Lesson 15

13.03.2025

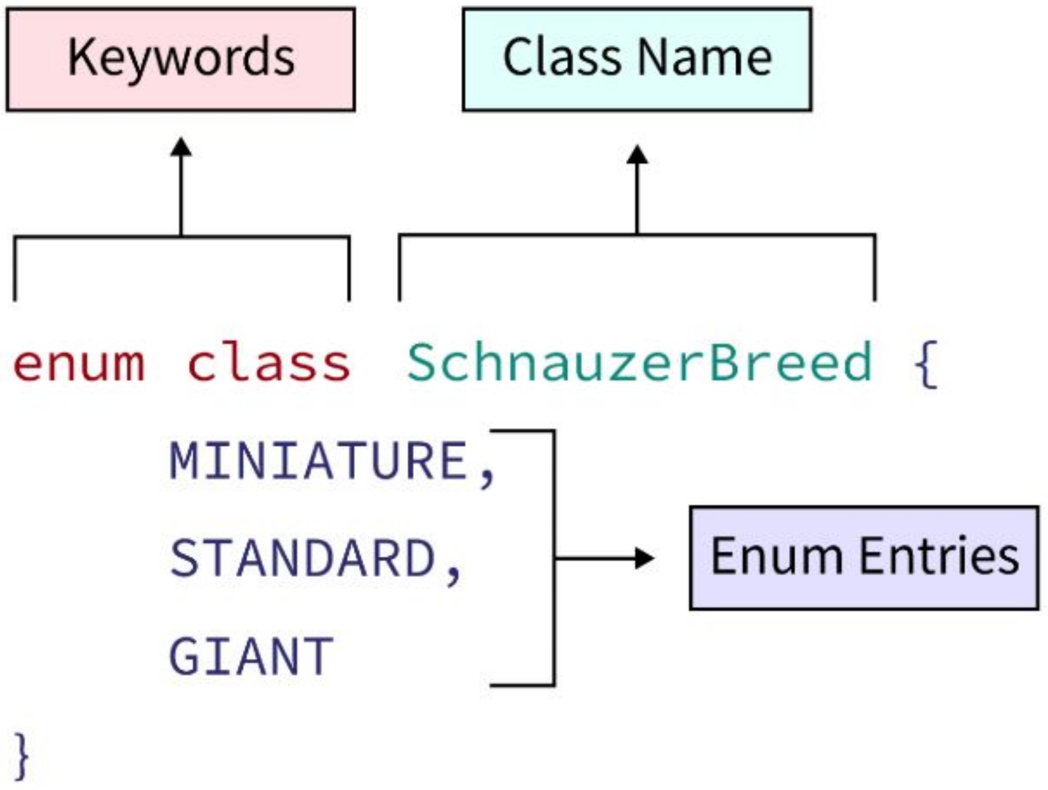
```java
public class Ex1 {
    public static void main(String[] args) {
        Integer i1 = Integer.valueOf(717);
        Integer i2 = Integer.valueOf(717);
        System.out.println(i1 == i2);
    }
}
```

```java
public class Ex2 {
    public static void main(String[] args) {
        String m = "Hello";
        System.out.print(m);
        bar(m);
        System.out.print(m);
    }

    static void bar(String m) {
        m += " World!";
    }
}
```

```java
public class Ex3 {
    public static void main(String[] args) {
        do {
            int count = 0;
            do {
                count++;
            } while (count < 2);
            break;
        } while (true);
        System.out.println(count);

    }
}
```

