

# Software Project Management - Laboratory

Lecture n° 5  
A.Y. 2021-2022

Prof. Fabrizio Fornari

# Recap Previous Lectures

# Distributed Version Control System

Distributed means that there is no main server and all of the full history of the project is available once you cloned the project.

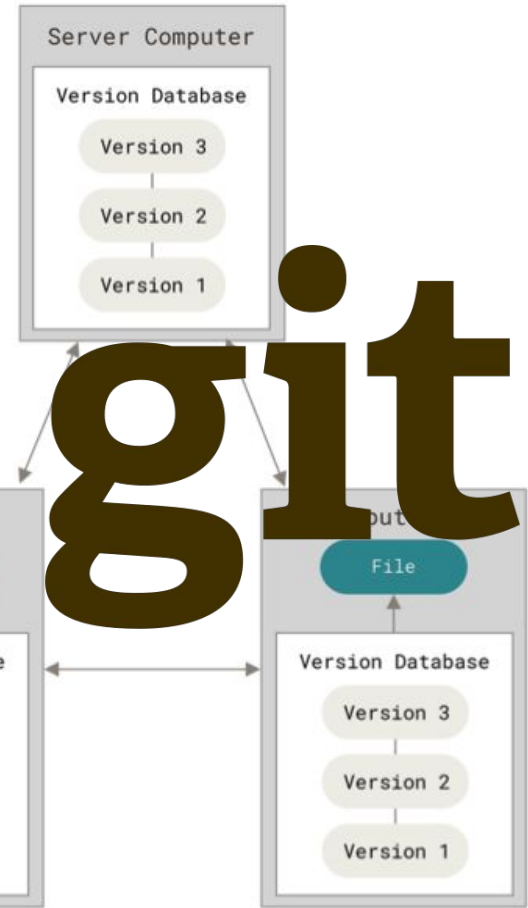
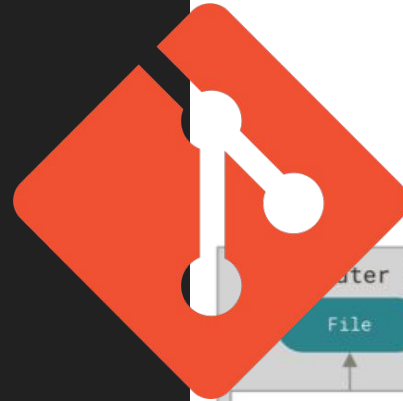


Figure 3. Distributed version control

# Remotes in Git

A remote repository is a repository stored somewhere else.

Most programmers use hosting services like:

- **GitHub**,
- **BitBucket**,
- **GitLab**



# Collaboration on GitHub

One person will be the “Owner” and the others will be the “Collaborators”.

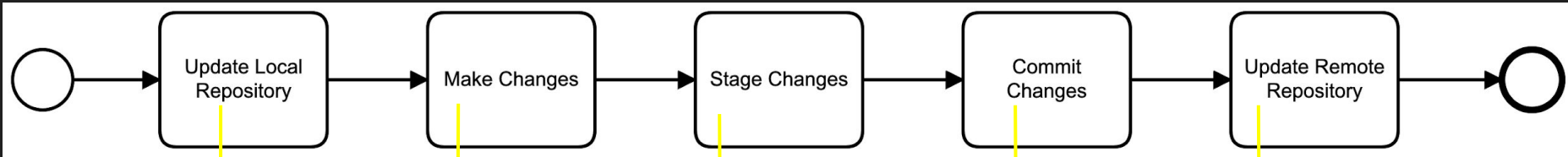
The Owner needs to give the Collaborators access.

The Collaborators will receive an email and/or they can check notifications on <https://github.com/notifications>

The screenshot shows the GitHub repository settings page for 'FabrizioFornari / spm2020'. The page is divided into several sections:

- Navigation:** At the top, there are links for 'Code', 'Issues', 'Pull requests', 'Actions', 'Projects', 'Wiki', 'Security', 'Insights', and 'Settings' (which is highlighted).
- Repository Info:** Shows 'Unwatch' (1), 'Star' (0), and 'Fork' (0).
- Options:** A sidebar menu with 'Manage access' selected.
- Who has access:** Two panels are visible:
  - PUBLIC REPOSITORY:** This repository is public and visible to anyone. A 'Manage' link is provided.
  - DIRECT ACCESS:** 0 collaborators have access to this repository. Only you can contribute to this repository.
- Manage access:** A section with a lock icon and the text 'You haven't invited any collaborators yet'. A green button labeled 'Invite a collaborator' is present.

