Sr.No.	Topics	Duration(Mins)	Session No(2 Hours)	Session No.(4 Hours)
1	Overview of Various Build Tool	15	1	1
2	What is Maven?	15	1	1
3	Maven Architecture	30	1	1
4	Maven Plugins	30	1	1
5	Maven Archetypes	15	1	1
6	Maven Commands	30	1	1
7	Setting Up Maven Application	45	2	1
8	Practical	60	2	1

Understanding and Using Build Tool

1. Overview of Various Build Tool

Information

- Build tools are essential software programs used in software development to automate the compilation, testing, and packaging of code into a deployable application.
- They help streamline the build process and improve efficiency.

A. Apache Maven:

- Maven is a widely used build automation tool primarily used for Java projects.
- It uses XML-based configuration files (pom.xml) to define the project structure, dependencies, and build goals.
- Maven handles dependency management, compilation, testing, packaging, and deployment of Java applications.
- It follows a convention-over-configuration approach, providing a standardized project structure and build lifecycle.

B. Gradle:

- Gradle is a powerful build automation tool that supports multiple languages, including Java, Kotlin, Groovy, and more.
- It uses a Groovy-based DSL (Domain-Specific Language) or Kotlin scripts for defining the build configuration.
- Gradle allows for incremental builds, parallel execution, and efficient dependency management.

- It offers flexibility and extensibility, allowing developers to customize the build process to meet specific project requirements.

C. Apache Ant:

- Ant is a popular build tool primarily used for Java projects, offering flexibility and simplicity.
- It uses XML-based build scripts (build.xml) to define the build process and tasks.
- Ant provides built-in tasks for compilation, testing, packaging, and deployment of Java applications.
- It allows developers to define custom build tasks and supports easy integration with external tools and libraries.

D. Make:

- Make is a classic build tool widely used in Unix-like systems.
- It uses makefiles (typically named "Makefile") that define rules and dependencies to build a target.
- Make determines which parts of the code need to be rebuilt by tracking file timestamps and their dependencies.
- It is language-agnostic and can be used to build projects in various programming languages.

E. npm (Node Package Manager):

- npm is the default package manager for JavaScript and Node.js projects.
- It includes built-in functionality for project initialization, dependency management, and script execution.
- npm leverages a configuration file (package.json) to define project metadata, dependencies, and build scripts.
- It allows developers to define custom build scripts using the "scripts" section in package.json.

- These are just a few examples of popular build tools used in software development.
- Each build tool offers different features, focuses on different languages or platforms, and provides unique advantages. The choice of a build tool depends on project requirements, team preferences, and the ecosystem in which the project operates.

Building Application :

- Building an application refers to the process of transforming source code into a runnable and deployable form.
- It involves compiling source files, resolving dependencies, running tests, and packaging the application into a format suitable for deployment.
- The build process ensures that the application is correctly built and ready for execution.

A. Source Code Compilation:

- The first step is to compile the application's source code.
- This involves converting the human-readable source files (e.g., Java, C++, Python) into machine-executable code.
- The compiler checks for syntax errors, resolves references, and generates the compiled code or intermediate representation.

B. Dependency Management:

- Applications often rely on external libraries or modules, known as dependencies.
- Managing dependencies involves specifying the required libraries and their versions, ensuring they are available during the build process.
- A build tool or package manager (such as Maven, Gradle, or npm) is used to automatically download and manage dependencies.

C. **Testing:**

- Testing is an essential part of the build process to ensure the application functions as expected.
- Unit tests, integration tests, and other types of tests are executed to validate the behavior and correctness of the application.
- Test frameworks and tools are used to automate the testing process and report any failures or issues.

D. Packaging:

- Once the code is compiled and tested, the next step is to package the application for deployment.
- This involves bundling the compiled code, resources, and other required files into a distributable format.
- The packaging may differ depending on the application type, such as creating JAR files for Java applications or creating Docker images for containerized applications.

E. Build Automation:

- To streamline the build process and make it repeatable, build automation tools like Maven, Gradle, or Make are often used.
- These tools automate the execution of build tasks, including compilation, testing, and packaging.
- They provide configuration files (such as pom.xml for Maven or build.gradle for Gradle) to define the build steps and dependencies.

F. **Continuous Integration and Continuous Deployment (CI/CD):**

- In a CI/CD pipeline, building an application is an integral part. CI/CD tools, such as Jenkins, GitLab CI/CD, or CircleCI, automate the entire build process, including triggering builds upon code changes, running tests, and deploying the application to various environments.

- Building an application ensures that it is properly compiled, tested, and packaged, ready for deployment to various environments.
- It plays a crucial role in delivering reliable and functional software.

