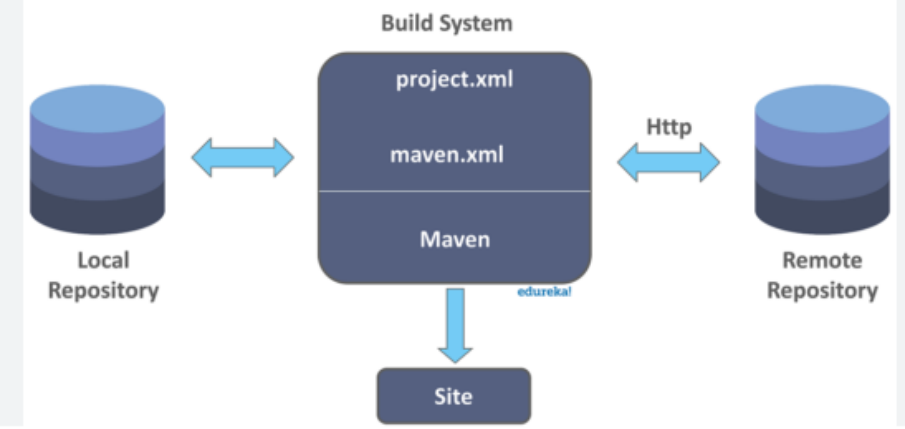
Maven is used for building code . To work in java project, we have to use third party libraries . Adding dependencies were difficult manually .
We can use maven to specify dependency in pom.xml .
Maven describe – how software is built and how to specify dependency .
Maven download dependency from maven central repo .



When we specify dependency in pom.xml maven will search that in central repo and maven will copy it to local repo if dependency is not present in central repo then it be taken from internet .
Maven lifecycle .

Clean Lifecyle
pre-clean
clean
post-clean

Default Lifecyle	
validate	test-compile
initialize	process-test-classes
generate-sources	test
process-sources	prepare-package
generate-resources	package
process-resources	pre-integration-test
compile	integration-test
process-classes	post-integration-test
generate-test-sources	verify
process-test-sources	install
generate-test-resources	deploy
processs-test-resources	

Site Lifecyle
pre-site
site
post-site
site-deploy

Default is main lifecycle it is responsible for project deployment . clean lifecycle is used the project and remove all files by previous build and site is used to create projects site document

Each lifecycle contain sequence of phases .
Default – 23 , clean – 3 , site – 4 .

- `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
- `deploy` - done in the build environment, copies the final package to the remote repository for sharing with other developers and projects.

These lifecycle phases (plus the other lifecycle phases not shown here) are executed sequentially to complete the `default` lifecycle. Given the lifecycle phases above, this means that when the default lifecycle is used, Maven will first validate the project, then will try to compile the sources, run those against the tests, package the binaries (e.g. jar), run integration tests against that package, verify the integration tests, install the verified package to the local repository, then deploy the installed package to a remote repository.

Phases are run in specific order eg if we run maven test so all preceding phases before maven test will also be executed

Each phase Is sequence of goal . and each goal is responsible for specific task when we run a phase all goal bound to that phase will also be executed in order

Phase	plugin:goal
`process-resources`	`resources:resources`
`compile`	`compiler:compile`
`process-test-resources`	`resources:testResources`
`test-compile`	`compiler:testCompile`
`test`	`surefire:test`
`package`	`ejb:ejb` *or* `ejb3:ejb3` *or* `jar:jar` *or* `par:par` *or* `rar:rar` *or* `war:war`
`install`	`install:install`
`deploy`	`deploy:deploy`

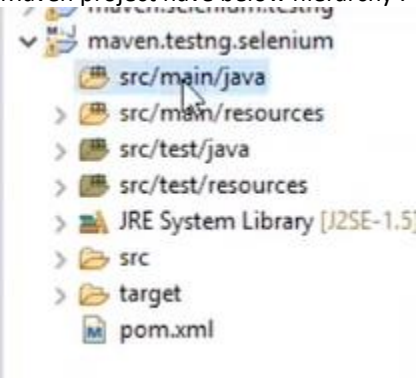
In above compiler is plugin and compile is phase , compiled goal from compiler plugin is bound to compile phase .similarly others also .

Maven plugin is group of goals , all execution in maven is done by plugin , plugin is mapped to goal and executed as part of it .

Phase is mapped to multiple goals and these goal are executed by plugin . We can directly invoke our specific goal while maven execution . a plugin configuration can be modified using plugin declaration .

Maven build project based on pom .
Group id – unique identifier that owns a project eg : com.edureka .
Artifact id – final name of compilation unit of project . eg maven.selenium.testng ,
Version – version of created artifact .
Packaging – jar / war / ear / pom

Maven project have below hierarchy .



Src/main/java – application code resides here .
Src/main/resources – all resoiurces required for developing code present here .
Src/test/java – for testing purpose code is written here .
Src/test/resources – resources for testing purpose resides here .

Pom.xml

A Project Object Model or POM is the fundamental unit of work in Maven. It is an XML file that contains information about the project and configuration details used by Maven to build the project. It contains default values for most projects. Examples for this is the build directory, which is `target`; the source directory, which is `src/main/java`; the test source directory, which is `src/test/java`; and so on.

When executing a task or goal, Maven looks for the POM in the current directory. It reads the POM, gets the needed configuration information, then executes the goal.

Some of the configuration that can be specified in the POM are the project dependencies, the plugins or goals that can be executed, the build profiles, and so on. Other information such as the project version, description, developers, mailing lists and such can also be specified.

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.

The minimum requirement for a POM are the following:

- `project` root
- `modelVersion` - should be set to 4.0.0
- `groupId` - the id of the project's group.
- `artifactId` - the id of the artifact (project)
- `version` - the version of the artifact under the specified group

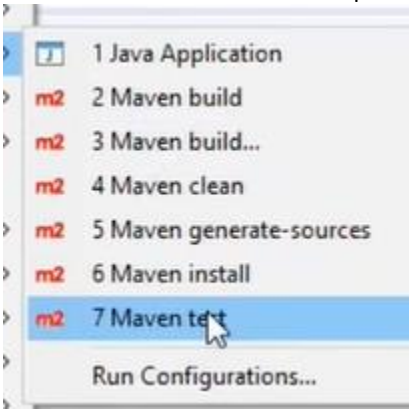
Here's an example:

```
1. <project>
2.   <modelVersion>4.0.0</modelVersion>
3.
4.   <groupId>com.mycompany.app</groupId>
5.   <artifactId>my-app</artifactId>
6.   <version>1</version>
7. </project>
```

Project – root element
We can add all extra dependency we use for our project in pom.xml .
We have to add all plugins that will be used for our project in pom.xml .

```
<build>
  <plugins>
   <plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-compiler-plugin</artifactId>
    <configuration>
     <source>1.8</source>
     <target>1.8</target>
    </configuration>
   </plugin>
   <plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-surefire-plugin</artifactId>
    <version>2.18.1</version>
    <configuration>
```

Eg
Now maven will download all dependency and build our project .



Commands : mvn clean , mvn test .