# Remember the Workflow



`git pull origin master`

`echo A new line in a text file > NewFile.txt`

`git add NewFile.txt`

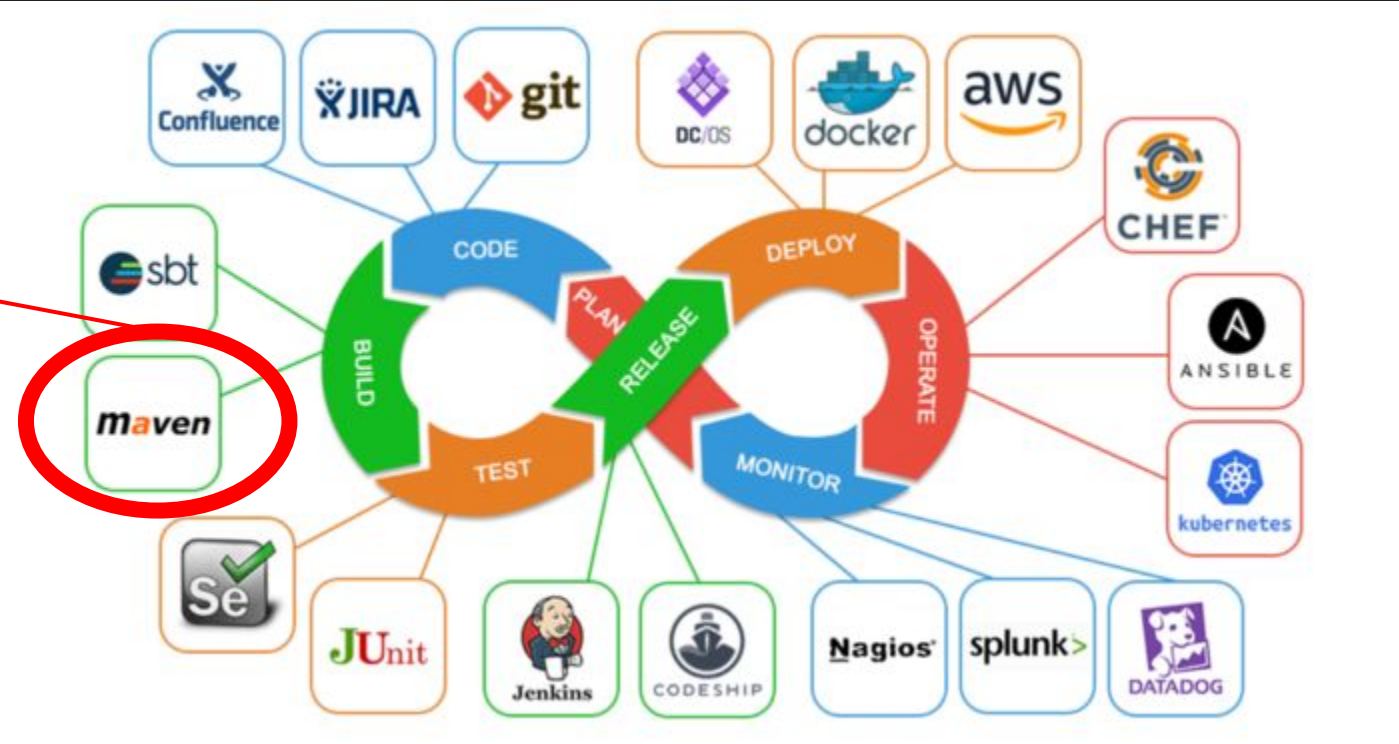
`git commit -m "Add a new file"`

`git push origin master`

# Today's Lecture

# DevOps

Our Focus





# Apache Maven

Apache Maven is an open source, standards-based project management framework that simplifies the building, testing, reporting, and packaging of projects.



<http://maven.apache.org>

# Maven's History

Maven's initial **roots** were in the Apache Jakarta Alexandria project that took place in early 2000. It was subsequently used in the Apache Turbine project. Like many other Apache projects at that time, the Turbine project had several subprojects, each with its own Ant- based build system.

<https://www.apache.org/>

Back then, there was a strong **desire for developing a standard way to build projects and to share generated artifacts easily across projects**. This desire gave birth to Maven. **Maven version 1.0** was released in **2004**, followed by **version 2.0** in **2005** and **version 3.0** in **2010**.

The **current version** of Maven is **3.8.3**

Maven has become one of the most widely used open source software programs in enterprises around the world. Let's look at some of the reasons why Maven is so popular.

# Maven - Standardized Directory Structure

Maven addresses the preceding problems by **standardizing the folder structure and organization of a project**.

Maven provides **recommendations** on where different parts of a project, such as source code, test code, and configuration files, should reside.

For example, Maven suggests that all of the **Java source code** should be placed in the **src\main\java** folder. This makes it easier to understand and navigate any Maven project.

# Maven - Declarative Dependency Management

Most Java projects rely on other projects and open source frameworks to function properly. It can be cumbersome to download these dependencies manually and keep track of their versions as you use them in your project.

Maven provides a convenient way to **declare** these **project dependencies** in a separate, external **pom.xml** file. It then **automatically downloads** those **dependencies** and allows you to use them in your project. This simplifies project dependency management greatly. It is important to note that in the pom.xml file, you specify the what and not the how.

# Maven - Archetypes

Maven archetypes are **predefined project templates** that can be used to generate new projects.

Projects created using archetypes will contain all of the folders and files needed to easily start working on a software project.

E.g., consider a team that works heavily on Spring framework-based web applications.

All Spring-based web projects share common dependencies and require a set of configuration files. It is also highly possible that all of these web projects have similar Log4j/Logback configuration files, CSS/Images, and Thymeleaf page layouts. Maven lets this team bundle these common assets into an archetype. When new projects get created using this archetype, they will automatically have the common assets included.

# Maven - Alternatives

## Apache Ant

Apache Ant (<http://ant.apache.org>) is a popular **open source tool for scripting builds**. Ant is Java based, and it uses Extensible Markup Language (XML) for its configuration. The default configuration file for Ant is the **build.xml** file.

### *Listing 1-1.* Sample Ant build.xml File

```
<project name="Sample Build File" default="compile"
basedir=". ">

  <target name="compile" description="Compile Source Code">
    <echo message="Starting Code Compilation"/>
    <javac srcdir="src" destdir="dist"/>
    <echo message="Completed Code Compilation"/>
  </target>

</project>
```

# Maven - Alternatives

## Apache Ivy

Apache Ivy (<http://ant.apache.org/ivy/>) provides **automated dependency management**, making Ant more joyful to use. With Ivy, we declare the dependencies in an XML file called **ivy.xml**

### ***Listing 1-2.*** Sample Ivy Listing

```
<ivy-module version="2.0">
  <info organisation="com.apress" module="gswm-ivy" />

  <dependencies>
    <dependency org="org.apache.logging.log4j" name="log4j-api"
      rev="2.11.2" />
  </dependencies>
</ivy-module>
```

# Maven - Alternatives

## Gradle

Gradle (<http://gradle.org/>) is an **open source build, project automation tool** that can be used for Java and non-Java projects. Unlike Ant and Maven, which use XML for configuration, **Gradle uses a Groovy-based Domain-Specific Language (DSL)**.

