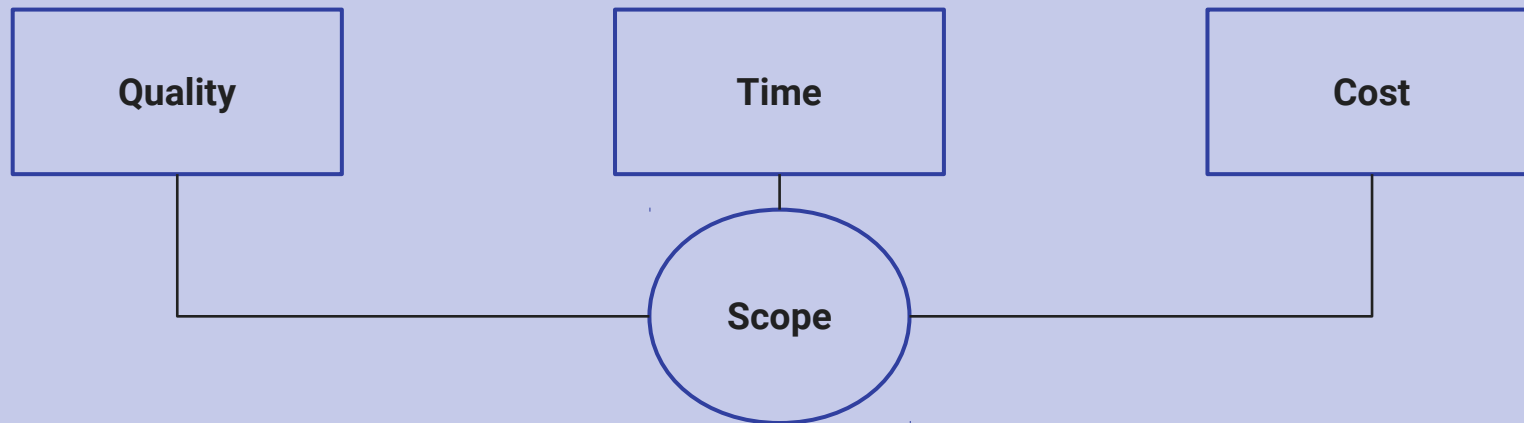
Maven è un framework per la **gestione** e la **comprensione** di progetti

Applica **pattern** ben collaudati all'infrastruttura del **build** dei progetti

Promuove l'utilizzo di **best practice**

Maven nacque nell'ambito del progetto **Jakarta Alexandria** (abbandonato)

Attualmente è migrato nel progetto **Turbine** (framework per la produzione di Web Apps).



Caratteristiche principali:

- Semplificazione del processo di build

- Fornire un sistema di build uniforme

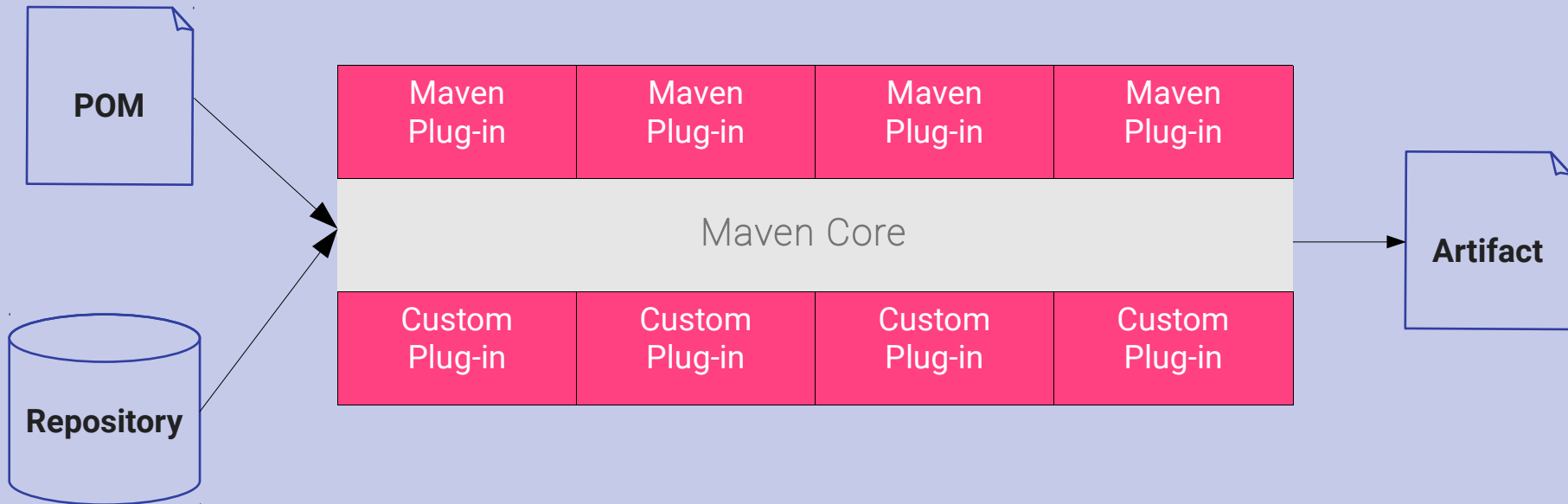
- Fornire informazioni di qualità sul progetto

