MAVEN

↳ Build tool

· Everything in the maven uses plugin.

make is a build tool
↓
ant → maven → gradle
↳ all are build Tools

compiling 1000's of code is not possible & there comes the Build tool → automates the compiling the Source code and generate the artifactory.

Maven → Java based built tool

make → C, C++ built tool.

nAnt → .Net (windows)

other task a Build tool can do?

1] download dependencies
2] compiling source code
3] Packaging
4] Testing
5] documentation
6] Deployment

Java - Ant, Maven, Gradle
.NET - Nant.

ANT → another need tool
→ used to develop TOMCAT server.
→ More complex & more disadvantages → Maven was Born

1) Every build tool has an Input file.

Input file
Build file → [Maven → POM.xml
Ant → build.xml]

contains the tasks that needs to be executed.
have all instructions that build tool should perform.

[1st element - Project ~~em~~ element → contains XML version
Target - specifies a task

Apache Maven is a Software Project management

Project Management → manages build, Releases, document

Maven use CONVENTION over CONFIGURATION.
maven is structured so it follows the convention.

download Java

JAVA_HOME - Environment variable

download Maven & set

MAVEN_HOME

append JAVA_HOME & MAVEN_HOME path in the Path Environment variable

*) THE POM → Project Object Model
input file for maven

→ POM is a fundamental unit
→ Resides in the Base directory

*] POM-xml contains

- goals
- plugins
- Project version
- build profiles

→ minimum Requirements for POM

- 'Project Root'
- model version
- groupId
- artifactId
- version

GroupId → Refers to the group your Project Belongs.
Artifact Id → Name of your Project.
Version → Version of your Project

GID → com. companyname . Projectname
~~AID →~~
GID → com. cognizant . icici
AID → insurance 1·3·6·4-00
version → 13·0·0·2·8
<modelversion> 4·0·0 <-11-> → version of POM·xml.

Every dependency have - GID
AID
version no

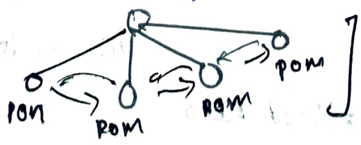
dependency is nothing but JAR files.

Build Section - have all the plugins needed

→ Everything Maven does is through Plugins

*] Super POM

→ ◦ Super POM → contains all the common directories dependencies. Super POM is mavens default POM..
all POM extends the Super POM



<Parent> - super POM

$\underbrace{\text{Super POM = effective POM = Parent POM}}_{\text{Same name}}$

$\boxed{\text{mvn help: effective-POM}}$ → check the default configuration in the POM file

$\boxed{\text{mvn Pluginname : Goalname}}$
↳ format to Run any Maven tool

Goals = Mojos

Plugin → Set of Goals.
Goal → Set of tasks.

Proj → Project variable (in POM.xml)

*] Property References → ${env}

*) Maven lifecycle

$$\begin{tikzcd}
\text{MLC} \arrow[r, dash] & \boxed{\text{Phases}} \arrow[r] \arrow[dr] \arrow[d] & \text{order in which does it build task} \\
& \boxed{\text{Plugins}} \arrow[d] & \text{Run in order} \\
& \boxed{\text{Goals}} \arrow[d] & \text{Plugins are Bound to Phases.} \\
& \boxed{\text{Tasks}} &
\end{tikzcd}$$

lifecycle → Sequence of Phases.

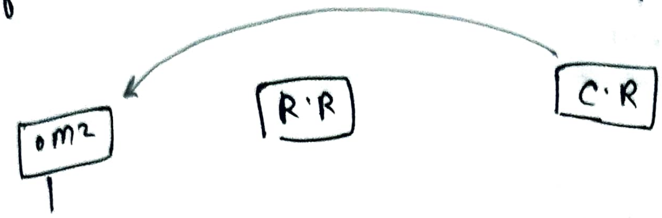
Maven has 3 standard lifecycle
- clean
- default (or build)
- site

*) MAVEN BUILD LIFECYCLE PHASES.

Prepare-Resources } - clean - clean your folder that contains classes.
Validate.
compile → compile the Java file.
test.
Package
install → .m2 → local REPO
deploy → Remote Repository

• Jar file is stored in .m2 folder
.m2 → local Repository

Remote Repo – A Repo maintained by your organization.
central Repo – the one on the internet, Public Repo
→ Jfrog, artifactory Repository, Nexus



.m2 → stores the info so we don't ~~me~~ need to access the C.R again & again

NEXUS
setting.xml → stores nexus connection information

.m2 > C.R > R.R – order of searching
L.R

*) using Maven Archetypes
*) Maven dependency mechanism
-CP (Class Path) → stores the dependant JAR | WAR
dependency → all dependent JAR are stored in POM.xml

*) Type of dependencies
1] Direct → build is directly dependent on a JAR
2] Transitive →
3] External →

d.
*) SnapShots & Releases.
SnapShots → deployment copy

Releases → Release is Production Ready copy.

Build Profile → set of element which allows you customize build for Particular Environment variable

POM have

- Project metadata
- Maven coordinates
- uniquely identifies components
→ group, name & version
- dependencies
- coordinates
- Build settings
- Java version
- artifact Name
- inheritance (sort of)

*) Super POM → that defines all default settings, which is automatically inherited by all of the Maven projects.

*) dependency management is the major advantage of using the maven
   ↳ dependencies can be downloaded from Maven Repo

$\boxed{\text{mvn compile}}$

Standards directory template → maven will look for files at a particular directory

src/test/Java → Junit
src/test/Resources → for unit test Resource file

$\boxed{\text{git Remote -v}}$ → git verbose action

*] PLUGINS

→ Maven = Plugin Engine
the plugin basically adds more functionality and adds more Goal

*) Plugins are dependencies

Some Plugins example
- compiling Source code
- Run unit tests
- Publish to artifact Repository
- Deploy to Remote Server
- Publish documentation

You can mention plugins in your Projects POM file.
→ change Plugin behaviour → Specify in POM configuration

Jar file ↑

~~target~~

$\boxed{\text{Java -cp } \overbrace{\text{maven-quick-start-1.0.jar}} \text{ clinic.Programming.training.Application.}}$

→ Start your Java application

*) Maven Supports 6 dependency Scopes
- compile
- Runtime
- Test
- Provided
- System
- import

i] compile Scope (default) → deals with compilation & execution
ii] Runtime Scope → deals with Deployment [Runtime & not Required for compilation
iii] Test Scope → used for dependencies that needed for unit testing
iv] Provided Scope → tell the maven, the dependencies will be Provided by the target Runtime or deployment system.
v] System Scope (~~Raro~~ Rarely used) → tells about local dependencies
vi] Import Scope → Rarely used, Almost never, used to import or Replace dependencies from other Projects.

Maven central → A place where you can find the dependencies you need to add to your project.

~~mvn depency~~

`mvn dependency:tree` → list off the dependencies that our project uses

dependency tree goal in dependency plugin

`mvn test` → To perform the unit testing

Reports of testing ⇒ Surefire-Reports

maven uses Surefire Plugin to do the unit testing

*) Archetype (Plugin) → can create maven Projects
↳ maven project templates we can use to start any Project

`mvn archetype:generate` → to use maven archetype

`maven-archetype` → Return a list of archetype