Maven is one of the most popular build tools in the Java universe (others being Gradle or old-school Ant). You can not only build Java projects with it, but pretty much every project written in a JVM language like Kotlin or Scala, as well as other languages like C# and Ruby.

what *exactly* does a build tool do?

Dependency Management: Maven lets you easily include 3rd party dependencies (think libraries/frameworks such as Spring) in your project.

Compilation through convention: In theory you could compile big Java projects with a ton of classes, by hand with the javac command line compiler (or automate that with a bash script). This does however only work for toy projects. Maven expects a certain directory structure for your Java source code to live in and when you later do a `mvn clean install` , the whole compilation and packaging work will be done for you.

Maven can also run code quality checks, execute test cases and even deploy applications to remote servers, through plugins.

Pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"

         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">

    <modelVersion>4.0.0</modelVersion>

<parent> (1)

        <groupId>org.springframework.boot</groupId>

        <artifactId>spring-boot-starter-parent</artifactId>

        <version>2.3.0.RELEASE</version>

        <relativePath/> <!-- lookup parent from repository -->

</parent>



    <groupId>com.ameya</groupId>

    <artifactId>my-project</artifactId> (2)
```

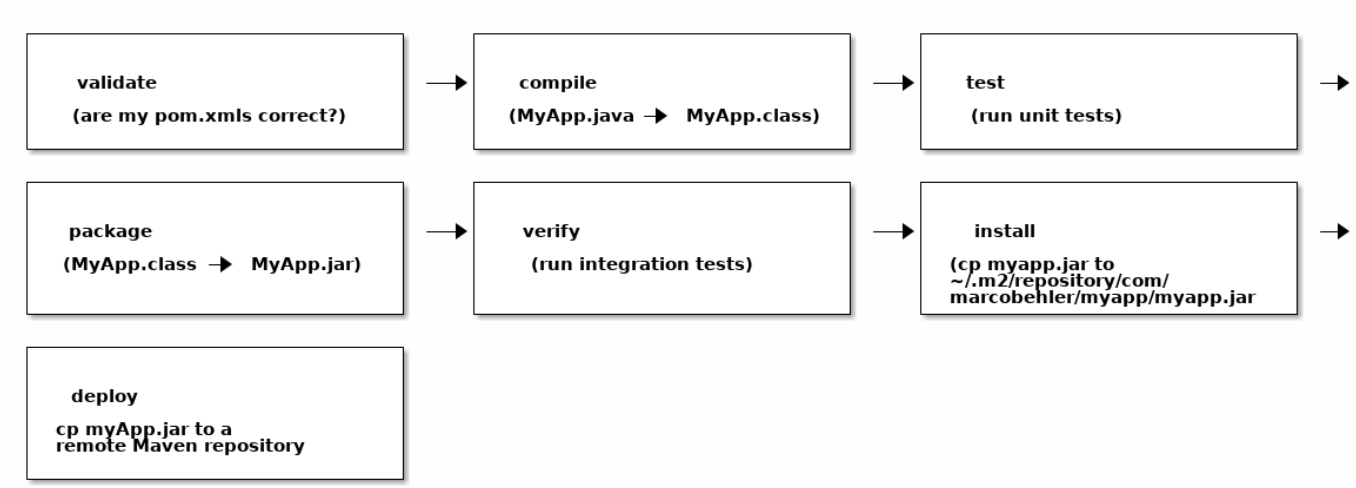
```
    <version>1.0-SNAPSHOT</version> (3)


    <properties>
        <maven.compiler.source>1.8</maven.compiler.source> (4)
        <maven.compiler.target>1.8</maven.compiler.target>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>


    <dependencies>
        <dependency> (5)
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
            <scope>test</scope>
        </dependency>
    </dependencies>
<build> (6)
        <plugins>
                <plugin>
                <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-maven-plugin</artifactId>
                        </plugin>
                </plugins>
</build>
</project>
```

The project structure will be like this

```
+ myproject
    + -- src
        + -- main
            + -- java
```

```
                MyApp.java

+ -- target

    + -- classes (after 'mvn compile')

         MyApp.class


    myproject.jar (upon mvn package or mvn install)


pom.xml
```

Maven Build Lifecycle Phases



These phases are sequential and depend on each other.

Example:

When you call `mvn deploy`, mvn will also execute *every* lifecycle phase before `deploy`, in order: `validate`, `compile`, `test`, `package`, `verify`, `install`.

Same for `verify`: `validate`, `compile`, `test`, `package`. Same for all other phases.

And as `clean` is not part of Maven's default lifecycle, you end up with commands like `mvn clean install` or `mvn clean package`. `Install` or `package` will trigger all preceding phases, but you need to specify clean in addition.