2. What is Maven?

Information

- Maven is a build automation and dependency management tool primarily used for Java projects.
- It provides a comprehensive framework for managing the entire software build lifecycle, including compiling source code, running tests, packaging, and deploying applications.
- Maven is widely adopted in the Java community and is known for its convention-over-configuration approach.

- **Key features and concepts of Maven:**

A. **Project Object Model (POM):**

- Maven uses a Project Object Model (POM) file, named pom.xml, to define the project structure, build configuration, and dependencies.
- The POM is an XML file that contains information about the project, such as its group ID, artifact ID, version, and dependencies.

B. Dependency Management:

- Maven simplifies the management of project dependencies by providing a centralized repository called Maven Central.
- Developers can declare dependencies in the POM file, specifying the artifact coordinates (group ID, artifact ID, version).
- Maven automatically downloads the required dependencies from the repository, resolving transitive dependencies as well.

C. **Build Lifecycle:**

- Maven defines a set of standard build phases, such as compile, test, package, install, and deploy.
- These build phases represent the different stages of the build process.
- Developers can bind plugins and execute tasks to specific build phases, enabling automation of various build activities.

D. Plugins:

- Maven offers a vast ecosystem of plugins that extend its functionality.
- Plugins are responsible for executing specific tasks during the build process, such as compiling code, running tests, generating documentation, and packaging applications. Developers can configure and customize plugins in the POM file.

E. Convention-over-Configuration:

- Maven follows a convention-over-configuration approach, where it enforces a standardized project structure and naming conventions.
- By adhering to these conventions, developers can benefit from automatic project configuration and streamlined build processes.
- However, Maven allows customization through configuration options for projects with non-standard structures.

F. **Maven Repository:**

- Maven maintains a local repository on the developer's machine to cache downloaded dependencies.
- It also allows the creation of private repositories for internal or proprietary libraries.
- Maven Central is the central repository accessible to all Maven users, hosting a vast collection of open-source libraries.

- Maven simplifies the build process, enhances collaboration, and promotes best practices in Java development.
- It provides a consistent and standardized approach for managing dependencies, facilitating project scalability and maintainability.
- Maven's rich ecosystem of plugins and its integration with popular development tools make it a preferred choice for many Java projects.

Benefits of Maven:

- Maven, as a build automation and dependency management tool, offers several benefits that contribute to improved software development and project management.

A. Dependency Management:

- Maven simplifies the management of project dependencies.
- It provides a centralized repository (Maven Central) that hosts a vast collection of open-source libraries.
- Developers can easily specify the required dependencies in the project's configuration file (pom.xml) using the artifact coordinates (group ID, artifact ID, version).
- Maven takes care of automatically downloading the necessary dependencies and their transitive dependencies, ensuring consistent and reliable builds.

B. Build Standardization:

- Maven promotes standardization and best practices in software builds.
- It follows a convention-over-configuration approach, providing a standardized project structure and build lifecycle.
- Developers can focus on writing code rather than spending time configuring and setting up the build process.
- Maven's predefined build phases and plugins ensure consistent build behaviors across different projects and teams, reducing build-related issues and increasing productivity.

C. **Build Automation:**

- Maven automates various aspects of the build process, reducing manual effort and enabling continuous integration and deployment practices.
- With Maven, developers can define build configurations and tasks in the project's configuration file.
- The build process can be easily triggered and executed using Maven commands or integrated into a continuous integration (CI) server.
- Automation improves build efficiency, enables faster feedback cycles, and promotes reliable and reproducible builds.

D. **Project Reusability:**

- Maven facilitates project reusability and modularity.
- By defining project dependencies and versioning in the POM file, developers can easily share and reuse their projects across different applications or teams.
- This promotes code reuse, reduces duplication, and simplifies project collaboration.
- Maven's dependency management ensures that all required dependencies are properly resolved and downloaded, making it easier to integrate external libraries and components into projects.

E. **Documentation Generation:**

- Maven includes plugins for generating project documentation, such as JavaDoc, Surefire reports, and more.
- Documentation can be automatically generated during the build process, providing up-to-date and consistent documentation for the project.
- This helps in maintaining project documentation and improves code understanding and collaboration.

F. **Extensibility and Customization:**

- Maven is highly extensible and customizable.
- It offers a vast ecosystem of plugins that can be easily integrated into the build process.

- Developers can write custom plugins or configure existing plugins to meet specific project requirements.
- This flexibility allows for tailoring the build process to project-specific needs, integrating additional tools, or performing custom actions during the build.

- By leveraging the benefits of Maven, developers can streamline their build process, improve project maintainability, and enhance collaboration within the development team.
- Maven's dependency management, build standardization, and automation capabilities contribute to increased productivity, reliable builds, and overall project success.