- Fornire delle linee guida per le “best practices” di sviluppo

- Permettere l'introduzione di nuove funzionalità con basso effort



Componenti principali





POM Project Object Model:

Documento xml

Contiene una descrizione dichiarativa del progetto(meta-dati)

Include varie sezioni, al fine di assicurare una corretta:

1. Gestione della fase di **build**,
2. Gestione delle **dipendenze**,
3. Gestione dei **test**,
4. Generazione della **documentazione**

Goal:

Singola funzione che può essere eseguita sul progetto,

Plug-in:

Funzioni riutilizzabili e cross-project

Agiscono sul progetto utilizzando dei **goal**



Maven coding



Java

Maven

Eclipse

Java Setup

Scaricare ed installare Java Development Kit dal seguente URL:

<http://www.oracle.com/technetwork/java/javase/downloads/>

Dopo aver effettuato l'installazione, bisogna controllare che sia **Java** sia il **Java Compiler** siano stati aggiunti correttamente alle **variabili d'ambiente** relative al sistema operativo che si sta utilizzando.

In caso non lo siano, bisogna aggiungerli manualmente.



Java

Maven

Eclipse

Per verificare se Java è stato riconosciuto correttamente, basta aprire una shell e digitare il seguente comando:

java -version

```
[ivan18@localhost ~]$ java -version
java version "1.8.0_77"
Java(TM) SE Runtime Environment (build 1.8.0_77-b03)
Java HotSpot(TM) 64-Bit Server VM (build 25.77-b03, mixed mode)
```

Per verificare se il Java Compiler è stato riconosciuto correttamente, basta aprire una shell e digitare il seguente comando:

javac -version

```
[ivan18@localhost ~]$ javac -version
javac 1.8.0_77
```



Maven download

Prima di iniziare abbiamo bisogno di scaricare i package necessari per il funzionamento di Maven dal seguente link:

<https://maven.apache.org/download.cgi>

Scaricate le librerie, estraiamone il contenuto in una directory

Quest'ultima dovrà essere aggiunta alle **variabili d'ambiente**



Per verificare se Maven è stato riconosciuto correttamente, basta aprire una shell e digitare il seguente comando:

mvn --version

```
[ivan18@localhost ~]$ mvn --version
Apache Maven 3.3.9 (bb52d8502b132ec0a5a3f4c09453c07478323dc5; 2015-11-10T17:41:47+01:00)
Maven home: /home/ivan18/development/maven/versions/default
Java version: 1.8.0_77, vendor: Oracle Corporation
Java home: /usr/java/jdk1.8.0_77/jre
Default locale: it_IT, platform encoding: UTF-8
OS name: "linux", version: "4.5.7-202.fc23.x86_64", arch: "amd64", family: "unix"
```



Eclipse Java IDE

Ricordate che il codice relativo ad un qualsiasi linguaggio di programmazione può essere scritto tramite l'utilizzo di un qualsiasi **editor di testo**