**Listing 1-3.** Default build.gradle File

```
plugins {
    id 'java'
}

version = '1.0.0'

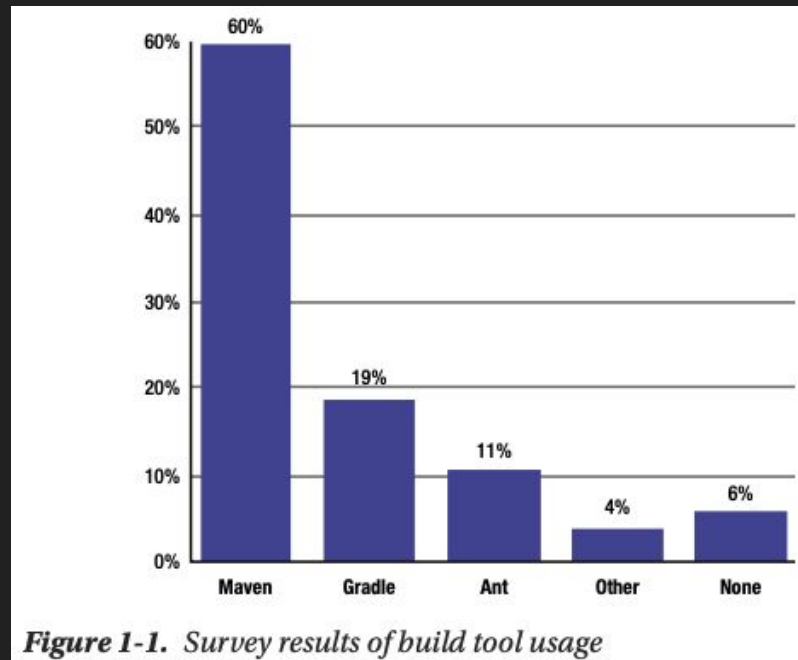
repositories {
    mavenCentral()
}

dependencies {
    testCompileOnly group: 'junit', name: 'junit',
    version: '4.10'
}
```



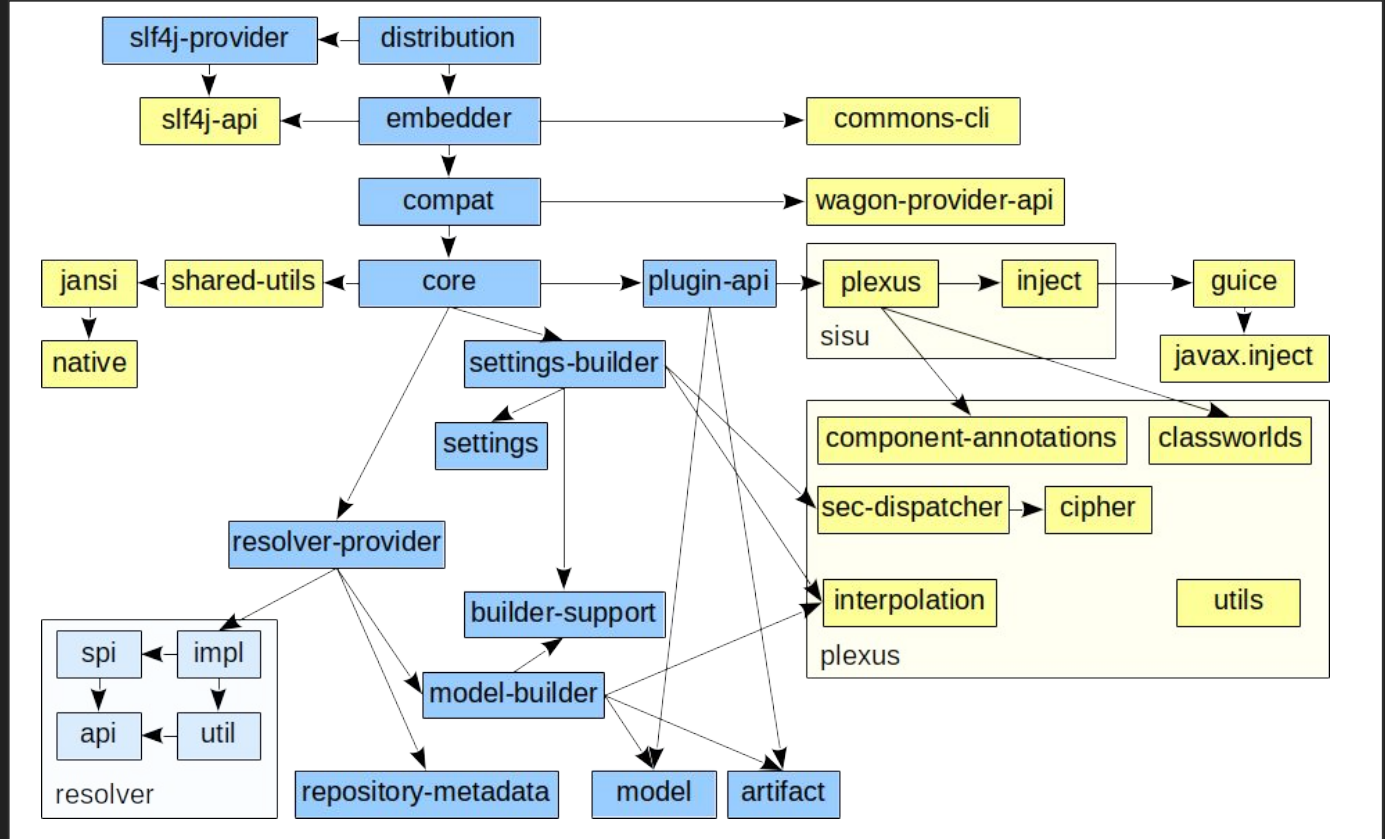
# Maven

Despite growing competition from other tools, Maven continues to dominate the build tool space.



# Maven

Maven can be extended by plugins to utilise a number of other development tools for reporting or the build process



# Maven - Convention over Configuration

Maven uses Convention over Configuration, which means developers are not required to create build process themselves.

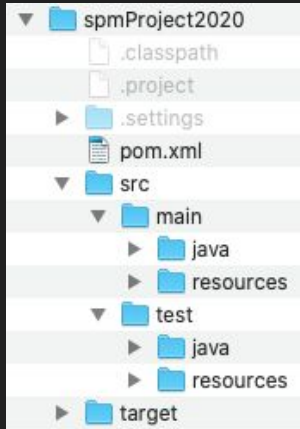
Developers do not have to mention each and every configuration detail. **Maven provides sensible default behavior for projects**. When a Maven project is created, **Maven creates default project structure**. Developer is only required to place files accordingly and he/she need not to define any configuration in **pom.xml**.