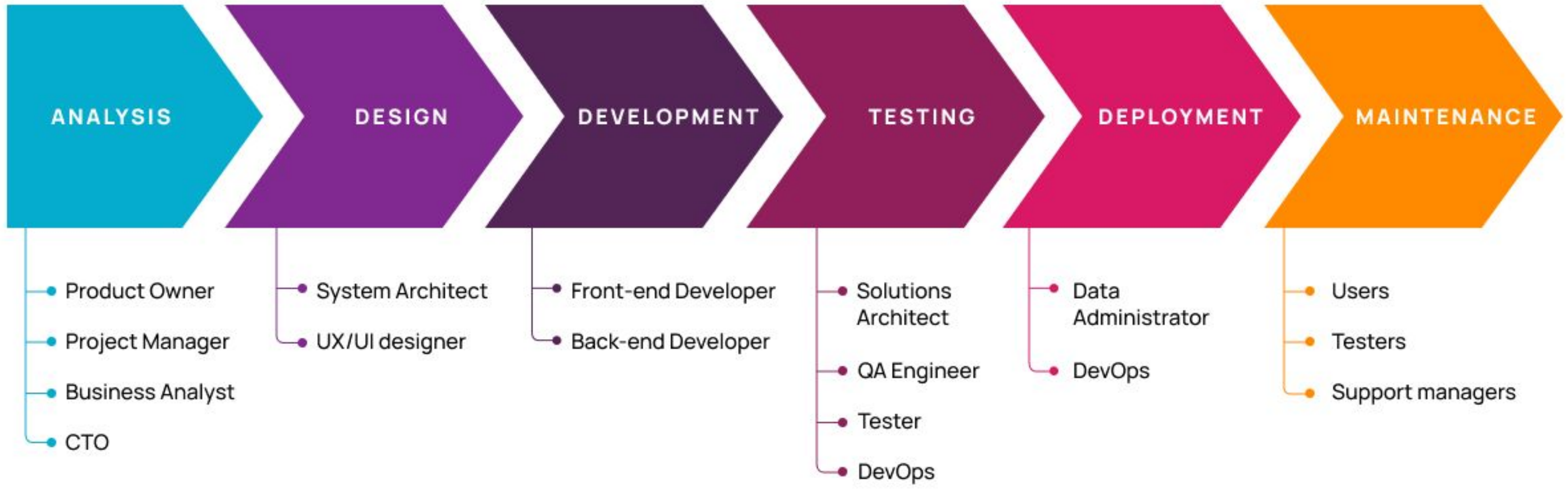
```java
public class Ex4 {
    public static void main(String[] args) {
        String tie = null;
        while (tie == null);
            tie = "shoelace";
            System.out.print(tie);
        }
}
```

```
public class Ex5 {
    public static void main(String[] args) {
        String[] nums = new String[] { "1", "9", "10" };
        Arrays.sort(nums);
        System.out.println(Arrays.toString(nums));
    }
}
```



