Apache Maven

What is Apache Maven

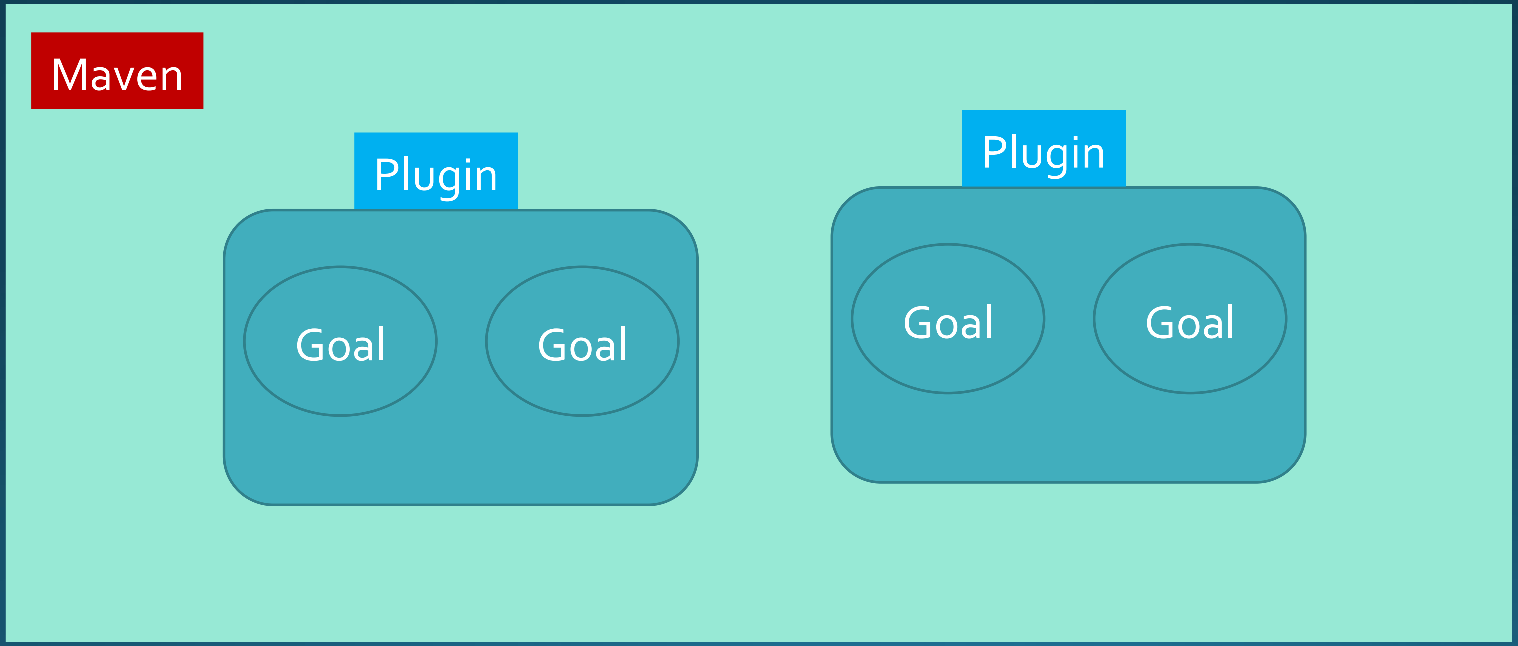
- Apache Maven is a project management and build automation tool.
- You use maven to create and manage your projects.
- It builds and deploys the projects as well.

Why Use Maven?

- **Simple project setup that follows best practices**
  - Maven tries to avoid as much configuration as possible, by supplying project templates (named *archetypes*)
- **Dependency management**
  - it includes automatic updating, downloading and validating the compatibility, as well as reporting the dependency closures (known also as transitive dependencies)
- **Isolation between project dependencies and plugins**
  - with Maven, project dependencies are retrieved from the *dependency repositories* while any plugin's dependencies are retrieved from the *plugin repositories*, resulting in fewer conflicts when plugins start to download additional dependencies
- **Central repository system**
  - project dependencies can be loaded from the local file system or public repositories, such as **Maven Central**

Architecture of Maven

- Maven does not do all the things itself other than validating and parsing the configuration XML file named ‘pom.xml’.
- Maven uses **Plugins** to perform all the tasks for building the project



Project Object Management(POM)

- POM is the fundamental unit of work in Maven
- Maven projects are configured using an XML file named ‘pom.xml’ called ‘Project Object Management (POM)’ file.
- The *POM* describes the project, manages dependencies, and configures plugins for building the software.

