MAVEN

Non Business KT Session

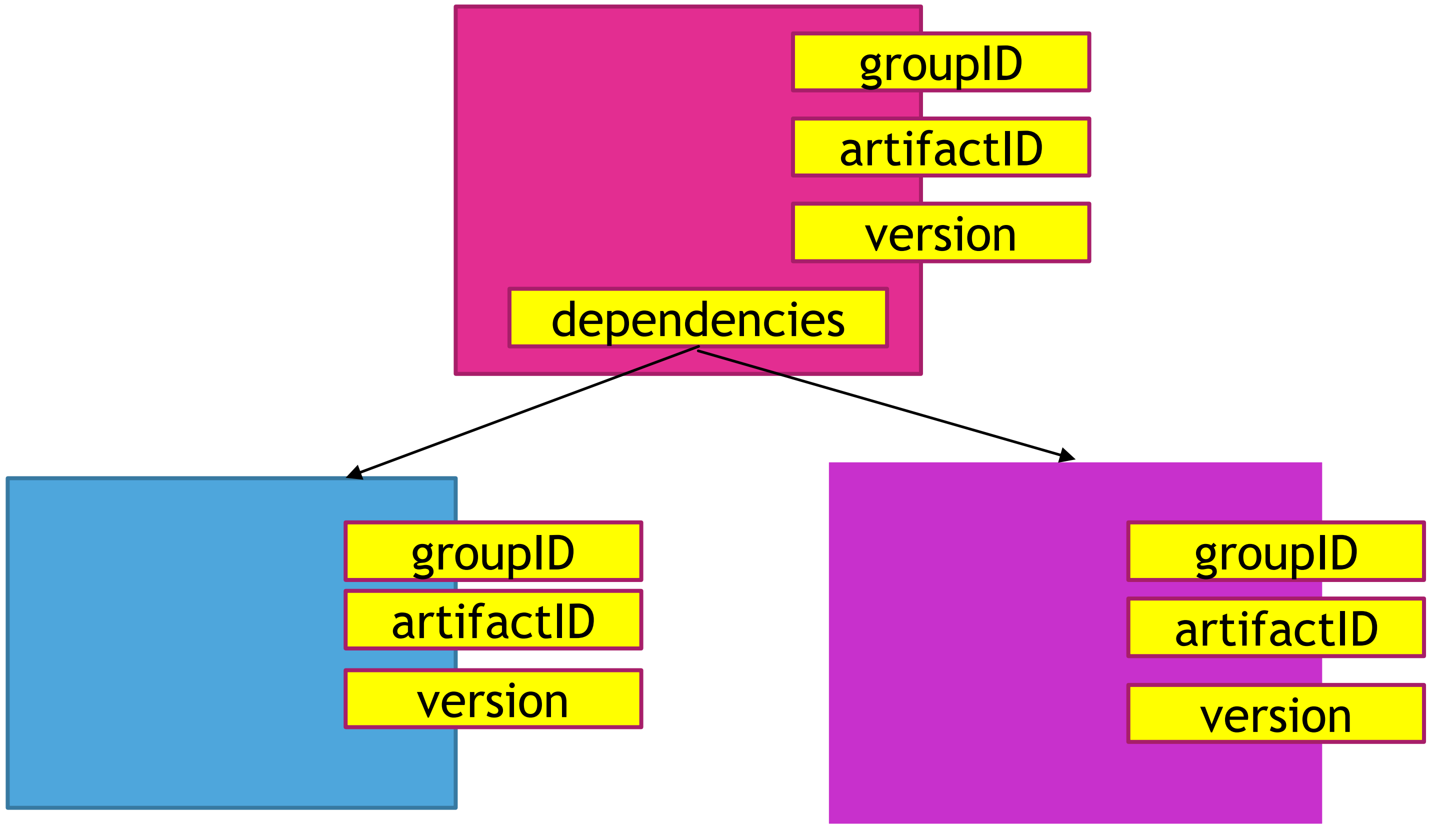
What is Maven ?