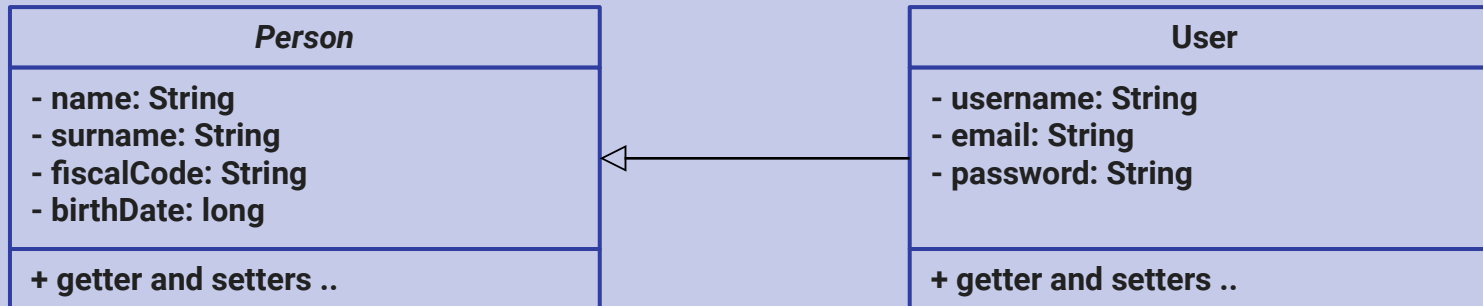
Una volta ottenuti i file con il codice sorgente, questi vanno **compilati** ed **eseguiti** manualmente tramite shell

Per facilitare e velocizzare la scrittura di software Java utilizzeremo **Eclipse**, un apposito **IDE** (Integrated development environment) scaricabile dal seguente link:

<https://www.eclipse.org/ide/>



Caso di studio





Project

Scaffolding

Implementation

Prima di creare il progetto è bene stabilirne alcune proprietà di base:

- 1 - **groupId** id relativo alla tipologia (gruppo) del progetto
- 2 - **artifactId** id del progetto generalmente rappresentato dal suo nome
- 3 - **version** versione del progetto

La **naming convention** che generalmente si usa per identificare un progetto è del tipo:

groupId:artifactId:version



Creazione progetto

Maven usufruisce di plugin opportunamente predisposti per la creazione di progetti

Questa tipologia di plugin viene denominata con il termine **archetype**

Useremo quindi il plugin **maven-archetype-quickstart** per inizializzare il progetto

