1. Maven Phase

A Maven phase represents a stage in the Maven build lifecycle. Each phase is responsible for a specific task.

Here are some of the most important phases in the *default* build lifecycle:

- *validate:* check if all information necessary for the build is available
- *compile:* compile the source code
- *test-compile:* compile the test source code
- *test:* run unit tests
- *package:* package compiled source code into the distributable format (jar, war, …)
- *integration-test:* process and deploy the package if needed to run integration tests
- *install:* install the package to a local repository
- *deploy:* copy the package to the remote repository

For the full list of each lifecycle's phases, check out the Maven Reference.

Phases are executed in a specific order. This means that if we run a specific phase using the command:

```
1   mvn <PHASE>
```

This won't only execute the specified phase but all the preceding phases as well.

For example, if we run the *deploy* phase – which is the last phase in the *default* build lifecycle – that will execute all phases before the *deploy* phase as well, which is the entire *default* lifecycle:

```
1   mvn deploy
```

2. Maven Goal

Each phase is a sequence of goals, and each goal is responsible for a specific task.

When we run a phase – all goals bound to this phase are executed in order.

Here are some of the phases and default goals bound to them:

- *compiler:compile* – the *compile* goal from the *compiler* plugin is bound to the *compile* phase
- *compiler:testCompile* is bound to the *test-compile* phase
- *surefire:test* is bound to *test* phase
- *install:install* is bound to *install* phase
- *jar:jar* and *war:war* is bound to *package* phase

We can list all goals bound to a specific phase and their plugins using the command:

```
mvn help:describe -Dcmd=PHASENAME
```

For example, to list all goals bound to the *compile* phase, we can run:

```
mvn help:describe -Dcmd=compile
```

And get the sample output:

```
compile' is a phase corresponding to this plugin:
org.apache.maven.plugins:maven-compiler-plugin:3.1:compile
```

Which, as mentioned above, means the *compile* goal from *compiler* plugin is bound to the *compile* phase.

3. Maven Plugin

A Maven plugin is a group of goals. However, these goals aren't necessarily all bound to the same phase.

For example, here's a simple configuration of the Maven Failsafe plugin which is responsible for running integration tests:

```

<build>
    <plugins>
        <plugin>
            <artifactId>maven-failsafe-plugin</artifactId>
            <version>${maven.failsafe.version}</version>
            <executions>
                <execution>
                    <goals>
                        <goal>integration-test</goal>
                        <goal>verify</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
    </plugins>
</build>

```

As we can see, the Failsafe plugin has two main goals configured here:

- *integration-test*: run integration tests
- *verify*: verify all integration tests passed

We can use the following command to **list all goals in a specific plugin**:

```
mvn <PLUGIN>:help
```

For example, to list all goals in the Failsafe plugin:

```
mvn failsafe:help
```

And the output of this will be:

```

This plugin has 3 goals:

failsafe:help
  Display help information on maven-failsafe-plugin.
  Call mvn failsafe:help -Ddetail=true -Dgoal=<goal-name> to display parameter
  details.

failsafe:integration-test
  Run integration tests using Surefire.

failsafe:verify
  Verify integration tests ran using Surefire.

```

To run a specific goal, without executing its entire phase (and the preceding phases) we can use the command:

```
mvn <PLUGIN>:<GOAL>
```

For example, to run *integration-test* goal from Failsafe plugin, we need to run:

```
mvn failsafe:integration-test
```

4. Building a Maven Project

To build a Maven project, we need to execute one of the life cycles by running one of their phases:

```
mvn deploy
```

This will execute the entire *default* lifecycle. Alternatively, we can stop at the *install* phase:

```
mvn install
```

But usually we'll use the command:

```
mvn clean install
```

To clean the project first – by running the *clean* lifecycle – before the new build.

We can also run only a specific goal of the plugin:

```
mvn compiler:compile
```