A Sample POM

```
<project>
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.demo</groupId>
    <artifactId>simple-app</artifactId>
    <packaging>jar</packaging>
    <version>1.0-SNAPSHOT</version>
    <name>simple-app</name>
        <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
            <scope>test</scope>
        </dependency>
    </dependencies>
    <build>
        <plugins>
            <plugin>
            //...
            </plugin>
        </plugins>
    </build>
</project>
```

Super POM

- The Super POM is Maven's default POM.
- All POMs extend the Super POM unless explicitly set, meaning the configuration specified in the Super POM is inherited by the POMs you created for your projects

Maven Co-Ordinates

- Maven uses a set of identifiers to identify a project uniquely and to manage the dependencies.
- These identifiers are known as ‘Maven Co ordinates’
- **groupId**
  - A unique base name of the project creator organisation.
  - Organisation domain is used more frequently
- **artifactId**
  - A unique name given to the project
- **version**
  - The project release version
- **packaging**
  - How the build shoud be packaged e.g. JAR,WAR,EAR etc.

Maven Coordinates

groupId:artifactId:version

forms the unique Identifier

```
<dependency>             <groupId>org.springframework</
groupId>             <artifactId>spring-context </artifactId>
    <version>5.1.2.RELEASE</version>
</dependency>
```

Dependency Management

- Every project uses some dependency libraries (e.g JAR File).
- Maven’s dependency management feature downloads libraries(jars) from central Repository to satisfy project’s needed dependency.
- Once downloaded, the library jars are cached locally in <user_home>/.m2 folder.
- The downloaded library is reused for other projects.
- You don’t have to store them locally

Dependency Management

- You declare the dependency as follows:

```
<dependency>              <groupId>org.springframework</
groupId>              <artifactId>spring-context </artifactId>
      <version>5.1.2.RELEASE</version>
</dependency>
```

The library jar and its dependency will be downloaded and cached in <USER_HOME/.m2

Maven Repositories

- Maven stores build artifacts and dependencies of varying types in a repository.
- The default local repository is located in the *.m2/repository* folder under the home directory of the user.
- If an artifact or a plug-in is available in the local repository, Maven uses it. Otherwise, it is downloaded from a central repository and stored in the local repository. The default central repository is Maven Central

Third Party Repositories

- Some libraries, e.g. those from Oracle, JBoss etc are not available in the central repository.
- The companies maintain their own public repository.
- For example, you can configure such repositories as follows :

Repository URL

```
<repositories>
    <repository>
        <id>JBoss repository</id>
        <url>http://repository.jboss.org/nexus/content/groups/public/</url>
    </repository>
</repositories>
```

Maven Properties

- Properties can help to make your *pom.xml* file easier to read and maintain.
- Developer generally defines one or more custom properties in pom.xml
- **Maven properties are value-placeholders and are accessible anywhere within a *pom.xml* by using the notation *${name}*,** where *name* is the property.

```
<properties>
    <spring.version>5.1.4.RELEASE</
spring.version>
</properties>

<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-core</artifactId>
        <version>${spring.version}</version>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>${spring.version}</version>
    </dependency>
</dependencies>
```

Maven Build Lifecycles

Maven Build Lifecycle

- Every Maven build follows a specified *lifecycle*.
- You can execute several build *lifecycle goals e.g.*
  - *compile* the project's code
  - create a *package*
  - *install* the archive file in the local Maven dependency repository.