`mvn archetype:generate`

`-DgroupId=uniroma2.is`

`-DartifactId=maven`

`-DarchetypeArtifactId=maven-archetype-quickstart`

`-DinteractiveMode=false`

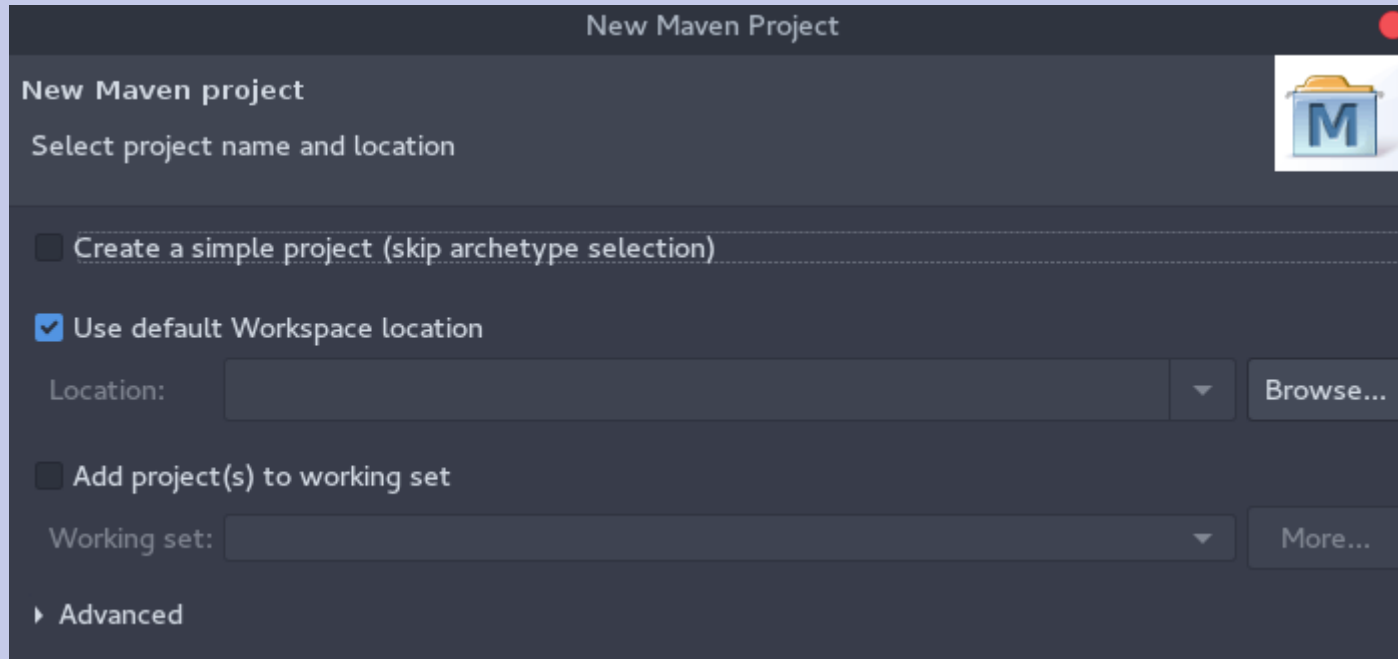


Project

Scaffolding

Implementation

Esempio creazione progetto tramite eclipse:



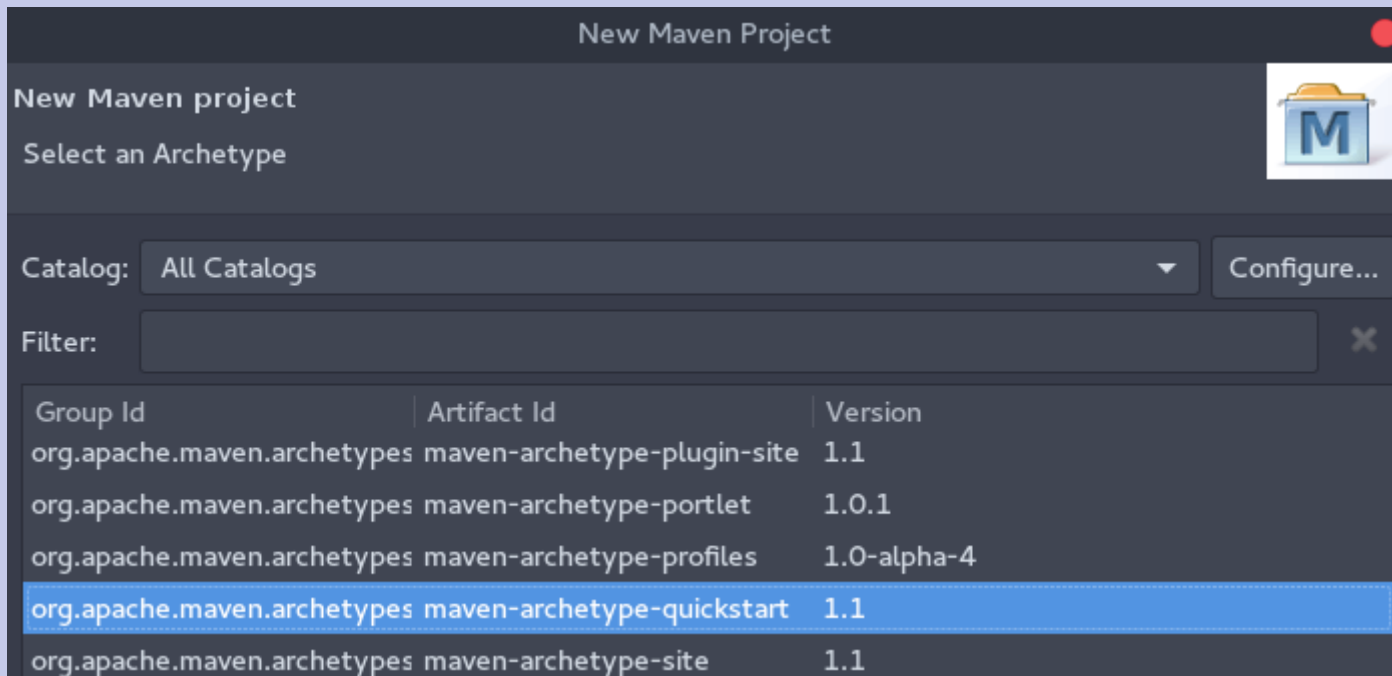


Project

Scaffolding

Implementation

Selezioniamo l'archetype opportuno e clicchiamo su avanti





Project

Scaffolding

Implementation

Inseriamo le proprietà di base individuate in precedenza e confermiamo

New Maven Project

New Maven project

Specify Archetype parameters

Group Id: uniroma2.is

Artifact Id: maven

Version: 0.0.1-SNAPSH

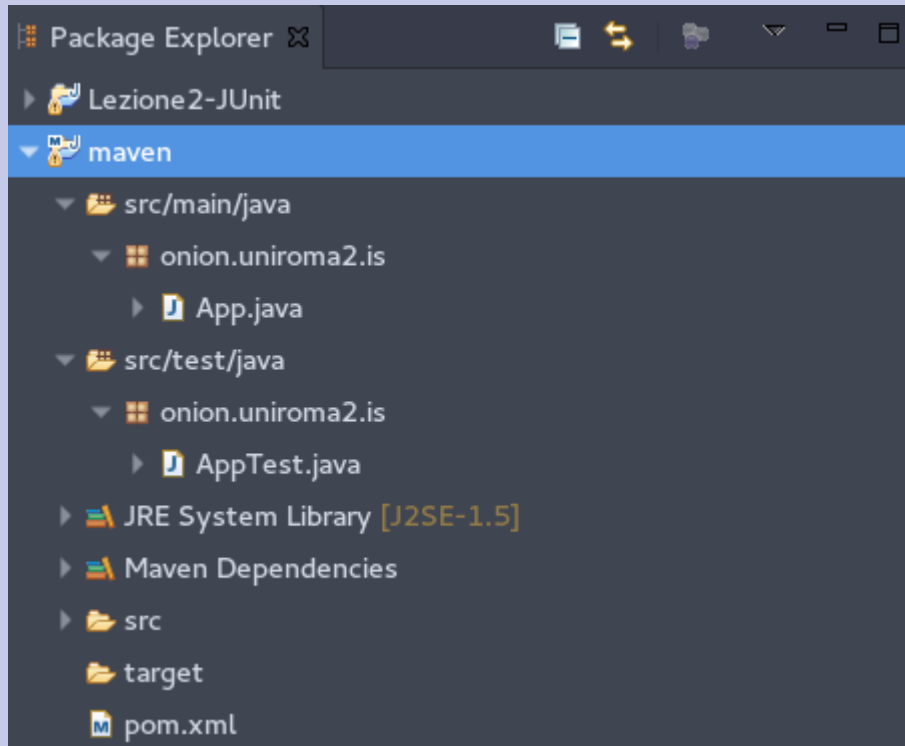
Package: uniroma2.is.maven



Project

Scaffolding

Implementation



Nell'area **PackageExplorer** possiamo notare che l'esecuzione del comando precedente ha correttamente inizializzato il progetto

Oltre al **pom.xml** sono state create 2 classi:

- 1 - App.java
- 2 - AppTest.java



Project

Scaffolding

Implementation

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>onion.uniroma2.is</groupId>
  <artifactId>maven</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>maven</name>
  <url>http://maven.apache.org</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```