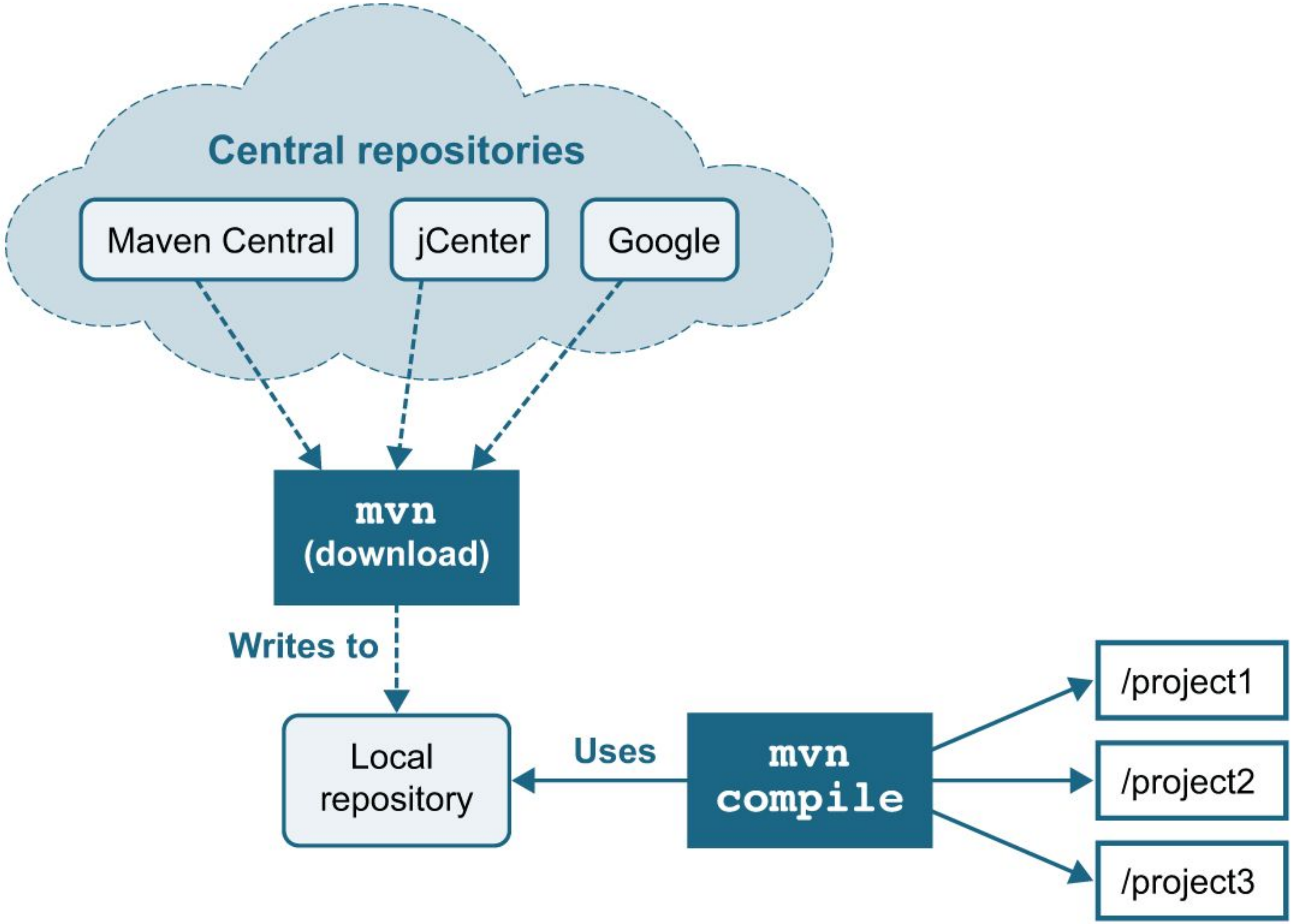
```
enum class SchnauzerBreed {
    MINIATURE,
    STANDARD,
    GIANT
}
```