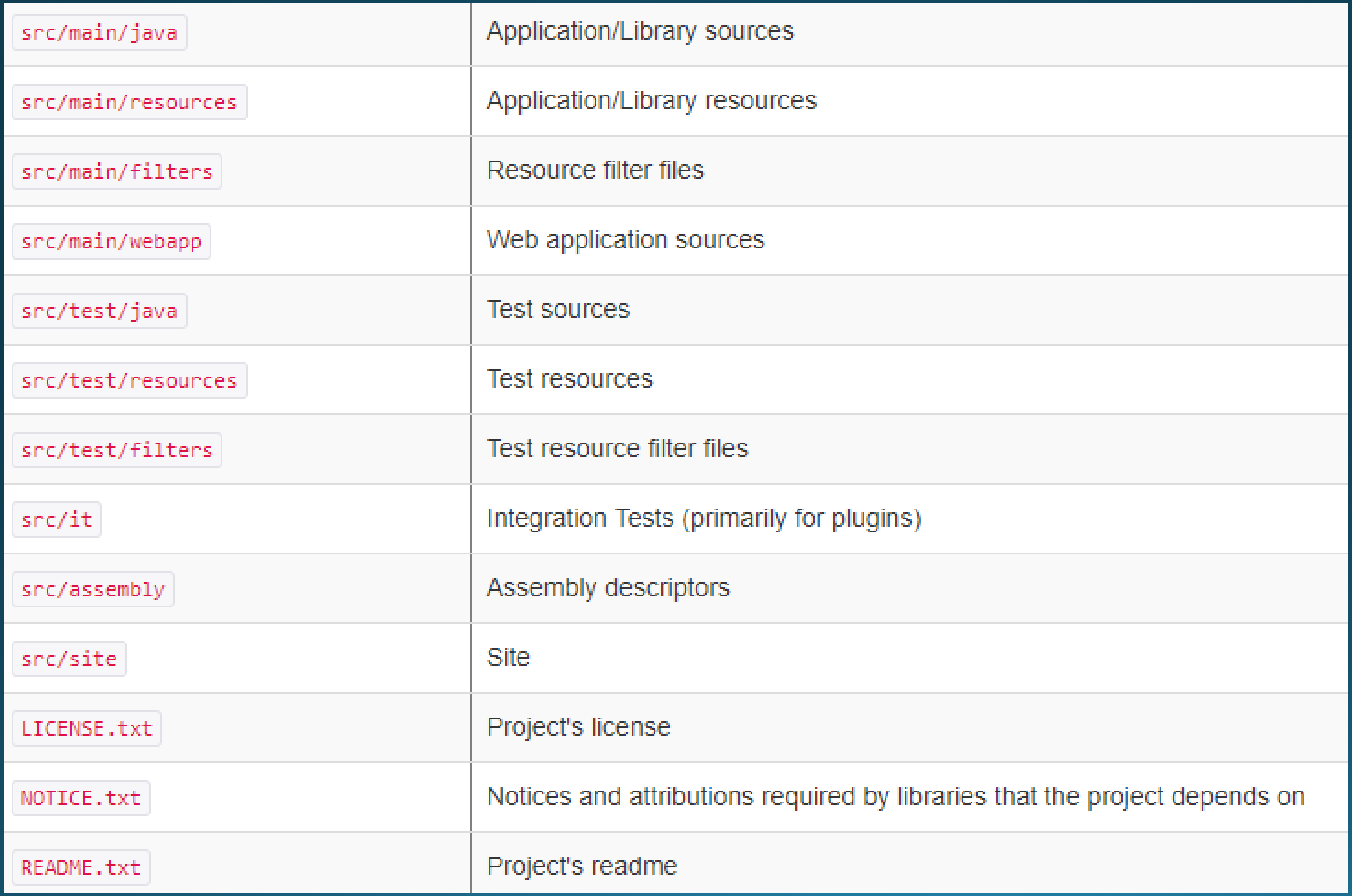
Lifecycle Phases

- The most important Maven *lifecycle* phases:
  - *validate* – checks the correctness of the project
  - *compile* – compiles the provided source code into binary artifacts
  - *test* – executes unit tests
  - *package* – packages compiled code into an archive file
  - *integration-test* – executes additional tests, which require the packaging
  - *verify* – checks if the package is valid
  - *install* – installs the package file into the local Maven repository
  - *deploy* – deploys the package file to a remote server or repository

Plugins and Goals

- A Maven *plugin* is a collection of one or more *goals*.
- Goals are executed in phases, which helps to determine the order in which the *goals* are executed.

Maven Project Directory Layout



`src/main/java`	Application/Library sources
`src/main/resources`	Application/Library resources
`src/main/filters`	Resource filter files
`src/main/webapp`	Web application sources
`src/test/java`	Test sources
`src/test/resources`	Test resources
`src/test/filters`	Test resource filter files
`src/it`	Integration Tests (primarily for plugins)
`src/assembly`	Assembly descriptors
`src/site`	Site
`LICENSE.txt`	Project's license
`NOTICE.txt`	Notices and attributions required by libraries that the project depends on
`README.txt`	Project's readme

Maven Archetypes

- Archetype is a Maven project templating toolkit.
- An archetype is defined as *an original pattern or model from which all other things of the same kind are made*.
- You use : **mvn archetype:generate** command to start creating a templated maven project
- Example to create a simple java project:

```
mvn -B archetype:generate \
-DarchetypeGroupId=org.apache.maven.archetypes \
-DgroupId=com.demo.app \
-DartifactId=myapp
```