Project

Scaffolding

Implementation

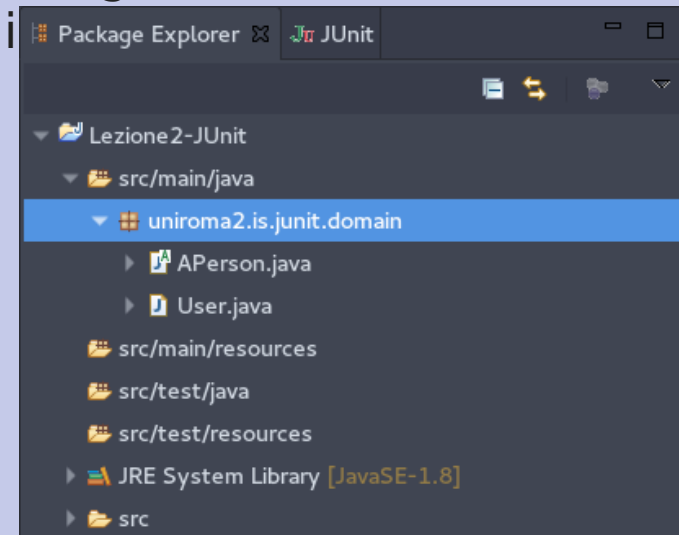
Implementazione caso di studio

Procediamo con l'implementazione del diagramma delle classi mostrato in precedenza

Creiamo ne package src/main/java un nuovo package che andrà a contenere le classi di dominio individuate nel caso di studio

1 - APerson (classe astratta)

2 - User (classe concreta)





Project

Scaffolding

Implementation

JUnit

Per aggiungere la libreria JUnit al nostro progetto, colleghiamoci al seguente link:

<https://github.com/junit-team/junit4/wiki/Download-and-Install>

Nella sezione Maven notiamo il seguente **snippet**:

```
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.12</version>
  <scope>test</scope>
</dependency>
```

Copiamolo ed inseriamolo nel pom.xml sotto la sezione **dependencies**



Compile

Test

Package

Clean

Install

Site

Compilare i sorgenti

Per effettuare la compilazione dei sorgenti, Maven mette a disposizione l'opzione:

```
mvn compile
```

L'esecuzione del comando farà sì che maven scaricherà da opportuni repository tutti i plugin necessari alla corretta esecuzione del comando stesso

Gli output (classi compilate) verranno inserite nel path

```
${basedir}/target/classes
```



Esempi



Compile

Test

Package

Clean

Install

Site

Esecuzione di test

Per eseguire il test è necessario compilarli ed eseguirli, ciò può essere fatto usando l'opzione test:

```
mvn test
```

Nel caso in cui si vogliano solamente compilare i sorgenti di test si utilizzerà:

```
mvn test-compile
```

Nota:

Maven scaricherà più dipendenze -> necessarie ad eseguire i test

Prima di compilare ed eseguire i test Maven compilerà interamente il codice in modo tale da assicurare l'integrità dell'applicativo che si sta sviluppando



Esempi



Compile

Test

Package

Clean

Install

Site

Pacchettizzare l'applicativo

Compilati i sorgenti dell'applicazione è possibile pacchettizzarli secondo il linguaggio di programmazione utilizzato

Nel nostro caso, nel pom.xml, è stato settato l'elemento **packaging** su **jar**

Tramite l'opzione:

```
mvn package
```

Maven analizzerà il progetto e produrrà la sua pacchettizzazione nel path:

```
${basedir}/target
```



Esempi



Compile

Test

Package

Clean

Install

Site

Clean

Il plugin clean permette di cancellare i compilati da maven dal progetto

`mvn clean`

Come output di questo goal, ci si aspetta di avere completamente vuota, la directory:

`${basedir}/target`



Esempi



Compile

Test

Package

Clean

Install

Site

Install

Permette di depositare il pacchetto generato nel repository locale.

`mvn install`

Se l'operazione va a buon fine, andando nel repository locale:

`${user_home}/.m2/repository`

Dovrebbe essere stata creata una directory con il nostro **groupid**

Quando deposito una nuova build in un repository repository altri developer possono subito recuperarla ed utilizzarla



Esempi



Compile

Test

Package

Clean

Install

Site

Reporting

Un aspetto da non trascurare quando si sviluppano sistemi software è la produzione di **documentazione**

Possiamo generare automaticamente documentazione dal nostro codice utilizzando il plugin **site** in combinazione con vari plugin:

1. Maven-**javadoc**-plugin:
<https://maven.apache.org/plugins/maven-javadoc-plugin/usage.html>
2. Maven-**surefire-report**-plugin:
<http://maven.apache.org/surefire/maven-surefire-report-plugin/usage.html>



Compile

Test

Package

Clean

Install

Site

Maven-javadoc-plugin

Questo plugin consente di generare documentazione java sfruttando il tool Javadoc.