# Maven - Convention over Configuration

As an example, following table shows the **default values for project source code files, resource files and other configurations**. Assuming, `$basedir` denotes the project location

Item	Default
Source code	<code>\${basedir}/src/main/java</code>
Resources	<code>\${basedir}/sr/main/resources</code>
Tests	<code>\${basedir}/src/tests</code>
Compiled byte code	<code>\${basedir}/target</code>
Distributable JAR	<code>\${basedir}/target/classes</code>

# Maven - Convention over Configuration



- **spmProject2020** is the root folder of the project. Typically, the name of the root folder matches the name of the generated artifact.
- **src** contains project-related artifacts such as source code or property files, which you typically would like to manage in a source control management (SCM) system, such as Git.
- **src/main/java** folder contains the Java source code.
- **src/test/java** folder contains the Java unit test code.
- **target** folder holds generated artifacts, such as .class files. Generated artifacts are typically not stored in SCM, so you don't commit the target folder and its contents into SCM.
- **pom.xml** file. It holds project and configuration information, such as dependencies and plug-ins

# Maven - Convention over Configuration

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 - Environment Setup

Maven is a Java-based application and requires the Java Development Kit (JDK) to function properly.

You can install it by downloading the latest version of Maven from the Apache Maven web site <http://maven.apache.org/download.html>

After some setup you will end up with Maven installed <https://maven.apache.org/install.html>

```
fabriziounicam:Local user$ mvn -v
Apache Maven 3.3.9 (bb52d8502b132ec0a5a3f4c09453c07478323dc5; 2015-11-10T17:41:47+01:00)
Maven home: /usr/local/apache-maven/apache-maven-3.3.9
Java version: 1.8.0_161, vendor: Oracle Corporation
Java home: /Library/Java/JavaVirtualMachines/jdk1.8.0_161.jdk/Contents/Home/jre
Default locale: en_US, platform encoding: UTF-8
OS name: "mac os x", version: "10.15.4", arch: "x86_64", family: "mac"
```


 All modern IDEs come with full Maven integration without needing any further configuration

# Maven - Environment Setup

Install Eclipse IDE for Enterprise Java and Web Developers


## Eclipse IDE for Enterprise Java and Web Developers

517 MB | 508,841 DOWNLOADS



Tools for developers working with Java and Web applications, including a Java IDE, tools for JavaScript, TypeScript, JavaServer Pages and Faces, Yaml, Markdown, Web Services, JPA and Data Tools, Maven and Gradle, Git, and more.

Click [here](#) to file a bug against Eclipse Web Tools Platform.  
Click [here](#) to file a bug against Eclipse Platform.  
Click [here](#) to file a bug against Maven integration for web projects.  
Click [here](#) to report an issue against Eclipse Wild Web Developer (incubating).

 Windows [x86\\_64](#)  
macOS [x86\\_64](#) |  
Linux [x86\\_64](#) | [AArch64](#)

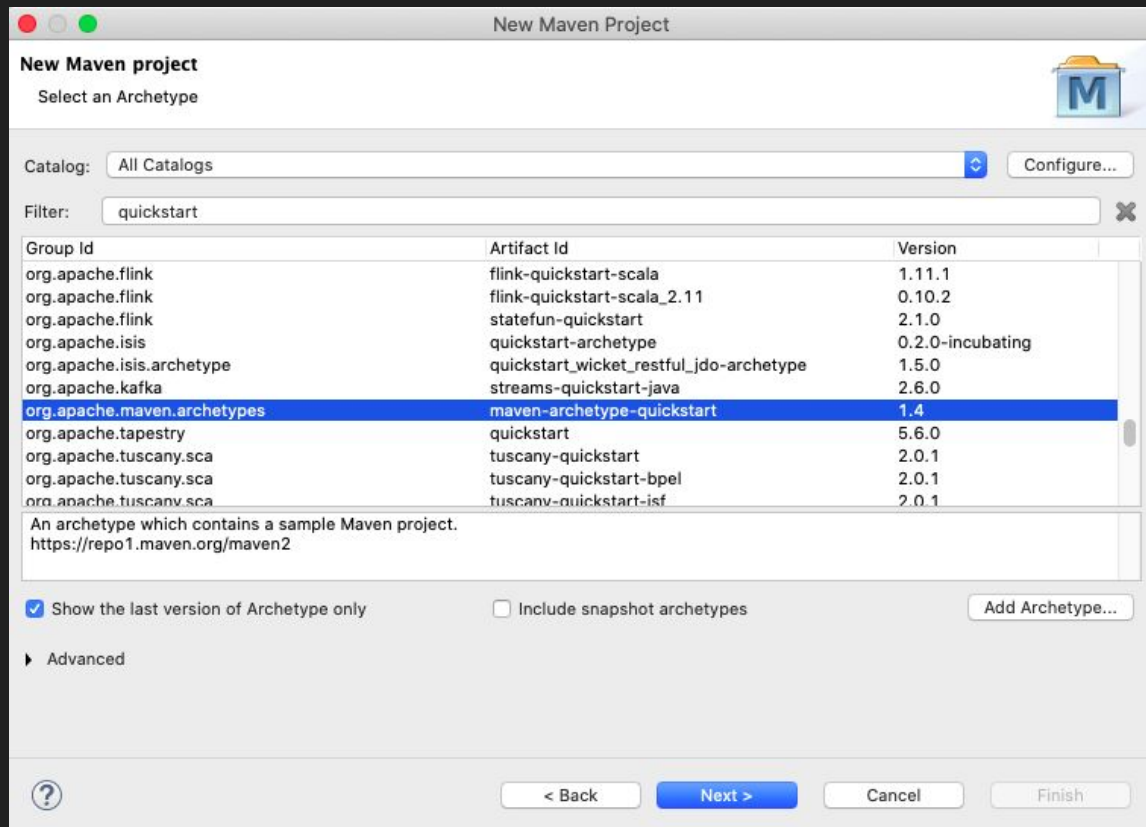
<https://www.eclipse.org/downloads/packages/release/2020-12/r/eclipse-ide-enterprise-java-developers>