Generate a Web project

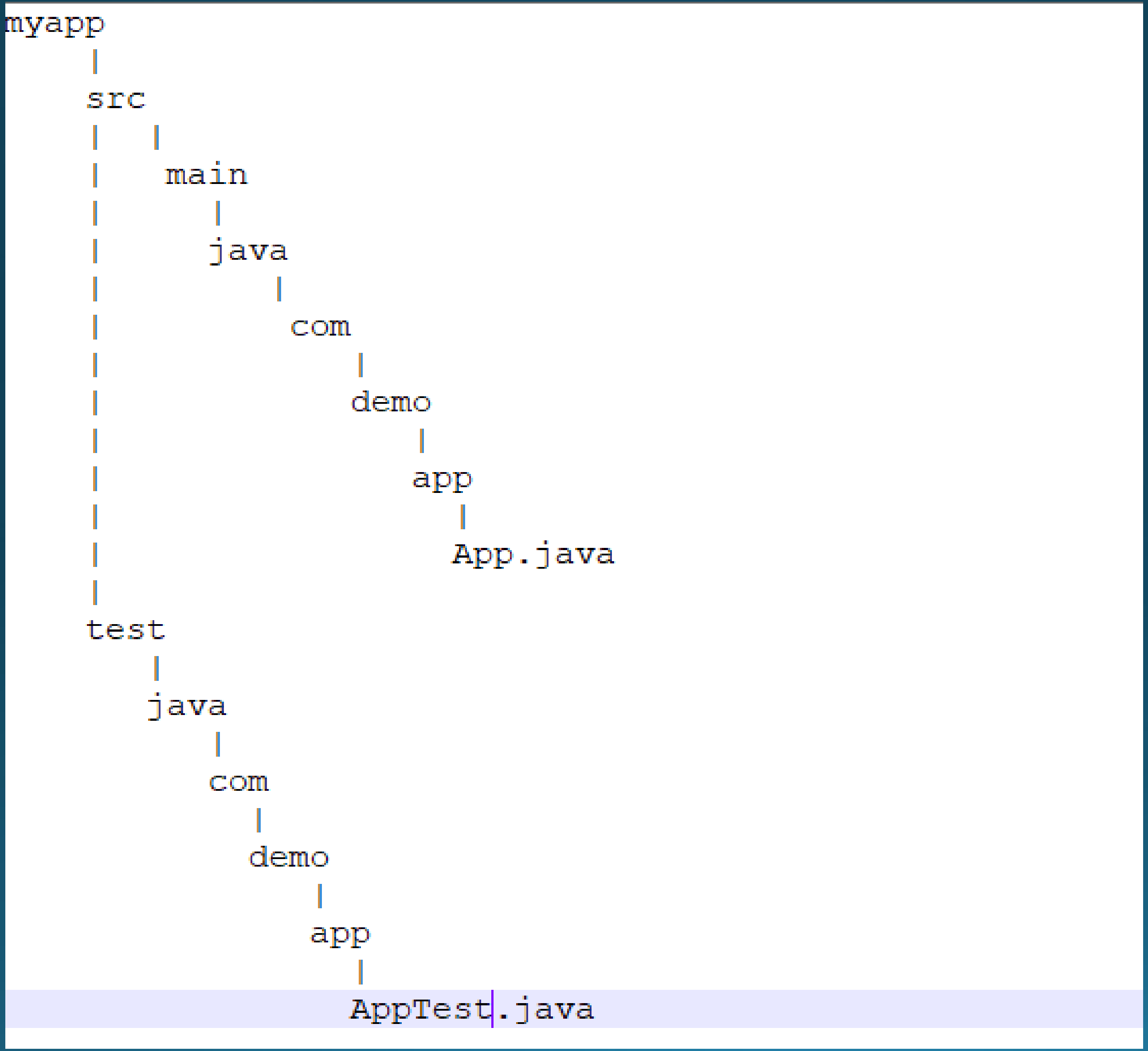
```
mvn archetype:generate -
DarchetypeGroupId=org.apache.maven.archetypes
-DarchetypeArtifactId=maven-archetype-webapp -
DarchetypeVersion=1.4
```

Create a Simple Maven Project

```
mvn -B archetype:generate \
-DarchetypeGroupId=org.apache.maven.archetypes
-DgroupId=com.demo.app \
-DartifactId=myapp
```

```
<project ..schema details ommited..>
 <modelVersion>4.0.0</modelVersion>
 <groupId>com.demo.app</groupId>
 <artifactId>myapp</artifactId>
 <packaging>jar</packaging>
 <version>1.0-SNAPSHOT</version>
 <name>myapp</name>
 <url>http://maven.apache.org</url>
 <dependencies>
  <dependency>
   <groupId>junit</groupId>
   <artifactId>junit</artifactId>
   <version>3.8.1</version>
   <scope>test</scope>
  </dependency>
 </dependencies>
</project>
```

Created Directory Structure



```
myapp
    |
    src
    |  |
    |   main
    |      |
    |      java
    |         |
    |          com
    |             |
    |             demo
    |                |
    |                app
    |                  |
    |                  App.java
    |
    test
       |
       java
          |
          com
            |
            demo
               |
               app
                 |
                 AppTest.java
```

Let’s Play with Maven now…

Install, run and create Projects and Test..