- How Spring Boot Works?
  - **dependencies** are nothing but **.jar** files which contains the *classes* that we want to use.
  - **maven** just download the **JAR**s, and puts them on the **compile + runtime classpath**.
  - **modules** are **JARs** with extra responsibilities.
    - spring-web, spring-context, spring-jdbc, spring-data-jpa
  - When you run your application, **JVM** searches for all the **.class** files inside the **classpath** which is provided by **maven** (I.e. **classpath** list is provided by Maven).
    - **target** folder is one of the classpaths; but it is not the only classpath;
    - **.jar** files present in the External Libraries are stored in **local Maven Cache**. Which is not inside the project.
      - ~/.m2/repository/org/springframework/spring-webmvc/6.1.x/spring-webmvc-6.1.x.jar
  - Then how **JVM** gets to know about the **.class** files that will be used?
    - **Maven** creates a **classpath** where all the list of paths are present.
    - **JVM** sees this and use those classes.
    - **External Libraries** are nothing but the **classpaths** that are not inside your project.
  - When you create **.jar** file of your application, the **self created classes** and the **external libraries classes** will stored separately.
    -
```
myapp.jar
  ├── BOOT-INF/classes   ← your classes
  ├── BOOT-INF/lib       ← dependency JARs
```
    - **classes** will contain the **self created classes**.
    - **lib** will contain the **external libraries classes**.
- **Maven** vs **JVM**
  - **maven** works at **build** time and **JVM** works at **run time**.
  - Initially **maven** is run when you trigger the commands like **mvn test**, **mvn compile**, **mvn package**, **mvn spring-boot:run** .
    - It downloads the dependencies after reading the **pom.xml** and stores them in ~/.m2/repository (*maven cache path*).
    - Then it **compiles** the **.java** files and convert those to **.class** files (bytecodes) *(if the command that was executed was **mvn compile**)*
    - Depending upon the maven command executed, it'll do the thing.

        - One important thing: **maven** is also written in **java**. So it also needs one **JVM** to run it. It is called **Maven JVM** (its not any special JVM, just normal JVM only)
        - Inside this Maven JVM, **maven modules** are being used, not **spring modules**.
        - After finishing its work, Maven creates files on disk (JAR, class files). Later, a separate JVM loads those files and runs the Spring Boot application..
    - Now **JVM** part comes.
        - **JVM** never reads **pom.xml**, it is just read by **maven** to download the dependencies and provide those to **JVM**.
        - **JVM** use those dependencies to run the application.
- There is one plugins that is present in the **pom.xml** which is **"maven-compiler-plugin"**.
    - **maven-compiler-plugin invokes javac behind the scenes, passing all the required flags, paths, and options.**
    - Needful flags means
        - -classpath → where dependencies are
        - -processorpath → where annotation processors (like Lombok) are
        - -source / -target (or --release) → Java version
        - -d target/classes → where compiled .class files go
        - list of .java source files
- **spring-boot-maven-plugin**
    - spring-boot-maven-plugin does NOT run your application logic.
    - It prepares and packages your Spring Boot application so it can be run easily by the JVM.
    - Without this the dependencies won't be there in the **.jar** file, you would have to copy and paste those dependencies to run the application.
    - Also it copies the **tomcat jars** into that.

- How Spring Boot Works
  - Dependencies are JAR files that contain reusable classes.
  - Maven reads pom.xml, downloads dependencies into ~/.m2/repository, and manages build steps.
  - Plugins are used by Maven to compile, test, package, and run the application.
  - Maven helps prepare classpath information, but the JVM actually uses the classpath to load .class files.
  - IntelliJ shows dependency JARs from Maven cache under External Libraries.
- Runtime Flow
  - JVM starts and executes main()
  - SpringApplication.run() processes @SpringBootApplication
  - Auto-configuration classes listed in the AutoConfiguration.imports file are considered
  - Beans are created only if conditions match, using conditional annotations
- Maven vs JVM
  - Maven works at build time
  - JVM works at run time
  - Maven downloads and prepares dependency JARs
  - JVM loads classes from those JARs and runs the application
- Plugins
  - maven-compiler-plugin
    - Invokes javac with required flags
    - Handles annotation processing (Lombok)
  - spring-boot-maven-plugin
    - Packages the app into an executable fat JAR
    - Bundles dependencies (including embedded Tomcat)
    - Makes java -jar app.jar possible

- In the **plugins** sections, under **maven-compiler-plugin**, there is something called **annotationProcesssorPaths**

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-compiler-plugin</artifactId>
    <configuration>
        <annotationProcessorPaths>
            <path>
                <groupId>org.projectlombok</groupId>
                <artifactId>lombok</artifactId>
            </path>
        </annotationProcessorPaths>
    </configuration>
</plugin>
```

  - **annotationProcessorPaths** tells javac which **JARs** contain **annotation processors** and should be loaded during **compilation**.
  - Lombok must be available on the annotation processor path for its annotations (@Getter, @Setter, etc.) to work.
  - Annotation processors are **compile-time** tools that analyze annotations and modify or generate code before **.class** files are created.
  - The order of entries in annotationProcessorPaths does not control execution order; processors do not run sequentially.
  -

- In case of **lombok** and **MapStruct**, why **lombok-mapstruct-binding** is needed?
  - Lombok does not generate new classes; it modifies existing classes at **compile time** by altering the compiler's internal representation (AST: Abstract Syntax Tree).
  - MapStruct generates new mapper classes and relies on the annotation-processing API to inspect methods.
  - Lombok's AST modifications are not fully visible to MapStruct by default.
  - **lombok-mapstruct-binding** acts as a bridge, making Lombok-generated getters and setters visible to MapStruct during annotation processing.
  - **This problem is about visibility, not execution order.**
  -

- Also you can see below, in this plugin **lombok** is included in the **excludes** list

```
<plugin>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-maven-plugin</artifactId>
    <configuration>
        <excludes>
            <exclude>
                <groupId>org.projectlombok</groupId>
                <artifactId>lombok</artifactId>
            </exclude>
        </excludes>
    </configuration>
</plugin>
```

- This is because, **lombok** doesn't provide any **.class** files that JVM needs to run the application.
- It just injects the **getters** and **setters** to the existing classes during **compile-time**.
- When you see the **.class** files, you'll see the **getters** and **setters** method's implementations there; **lombok** runs during **compilation-time**; as the **getters** and **setters** are already generated, hence there is no need ot including **lombok** in the **.jar** file.