Maven is based on the concept of a project object model (POM).
Maven can manage a project's build, reporting and documentation from a central piece of information.
Maven we can build and manage any Java based project.
Maven provides developers ways to manage the following −

- Builds
- Documentation
- Reporting
- Dependencies
- SCMs
- Releases
- Distribution
- Mailing list

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**.
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	${**basedir**}/src/main/java
Resources	${**basedir**}/src/main/resources
Tests	${**basedir**}/src/test
Complied byte code	${**basedir**}/target
distributable JAR	${**basedir**}/target/classes

Maven pom.xml is also not required to be written manually. Maven provides numerous **archetype** plugins to create projects, which in order, create the project structure and pom.xml

A Build Lifecycle is a well-defined sequence of phases, which define the order in which the goals are to be executed. Here phase represents a stage in life cycle. As an example, a typical Maven Build Lifecycle consists of the following sequence of phases.

Phase	Handles	Description
prepare-resources	resource copying	Resource copying can be customized in this phase.
validate	Validating the information	Validates if the project is correct and if all necessary information is available.
compile	compilation	Source code compilation is done in this phase.
Test	Testing	Tests the compiled source code suitable for testing framework.
package	packaging	This phase creates the JAR/WAR package as mentioned in the packaging in POM.xml.
install	installation	This phase installs the package in local/remote maven repository.
Deploy	Deploying	Copies the final package to the remote repository.

There are always pre and post phases to register goals, which must run prior to, or after a particular phase.

When Maven starts building a project, it steps through a defined sequence of phases and executes goals, which are registered with each phase.
Maven has the following three standard lifecycles −

- clean
- default(or build)
- site

A goal represents a specific task which contributes to the building and managing of a project. It may be bound to zero or more build phases. A goal not bound to any build phase could be executed outside of the build lifecycle by direct invocation.
The order of execution depends on the order in which the goal(s) and the build phase(s) are invoked. For example, consider the command below.
The clean and package arguments are build phases while the dependency:copy-dependencies is a goal.

```
mvn clean dependency:copy-dependencies package
```

Here the *clean* phase will be executed first, followed by the dependency:copy-dependencies goal, and finally *package* phase will be executed.

Clean Lifecycle

When we execute *mvn post-clean* command, Maven invokes the clean lifecycle consisting of the following phases.

- pre-clean
- clean

- post-clean

Maven clean goal (clean:clean) is bound to the *clean* phase in the clean lifecycle. Its clean:cleangoal deletes the output of a build by deleting the build directory. Thus, when *mvn clean* command executes, Maven deletes the build directory.

Default (or Build) Lifecycle

This is the primary life cycle of Maven and is used to build the application. It has the following 21 phases.

Sr.No.	Lifecycle Phase & Description
1	**validate** Validates whether project is correct and all necessary information is available to complete the build process.
2	**initialize** Initializes build state, for example set properties.
3	**generate-sources** Generate any source code to be included in compilation phase.
4	**process-sources** Process the source code, for example, filter any value.
5	**generate-resources** Generate resources to be included in the package.
6	**process-resources** Copy and process the resources into the destination directory, ready for packaging phase.
7	**compile** Compile the source code of the project.
8	**process-classes** Post-process the generated files from compilation, for example to do bytecode enhancement/optimization on Java classes.
9	**generate-test-sources** Generate any test source code to be included in compilation phase.
10	**process-test-sources** Process the test source code, for example, filter any values.
11	**test-compile** Compile the test source code into the test destination directory.
12	**process-test-classes** Process the generated files from test code file compilation.
13	**test** Run tests using a suitable unit testing framework (Junit is one).
14	**prepare-package**

	Perform any operations necessary to prepare a package before the actual packaging.
15	**package** Take the compiled code and package it in its distributable format, such as a JAR, WAR, or EAR file.
16	**pre-integration-test** Perform actions required before integration tests are executed. For example, setting up the required environment.
17	**integration-test** Process and deploy the package if necessary into an environment where integration tests can be run.
18	**post-integration-test** Perform actions required after integration tests have been executed. For example, cleaning up the environment.
19	**verify** Run any check-ups to verify the package is valid and meets quality criteria.
20	**install** Install the package into the local repository, which can be used as a dependency in other projects locally.
21	**deploy** Copies the final package to the remote repository for sharing with other developers and projects.

There are few important concepts related to Maven Lifecycles, which are worth to mention −

- When a phase is called via Maven command, for example mvn compile, only phases up to and including that phase will execute.
- Different maven goals will be bound to different phases of Maven lifecycle depending upon the type of packaging (JAR / WAR / EAR).