# Maven - Project Setup

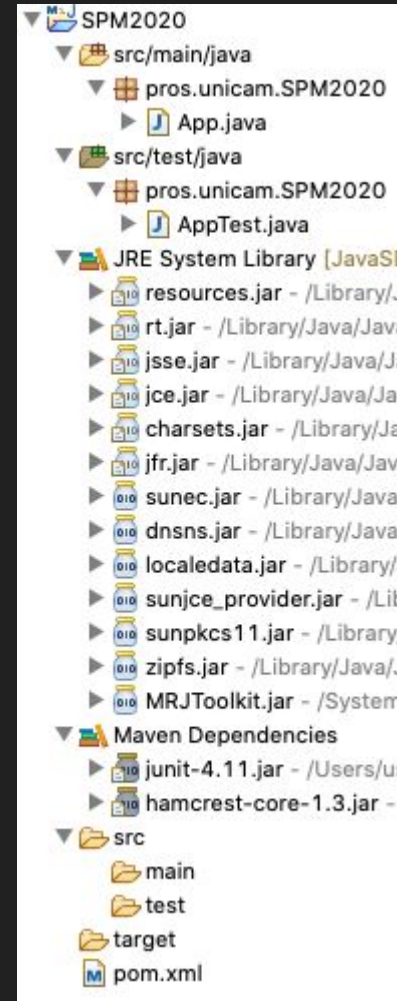
Create a Maven Project by following:  
File → New → Other → Maven  
Project → Next

Insert “maven-archetype-quickstart”,  
select and proceed



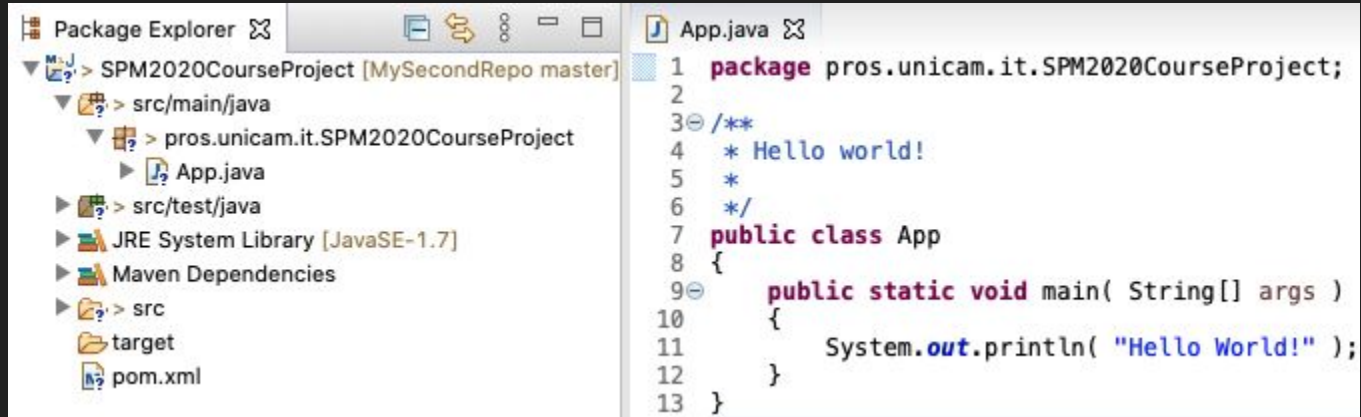
# Maven - Project Setup

This is the structure and the content of your project.



# Maven - Project Setup

1. Inspect the project folders and files.
2. Run App.java as Java Application



The screenshot displays an IDE interface with two main panels. The left panel, titled 'Package Explorer', shows a project tree for 'SPM2020CourseProject [MySecondRepo master]'. The tree structure is as follows:

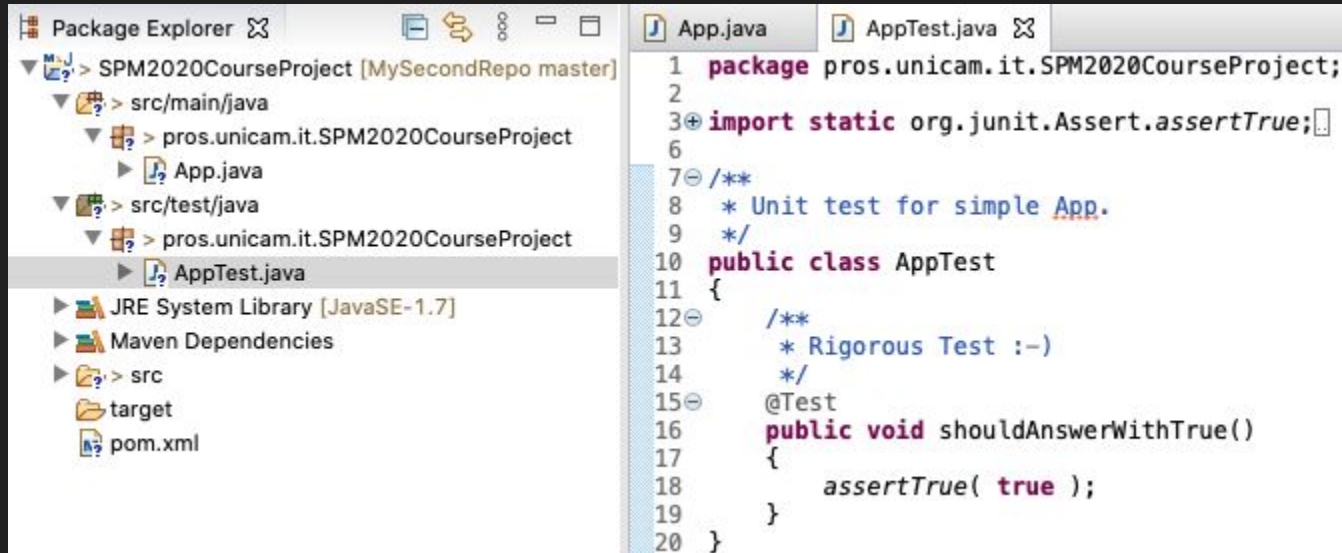
- SPM2020CourseProject [MySecondRepo master]
  - src/main/java
    - pros.unicam.it.SPM2020CourseProject
      - App.java
  - src/test/java
  - JRE System Library [JavaSE-1.7]
  - Maven Dependencies
  - src
  - target
  - pom.xml