- Maven is an attempt *to "apply patterns to a project's build infrastructure in order to promote comprehension and productivity by providing a clear path in the use of best practices"*
- It helps in managing
  - Builds
  - Dependencies and their versions
  - Documentation
  - Releases and Distribution
- Maven is useful for us developers as it uses standard conventions and practices to help accelerate the build process.

Why Maven ?

- Maven’s philosophy is to implement and encourage the use of patterns that display Visibility, Reusability, Maintainability and Comprehensibility.
- These characteristics are developed and decided after years of developing projects as groups.
  - Visibility allows team members to easily view the progress of each other.
  - This visibility allows members to reuse code.
  - Code reusability is backbone of maintainable system.
  - This ease allows teams to build easily comprehensible projects.
- Maven makes it very easy for developers to switch between projects.
- This is due to the same approach of build -> test -> package -> document -> deploy

Maven UIDs


groupID
artifactID
version
dependencies
groupID
artifactID
version
groupID
artifactID
version

POM.xml

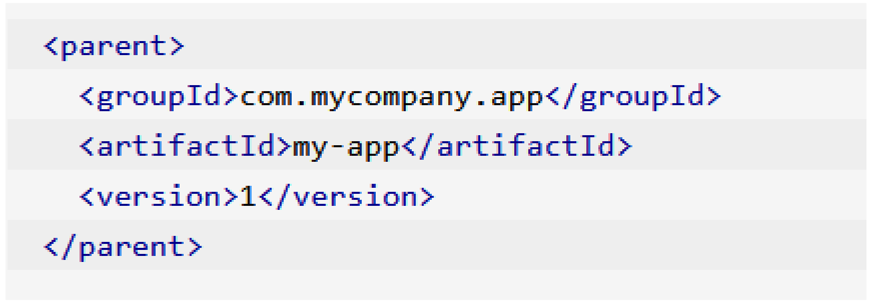
- A Project Object Model or POM is the fundamental unit of work in Maven.
- Basically an XML file, that contains configuration details to build project.
- Maven executing a task, Maven will look for configuration information from POM and execute it.

```
<modelVersion>4.0.0</modelVersion>
<groupId>org.openjfx</groupId>
<artifactId>hellofx</artifactId>
<packaging>jar</packaging>
<version>1.0-SNAPSHOT</version>
```

- Model version can be considered to be maven version.
- “groupID” is identifier for organization that is building this project.
- “artifictID” will be the name of the final product we create.
- Packaging is the way this package will be packaged .war, .jar,

POM.xml

- We can also use POM inheritance, to use all the dependencies mentioned in the parent so that it can be used in child parent.



```xml
<parent>
  <groupId>com.mycompany.app</groupId>
  <artifactId>my-app</artifactId>
  <version>1</version>
</parent>
```

Referenced : Maven Documentation. Available at: https://maven.apache.org/guides/introduction/introduction-to-the-pom.html (Last accessed Feb.19, 2023)

- Also SNAPSHOT means work in progress and RELEASE means the final released version.
- We also add dependencies that were mentioned in previous slides, they are collection of external libraries or files that we need in our project.