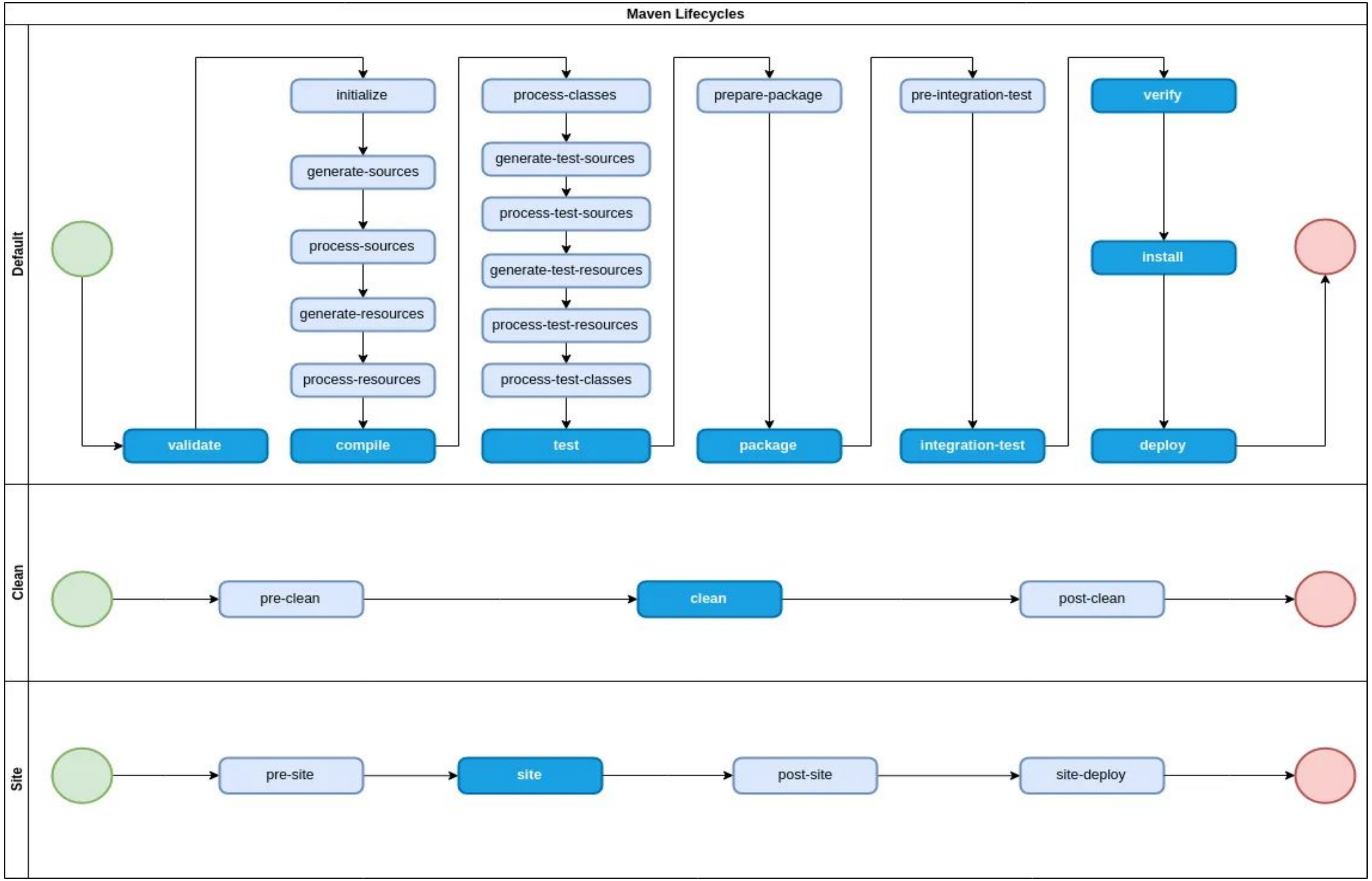
6 Phases of the Software Development Life Cycle