The right panel, titled 'App.java', shows the source code of the Java application:

```
1 package pros.unicam.it.SPM2020CourseProject;
2
3 /**
4  * Hello world!
5  *
6  */
7 public class App
8 {
9     public static void main( String[] args )
10    {
11        System.out.println( "Hello World!" );
12    }
13 }
```

# Maven - Testing

1. Inspect the project folders and files.
2. Run AppTest.java as JUnit Test



The screenshot displays an IDE interface with two main panels. The left panel, titled 'Package Explorer', shows a project named 'SPM2020CourseProject [MySecondRepo master]'. The project structure is as follows:

- src/main/java
  - pros.unicam.it.SPM2020CourseProject
    - App.java
- src/test/java
  - pros.unicam.it.SPM2020CourseProject
    - AppTest.java

Below the project structure, there are sections for 'JRE System Library [JavaSE-1.7]', 'Maven Dependencies', and a 'src' folder containing 'target' and 'pom.xml'.

The right panel shows the code for 'AppTest.java'. The code is as follows:

```
1 package pros.unicam.it.SPM2020CourseProject;
2
3 import static org.junit.Assert.assertTrue;
4
5
6
7 /**
8  * Unit test for simple App.
9  */
10 public class AppTest
11 {
12     /**
13      * Rigorous Test :-)
14      */
15     @Test
16     public void shouldAnswerWithTrue()
17     {
18         assertTrue( true );
19     }
20 }
```

Testing

To be continued...

# Maven - POM

Maven project structure and contents are declared in an xml file, `pom.xml`, referred as Project Object Model (POM), which is the **fundamental unit of the entire Maven system**.

The POM contains information about the project and various configuration details used by Maven to build the project(s).

POM also contains the goals and plugins. While executing a task or goal, Maven looks for the POM in the current directory. It reads the POM, gets the needed configuration information, and then executes the goal.

Some of the configuration that can be specified in the POM are:

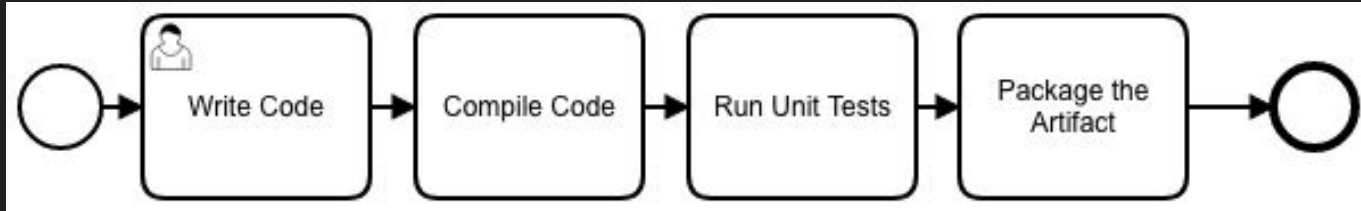
- project dependencies
- plugins
- goals
- build profiles
- project version

# Maven

Tag	Description
project	The project root tag. You need to specify the basic schema settings such as apache schema and w3.org specification
model Version	Should be set to 4.0.0. Is the version of project descriptor your POM conforms to.
groupId	The Id of the project group. It is generally unique amongst an organization or a project.
artifactId	The Id of the project.
version	The version of the project. E.g., 0.0.1-SNAPSHOT
name	The name of the project.

# Maven Lifecycle

**Build processes** generating artifacts such as JAR or WAR files typically require several steps and tasks to be completed successfully in a well-defined order. Examples of such tasks include **compiling source code**, **running unit tests**, and **packaging of the artifact**. Maven uses the concept of goals to represent such granular tasks.





# Maven - Plugins

Goals in Maven are packaged in **plug-ins**, which are essentially a collection of one or more goals.

## Compiler Plugin

```
<plugin>
  <artifactId>maven-compiler-plugin</artifactId>
  <version>3.8.0</version>
</plugin>
```

The compile goal identifies the Java class HelloWorld.java under **src/main/java**, **compiles it**, and **places the compiled class file** under the **target\classes** folder.

## Clean Plugin

```
<!-- clean lifecycle, see https://maven.apache.org/ref/current/maven-core/lifecycles.html#clean_Lifecycle -->
<plugin>
  <artifactId>maven-clean-plugin</artifactId>
  <version>3.1.0</version>
</plugin>
```

The clean goal accomplishes exactly that, as it attempts to **delete the target folder and all its contents**.

# Maven Lifecycle

Maven goals are granular and typically perform one task.

To perform complex operations.

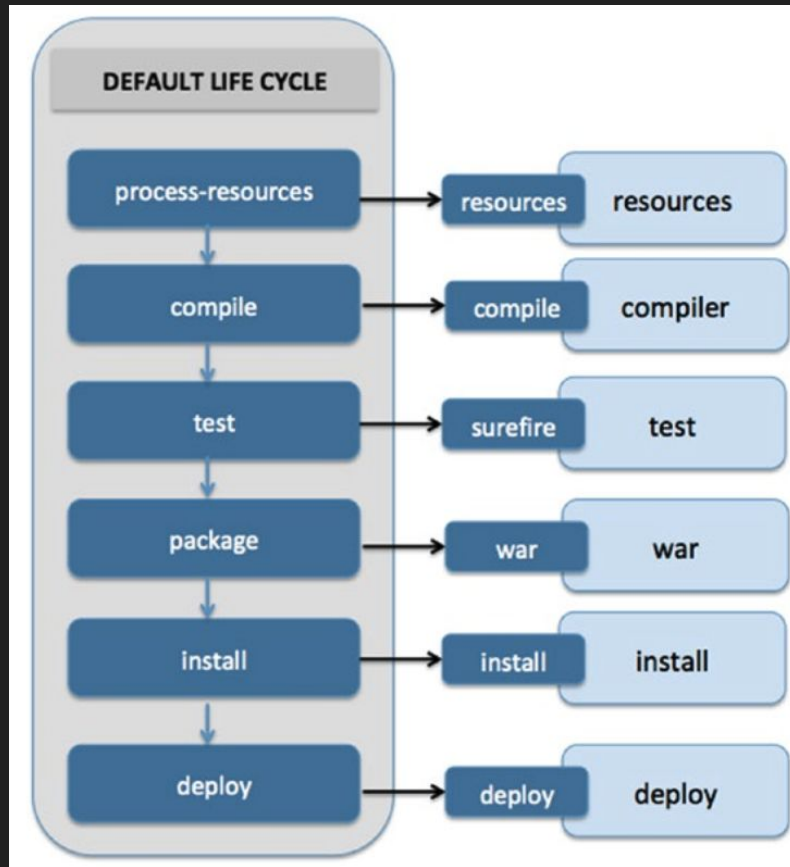
Maven simplifies these complex operations via lifecycle and phase abstractions