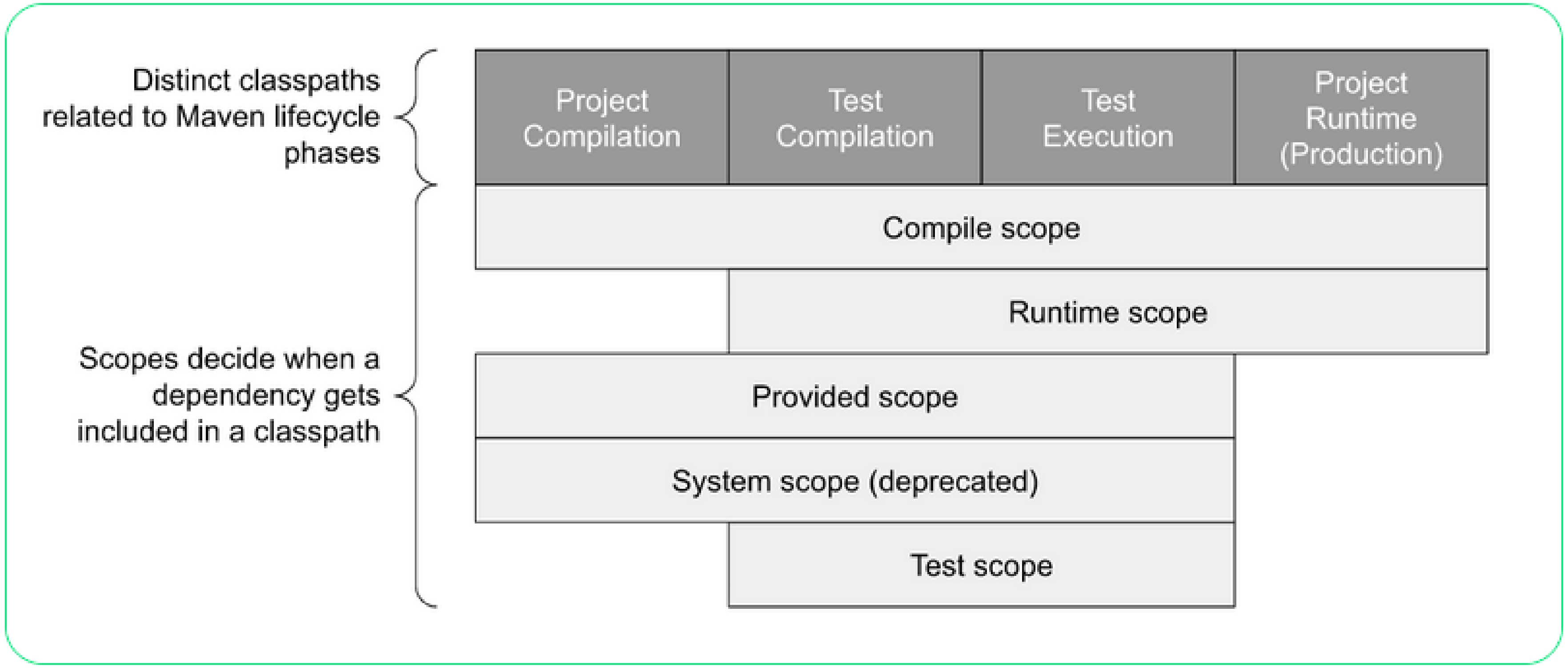
Dependencies in POM.xml

- We have an individual dependency set, which has to be put inside dependencies tag XML listing all dependencies.
  - Version
  - groupID
  - artifactID
- Maven has transitive dependencies, so if we include a dependency that itself needs another dependency maven will take care of it.
- Maven also has a method of “dependency mediation” that helps in using a single “nearest definition” of a dependency will be used in the project.
- We can also specify the specific version of dependency if it is not provided, we can also specify the scope of dependency for specific stage of build.

Scopes in dependencies

- Maven has in total six scopes: compile, runtime, test, provided, system and import.
- The **compile** scope contains dependencies that are required for the compilation of the project under development. In other words, the source code of project classes directly or indirectly references classes of compile dependencies.
- **Runtime** scope indicates that the dependency is not required for compilation, but is for execution. Maven includes a dependency with this scope in the runtime and test classpaths, but not the compile classpath.
- The **provided** scope contains classes that will be provided by the project's runtime environment and thus do not need to be included in the artifact produced by the development project.
- **Test** scope indicates that the dependency is not required for normal use of the application, and is only available for the test compilation and execution phases.