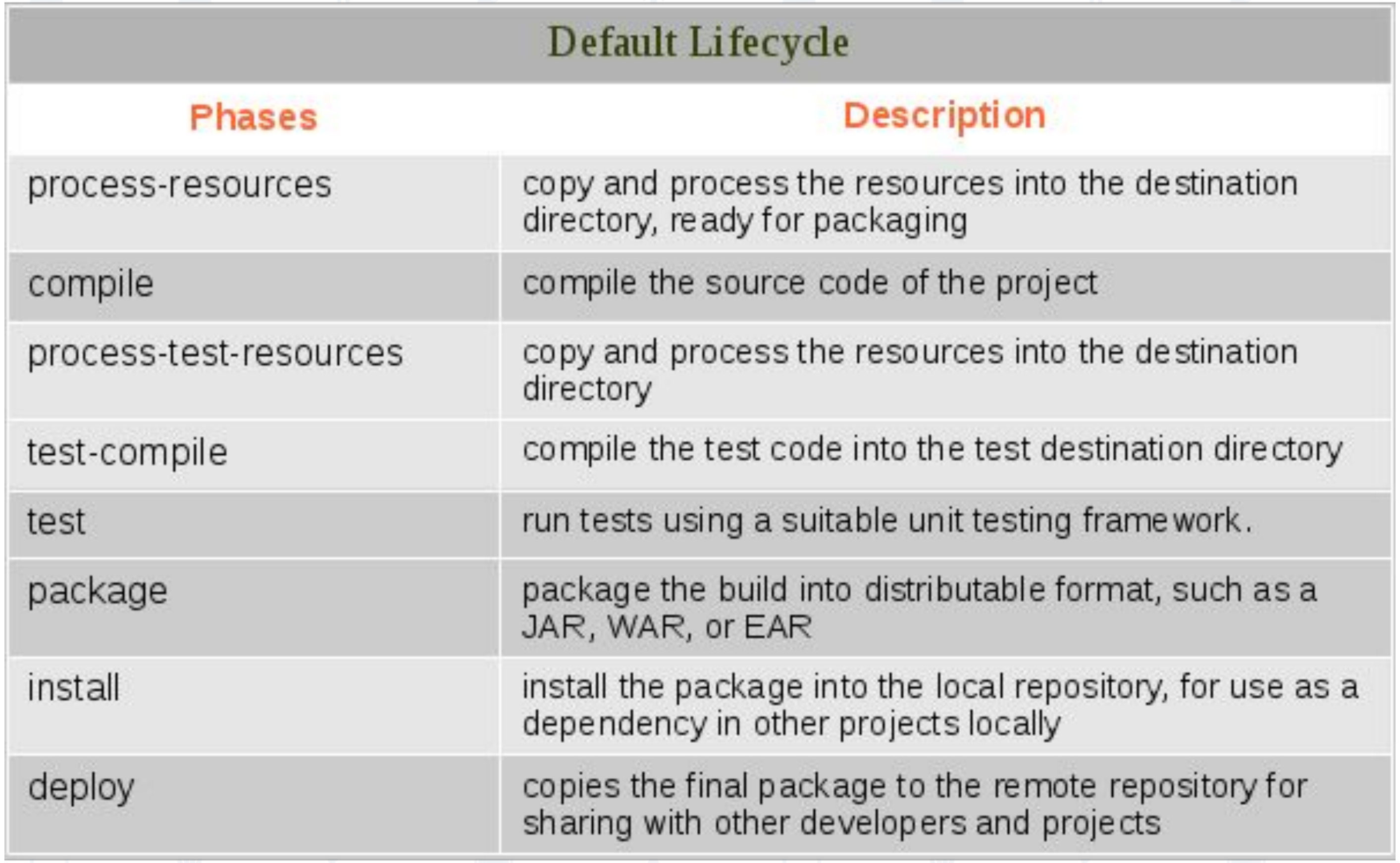


Gradle
Maven™
<APACHE ANT>

- Apache Ant http://ant.apache.org/
- Apache Maven https://maven.apache.org/
- Gradle https://gradle.org/