Every Maven project has the following three built-in lifecycles:

default	This lifecycle handles the compiling, packaging, and deployment of a Maven project.
clean	This lifecycle handles the deletion of temporary files and generated artifacts from the target directory.
site	This lifecycle handles the generation of documentation and site generation.

# Maven Lifecycle

<i>validate</i>	Runs checks to ensure that the project is correct and that all dependencies are downloaded and available.
<i>compile</i>	Compiles the source code.
<i>test</i>	Runs unit tests using frameworks. This step doesn't require that the application be packaged.
<i>package</i>	Assembles compiled code into a distributable format, such as JAR or WAR.
<i>install</i>	Installs the packaged archive into a local repository. The archive is now available for use by any project running on that machine.
<i>deploy</i>	Pushes the built archive into a remote repository for use by other teams and team members.



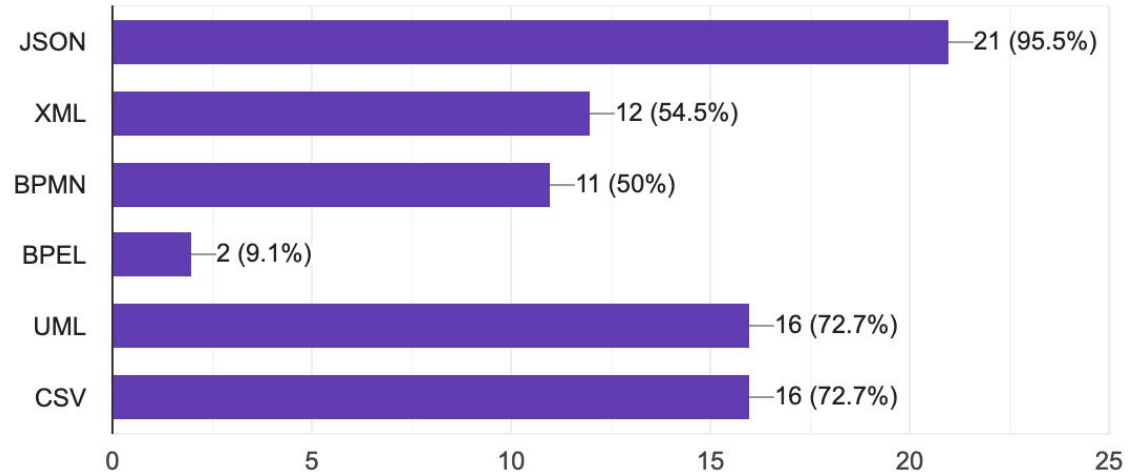
# Time to Exercise



# Do you know JSON?

Which of the following formats/notations are you familiar with?

22 responses



<https://www.json.org/json-en.html>

# Maven - Dependency Management



<https://mvnrepository.com/>

Search the library you need  
and add it to the POM

I searched for a JSON library

Note: if you don't know about JSON  
<https://www.json.org/json-en.html>

The screenshot shows the Maven Repository website interface. At the top, there is a search bar with the text "Search for groups, artifacts, categories" and a "Search" button. Below the search bar, the breadcrumb navigation reads "Home » org.json » json » 20200518". The main content area displays the title "JSON In Java » 20200518" next to a circular icon. A description follows: "JSON is a light-weight, language independent, data interchange format. See <http://www.JSON.org/> The files in this package implement JSON encoders/decoders in Java. It also includes the capability to convert between JSON and XML, HTTP headers, Cookies, and CDL. This is a reference implementation. There is a large number of JSON packages in Java. Perhaps someday the Java community will standardize on one. Until then, choose carefully. The license includes this restriction: "The software ..."

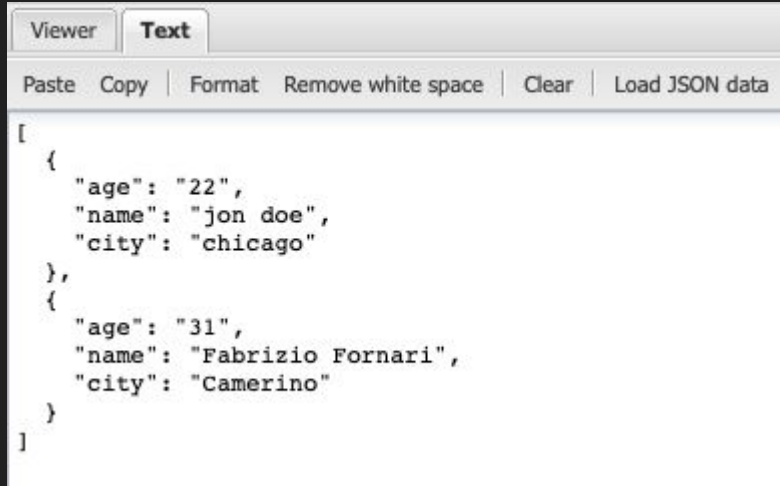
Below the description is a table of metadata:

License	JSON
Categories	JSON Libraries
HomePage	<a href="https://github.com/douglascrockford/JSON-java">https://github.com/douglascrockford/JSON-java</a>
Date	(May 22, 2020)
Files	bundle (64 KB) View All
Repositories	Central
Used By	3,839 artifacts