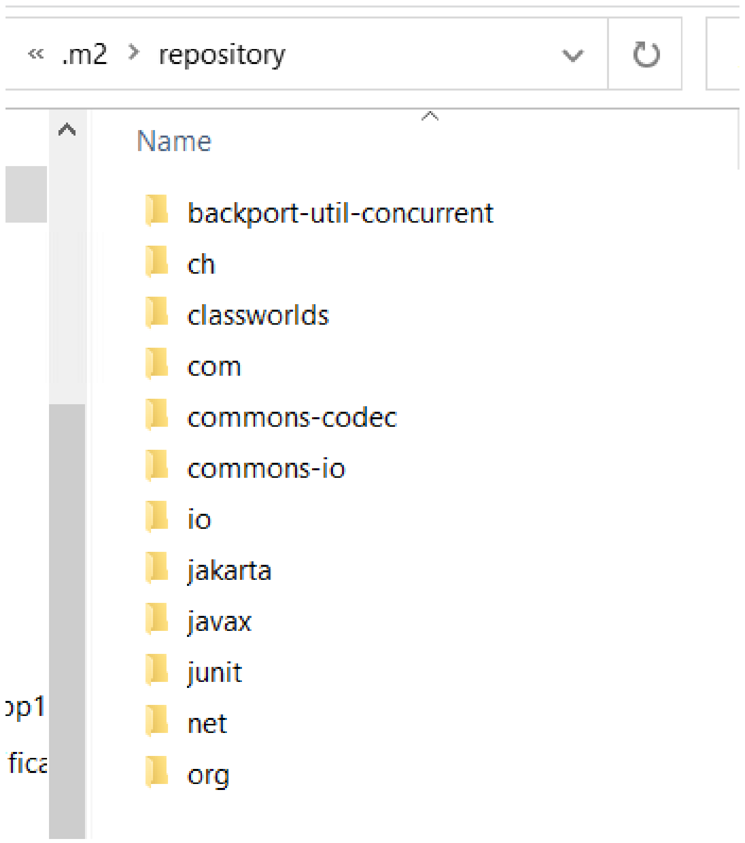
Dependencies Scopes


Distinct classpaths related to Maven lifecycle phases
Project Compilation
Test Compilation
Test Execution
Project Runtime (Production)
Compile scope
Runtime scope
Provided scope
System scope (deprecated)
Test scope
Scopes decide when a dependency gets included in a classpath

Reference: MAVEN Dependency Scopes. Available at: https://www.endorlabs.com/blog/what-are-maven-dependency-scopes-and-their-related-security-risks (Last accessed Feb 20, 2023)

Repositories in Maven

- We have two types of repositories in Maven which have build artifacts and dependencies of varying types.
- the **local** repository is a directory on the computer where Maven runs. It caches remote downloads and contains temporary build artifacts that we have not yet released.
- **remote** repositories refer to any other type of repository, accessed by a variety of protocols such as file:// and https://



backport-util-concurrent
ch
classworlds
com
commons-codec
commons-io
io
jakarta
javax
junit
net
org

Maven Build Cycle

- Maven is based around the central concept of a build lifecycle. What this means is that the process for building and distributing a particular artifact (project) is clearly defined.
- So the default lifecycle has some phases in it
  - Validate : validate the project is correct and all necessary information is available
  - Compile : compile the source code of the project
  - Test : test the compiled source code using a suitable unit testing framework. These tests should not require the code be packaged or deployed
  - Package: take the compiled code and package it in its distributable format, such as a JAR.
  - Verify: run any checks on results of integration tests to ensure quality criteria are met
  - Install: install the package into the local repository, for use as a dependency in other projects locally

Some Commands used often.

- “mvn clean” remove all files generated by the previous build.
- “mvn compile” compile the source code of the project.
- “mvn test” run tests using a suitable unit testing framework. These tests should not require the code be packaged or deployed.
- “mvn package” take the compiled code and package it in its distributable format, such as a JAR.
- “mvn install” install the package into the local repository, for use as a dependency in other projects locally.