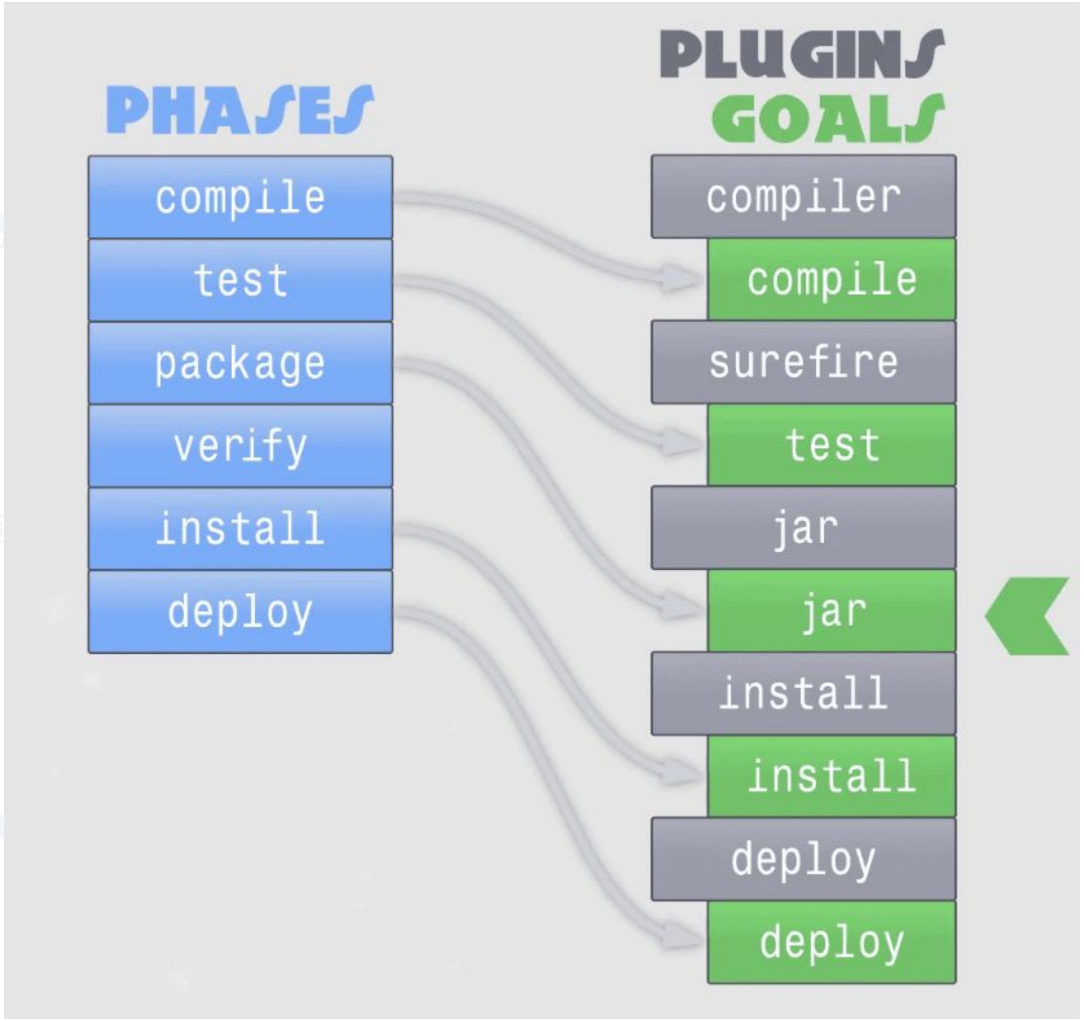

Central repositories
Maven Central
jCenter
Google
mvn (download)
Writes to
Local repository
Uses
mvn compile
/project1
/project2
/project3


Maven Lifecycles
Default
initialize
generate-sources
process-sources
generate-resources
process-resources
validate
compile
process-classes
generate-test-sources
process-test-sources
generate-test-resources
process-test-resources
process-test-classes
test
prepare-package
package
pre-integration-test
integration-test
verify
install
deploy
Clean
pre-clean
clean
post-clean
Site
pre-site
site
post-site
site-deploy



Default Lifecycle

Phases	Description
process-resources	copy and process the resources into the destination directory, ready for packaging
compile	compile the source code of the project
process-test-resources	copy and process the resources into the destination directory
test-compile	compile the test code into the test destination directory
test	run tests using a suitable unit testing framework.
package	package the build into distributable format, such as a JAR, WAR, or EAR
install	install the package into the local repository, for use as a dependency in other projects locally
deploy	copies the final package to the remote repository for sharing with other developers and projects


PLUGINS
GOALS
PHASES
compile
test
package
verify
install
deploy
compiler
compile
surefire
test
jar
jar
install
install
deploy
deploy

Maven Command	Description
mvn --version	Prints out the version of Maven you are running.
mvn clean	Clears the `target` directory into which Maven normally builds your project.
mvn package	Builds the project and packages the resulting JAR file into the `target` directory.
mvn package -Dmaven.test.skip=true	Builds the project and packages the resulting JAR file into the `target` directory - without running the unit tests during the build.
mvn clean package	Clears the `target` directory and Builds the project and packages the resulting JAR file into the `target` directory.
mvn clean package -Dmaven.test.skip=true	Clears the `target` directory and builds the project and packages the resulting JAR file into the `target` directory - without running the unit tests during the build.
mvn verify	Runs all integration tests found in the project.
mvn clean verify	Cleans the target directory, and runs all integration tests found in the project.
mvn install	Builds the project described by your Maven POM file and installs the resulting artifact (JAR) into your local Maven repository
mvn install -Dmaven.test.skip=true	Builds the project described by your Maven POM file without running unit tests, and installs the resulting artifact (JAR) into your local Maven repository
mvn clean install	Clears the `target` directory and builds the project described by your Maven POM file and installs the resulting artifact (JAR) into your local Maven repository
mvn clean install -Dmaven.test.skip=true	Clears the `target` directory and builds the project described by your Maven POM file without running unit tests, and installs the resulting artifact (JAR) into your local Maven repository
mvn dependency:copy-dependencies	Copies dependencies from remote Maven repositories to your local Maven repository.
mvn clean dependency:copy-dependencies	Cleans project and copies dependencies from remote Maven repositories to your local Maven repository.