At the bottom of the page, there are tabs for different build systems: Maven, Gradle, SBT, Ivy, Grape, Leiningen, and Buildr. Below these tabs is a code block showing the XML dependency declaration:

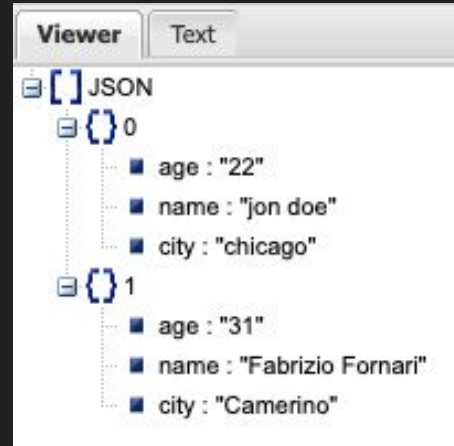
```
<!-- https://mvnrepository.com/artifact/org.json/json -->
<dependency>
  <groupId>org.json</groupId>
  <artifactId>json</artifactId>
  <version>20200518</version>
</dependency>
```

# Create a JSON file



A screenshot of a web-based JSON viewer interface. The interface has a top bar with 'Viewer' and 'Text' tabs. Below the tabs is a menu with options: 'Paste', 'Copy', 'Format', 'Remove white space', 'Clear', and 'Load JSON data'. The main area displays the following JSON text:

```
[
  {
    "age": "22",
    "name": "jon doe",
    "city": "chicago"
  },
  {
    "age": "31",
    "name": "Fabrizio Fornari",
    "city": "Camerino"
  }
]
```



<http://jsonviewer.stack.hu/>

# Maven - Dependency Management



<https://mvnrepository.com/>

Search the library you need  
and add it to the POM

I searched for a JSON library

I added it to the POM and I  
build the project

```
7 <groupId>pros.unicam</groupId>
8 <artifactId>SPM2020CourseProject</artifactId>
9 <version>0.0.1-SNAPSHOT</version>
10
11 <name>SPM2020CourseProject</name>
12 <!-- FIXME change it to the project's website -->
13 <url>http://www.example.com</url>
14
15⊖ <properties>
16   <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
17   <maven.compiler.source>1.7</maven.compiler.source>
18   <maven.compiler.target>1.7</maven.compiler.target>
19 </properties>
20
21⊖ <dependencies>
22⊖   <dependency>
23     <groupId>junit</groupId>
24     <artifactId>junit</artifactId>
25     <version>4.11</version>
26     <scope>test</scope>
27   </dependency>
28   <!-- https://mvnrepository.com/artifact/org.json/json -->
29⊖   <dependency>
30     <groupId>org.json</groupId>
31     <artifactId>json</artifactId>
32     <version>20200518</version>
33   </dependency>
34 </dependencies>
```



# Create a JSON file

```
public class App
{
    public static void main( String[] args )
    {
        System.out.println( "Hello World!" );
        writeJson();|
        //writeJsonArray();
    }

    public static void writeJson() {

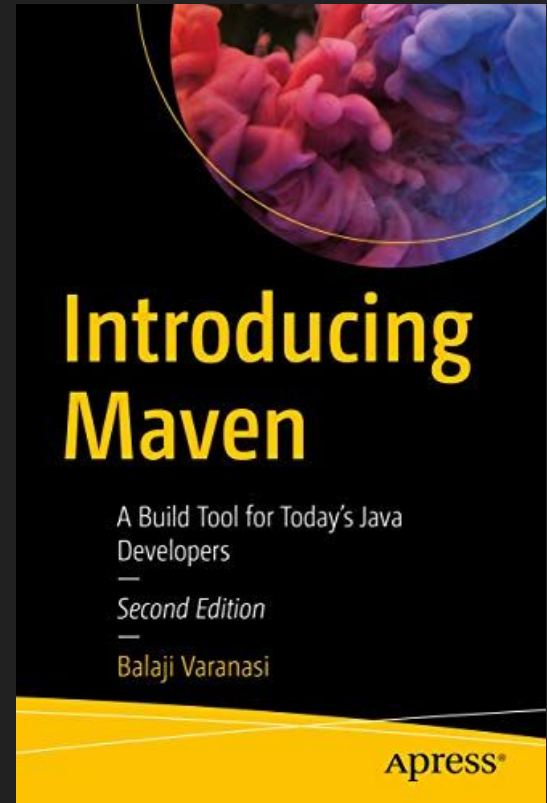
        JSONObject jo = new JSONObject();
        jo.put("name", "jon doe");
        jo.put("age", "22");
        jo.put("city", "chicago");

        String fileName = "/Users/user/Desktop/SPM/fileJSON.json";
        try (BufferedWriter writer = Files.newBufferedWriter(Paths.get(fileName), StandardCharsets.UTF_8)) {
            jo.write(writer);
            writer.write("\n");
        } catch (Exception ex) {
            System.err.println("Couldn't write contestNames\n"
                + ex.getMessage());
        }
        System.out.println("Successfully Copied JSON Object to File...");
        System.out.println("\nJSON Object: " + jo);
    }
}
```

# Maven - Additional Material

Introducing Maven:  
A Build Tool for Today's Java Developers.

by Balaji Varanasi



# Time to Exercise

1. From your command line, **Create a local git repository**
2. From a web browser, **Create a GitHub repository**
3. **Add** other your colleagues as **collaborators**
4. From your command line, **Connect the local repository with the remote repository on GitHub**
5. From your IDE (e.g. Eclipse), **Create a Maven project**
6. From your command line, **Run git status**

Did anything changed inside your local git repository?

Yes

No

Why?

# Time to Exercise

1. From your command line, **Create a local git repository**
2. From a web browser, **Create a GitHub repository**
3. **Add** other your colleagues as **collaborators**
4. From your command line, **Connect the local repository with the remote repository on GitHub**
5. From your IDE (e.g. Eclipse), **Create a Maven project**
6. From your command line, **Run git status**
7. **Ensure** that you have created the maven project inside your local repository folder
8. Make some changes to the code and **commit them and push them by using the command line**