Per poterlo utilizzare, dobbiamo informare maven della disponibilità di questo strumento in fase di generazione della reportistica.

Nel pom.xml aggiungiamo quindi

```
<reporting>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-javadoc-plugin</artifactId>
      <version>2.10.4</version>
    </plugin>
  </plugins>
</reporting>
```



Compile

Test

Package

Clean

Install

Site

Maven-surefire-report-plugin

Questo plugin consente di generare documentazione prendendo come input i risultati ottenuti dall'invocazione del plugin maven-surefire visto in precedenza (vedi esecuzione test).

Genera da tali file una serie di pagine HTML atte a dare una panoramica generale del software

Per poterlo utilizzare, dobbiamo anche in questo caso, informare maven della disponibilità di questo strumento in fase di generazione della reportistica.

Nel pom.xml aggiungiamo quindi sotto la sezione `<reporting>`

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-surefire-report-plugin</artifactId>
  <version>2.20</version>
</plugin>
```



Compile

Test

Package

Clean

Install

Site

Site

Aggiunti i nuovi plugin alla sezione di reportistica, possiamo invocare:

`maven site`

Quest'ultimo invocherà i plugin presenti nella sezione reporting.

L'esecuzione del maven-javadoc-plugin genererà la classica documentazione java nel path: `${basedir}/target/site/apidocs`

L'esecuzione del maven-surefire-report-plugin genererà un mini sito web accessibile tramite il file: `${basedir}/target/site/index.html`

maven

Last Published: 2017-05-30 | Version: 0.0.1-SNAPSHOT

maven

Project Documentation

▼ Project Information

Dependencies

Dependency

Convergence

Dependency Information

About


Plugin Management

Plugins


Summary

► Project Reports

Built by:



Project Information

This document provides an overview of the various documents and links that are part of this project's general information. All of this content is automatically generated by [Maven](#)  on behalf of the project.

Overview

| Document | Description |
|--|---|
| Dependencies | This document lists the project's dependencies and provides information on each dependency. |
| Dependency Convergence | This document presents the convergence of dependency versions across the entire project, and its sub modules. |
| Dependency Information | This document describes how to to include this project as a dependency using various dependency management tools. |
| About | There is currently no description associated with this project. |
| Plugin Management | This document lists the plugins that are defined through pluginManagement. |
| Plugins | This document lists the build plugins and the report plugins used by this project. |
| Summary | This document lists other related information of this project |



Compile

Test

Package

Clean

Install

Site

Project Documentation

- Project Information
- Project Reports
 - JavaDocs
 - Test JavaDocs
 - Surefire Report**



Surefire Report

Summary

[\[Summary\]](#) [\[Package List\]](#) [\[Test Cases\]](#)

| Tests | Errors | Failures | Skipped | Success Rate | Time |
|-------|--------|----------|---------|--------------|-------|
| 93 | 3 | 40 | 6 | 47.312% | 9.043 |

Note: failures are anticipated and checked for with assertions while errors are unanticipated.

Package List

[\[Summary\]](#) [\[Package List\]](#) [\[Test Cases\]](#)

| Package | Tests | Errors | Failures | Skipped | Success Rate | Time |
|--|-------|--------|----------|---------|--------------|-------|
| uniroma2.is.maven | 1 | 0 | 0 | 0 | 100% | 0.005 |
| uniroma2.is.maven.controller | 4 | 0 | 2 | 0 | 50% | 0 |
| uniroma2.is.maven.service | 12 | 0 | 6 | 0 | 50% | 0.006 |
| uniroma2.is.maven.domain | 76 | 3 | 32 | 6 | 46.053% | 9.032 |

Note: package statistics are not computed recursively, they only sum up all of its testsuites numbers.



Esempi

Grazie per l'attenzione



Ivan Bruno