mvn dependency:tree	Prints out the dependency tree for your project - based on the dependencies configured in the pom.xml file.
mvn dependency:tree -Dverbose	Prints out the dependency tree for your project - based on the dependencies configured in the pom.xml file. Includes repeated, transitive dependencies.
mvn dependency:tree -Dincludes=com.fasterxml.jackson.core	Prints out the dependencies from your project which depend on the com.fasterxml.jackson.core artifact.
mvn dependency:tree -Dverbose -Dincludes=com.fasterxml.jackson.core	Prints out the dependencies from your project which depend on the com.fasterxml.jackson.core artifact. Includes repeated, transitive dependencies.
mvn dependency:build-classpath	Prints out the classpath needed to run your project (application) based on the dependencies configured in the pom.xml file.

Maven cheat sheet

For more awesome cheat sheets visit rebellabs.org!

REBELLABS by ZEROTURNAROUND

Getting started with Maven

Create Java project

```
mvn archetype:generate
-DgroupId=org.yourcompany.project
-DartifactId=application
```

Create web project

```
mvn archetype:generate
-DgroupId=org.yourcompany.project
-DartifactId=application
-DarchetypeArtifactId=maven-archetype-webapp
```

Create archetype from existing project

```
mvn archetype:create-from-project
```

Main phases

clean — delete target directory
validate — validate, if the project is correct
compile — compile source code, classes stored in target/classes
test — run tests
package — take the compiled code and package it in its distributable format, e.g. JAR, WAR
verify — run any checks to verify the package is valid and meets quality criteria
install — install the package into the local repository
deploy — copies the final package to the remote repository

Useful command line options

-DskipTests=true compiles the tests, but skips running them

-Dmaven.test.skip=true skips compiling the tests and does not run them

-T - number of threads:
  -T 4 is a decent default
  -T 2C - 2 threads per CPU

-rf, --resume-from resume build from the specified project

-pl, --projects makes Maven build only specified modules and not the whole project

-am, --also-make makes Maven figure out what modules out target depends on and build them too

-o, --offline work offline

-X, --debug enable debug output

-P, --activate-profiles comma-delimited list of profiles to activate

-U, --update-snapshots forces a check for updated dependencies on remote repositories

-ff, --fail-fast stop at first failure

Essential plugins

Help plugin — used to get relative information about a project or the system.
mvn help:describe describes the attributes of a plugin
mvn help:effective-pom displays the effective POM as an XML for the current build, with the active profiles factored in.

Dependency plugin — provides the capability to manipulate artifacts.
mvn dependency:analyze analyzes the dependencies of this project
mvn dependency:tree prints a tree of dependencies

Compiler plugin — compiles your java code.
Set language level with the following configuration:

```
<plugin>
 <groupId>org.apache.maven.plugins</groupId>
 <artifactId>maven-compiler-plugin</
  artifactId>
 <version>3.6.1</version>
 <configuration>
   <source>1.8</source>
   <target>1.8</target>
 </configuration>
</plugin>
```

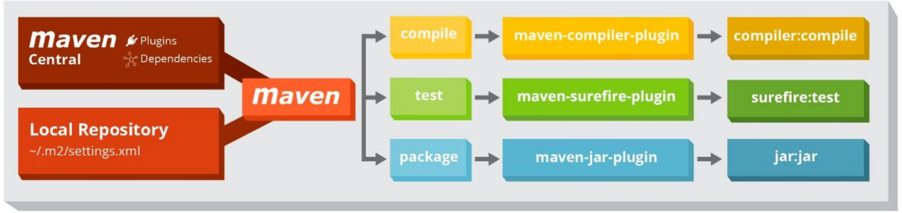
Version plugin — used when you want to manage the versions of artifacts in a project's POM.

Wrapper plugin — an easy way to ensure a user of your Maven build has everything that is necessary.

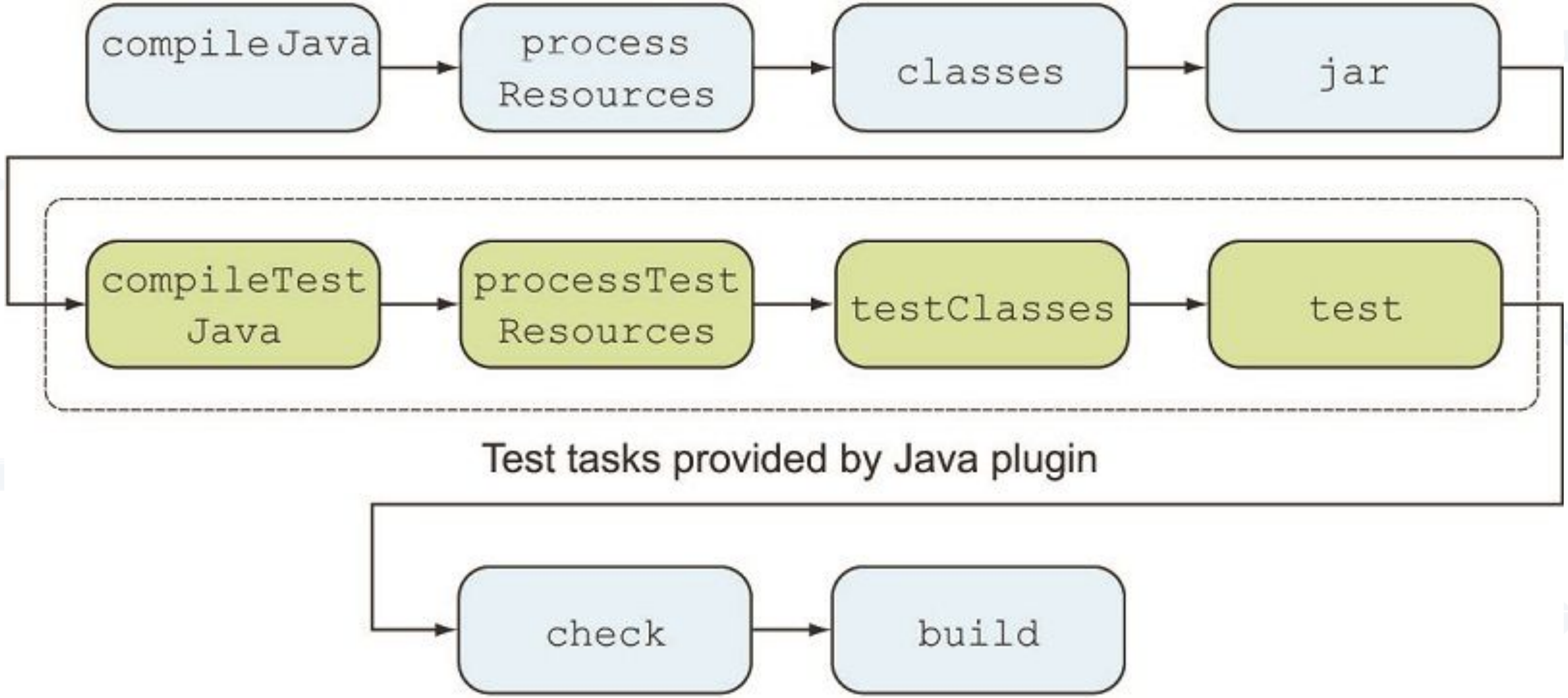
Spring Boot plugin — compiles your Spring Boot app, build an executable fat jar.

Exec — amazing general purpose plugin, can run arbitrary commands :)

The big picture



BROUGHT TO YOU BY JRebel


compileJava
processResources
classes
jar
compileTestJava
processTestResources
testClasses
test
Test tasks provided by Java plugin
check
build