Features of Maven:

- Maven, a popular build automation and dependency management tool, offers several features that enhance the software development process.

A. Dependency Management:

- Maven simplifies the management of project dependencies.
- It provides a centralized repository (Maven Central) where developers can find and download a wide range of open-source libraries.
- Maven automatically resolves dependencies, including transitive dependencies, ensuring that all required dependencies are available during the build process.

B. Build Lifecycle and Phases:

- Maven defines a standard build lifecycle with predefined phases.
- The build lifecycle represents different stages of the software build process, such as compilation, testing, packaging, installation, and deployment.
- Developers can bind plugins and execute tasks to specific build phases, allowing for automated and consistent build processes.

C. Convention-over-Configuration:

- Maven follows a convention-over-configuration approach, providing a standardized project structure and build configuration.
- By adhering to these conventions, developers can benefit from automatic project configuration and reduced setup time.
- Maven's conventions enable consistent builds across different projects and teams, improving collaboration and code maintainability.

D. Build Profiles:

- Maven supports the concept of build profiles, which allow developers to define different configurations for various environments (e.g., development, testing, production).
- Build profiles can include specific dependencies, plugins, and build settings tailored to each environment.
- This flexibility ensures that the build process can be easily customized and adapted for different deployment scenarios.

E. Extensive Plugin Ecosystem:

- Maven has a vast ecosystem of plugins that extend its functionality.
- Plugins enable additional features and tasks, such as code compilation, testing, code analysis, documentation generation, packaging, and deployment.
- Developers can leverage existing plugins or create their own custom plugins to meet project-specific requirements.
- This extensibility allows for easy integration with other tools and the ability to customize the build process.

F. Scalability and Reusability:

- Maven supports project scalability and reusability.
- Developers can define project dependencies, versions, and configurations in the project's configuration file (pom.xml).

- This allows for easy sharing and reuse of project configurations across different applications or teams.
- Maven's dependency management ensures consistent and reliable builds, simplifying the integration of external libraries and components.

G. Project Reporting:

- Maven provides reporting capabilities that generate various reports on project metrics, test results, code coverage, and more.
- These reports help developers and project stakeholders gain insights into the project's health, quality, and progress.
- Maven's reporting functionality promotes transparency and facilitates informed decision-making during the software development process.

- Maven's features streamline the build process, simplify dependency management, and promote best practices in software development.
- Its convention-over-configuration approach, extensive plugin ecosystem, and support for project scalability contribute to improved productivity, maintainability, and collaboration within development teams.

3. Maven Architecture :

Information

- The Maven architecture consists of various components and concepts that work together to facilitate the build process and manage dependencies.

A. Project Object Model (POM):

- The Project Object Model (POM) is a fundamental concept in Maven.
- It is an XML file named pom.xml that resides in the project's root directory.
- The POM defines the project's structure, configuration, and dependencies.
- It includes information such as the project's group ID, artifact ID, version, build settings, and dependency declarations.

B. Repository:

- Maven relies on repositories to store and manage project dependencies.
- Repositories can be local or remote.
- The local repository, located on the developer's machine, caches downloaded dependencies and plugins.

- Remote repositories, such as Maven Central, contain a vast collection of publicly available libraries.
- Maven downloads dependencies from remote repositories and stores them in the local repository for future use.

C. Plugins:

- Plugins are essential components in Maven architecture.
- They extend Maven's functionality and provide specific features and tasks.
- Plugins are defined in the POM file, and Maven automatically downloads and executes the necessary plugins during the build process.
- Plugins can perform tasks such as compiling code, running tests, generating documentation, packaging applications, and deploying artifacts.

D. Build Lifecycle and Phases:

- The build lifecycle represents the various stages of the build process in Maven.
- It is divided into build phases, each representing a specific step in the lifecycle.
- The default build lifecycle consists of the following phases: validate, compile, test, package, verify, install, and deploy.
- Developers can customize the build process by binding plugins to specific build phases.

E. Goals:

- Goals are specific tasks performed by Maven plugins within a build phase.
- Each plugin defines one or more goals that can be executed to perform specific actions.
- For example, the compiler plugin provides the compile goal to compile source code, while the surefire plugin offers the test goal to execute tests.

- Developers can configure plugins and their goals in the POM file to meet project requirements.

F. Dependency Management:

- Maven simplifies dependency management by providing a centralized repository (Maven Central) and a consistent approach to declaring and resolving dependencies.
- Dependencies are declared in the POM file, specifying the artifact coordinates (group ID, artifact ID, version).
- Maven automatically downloads and resolves dependencies, including transitive dependencies, ensuring that all required libraries are available during the build process.

G. Profiles:

- Profiles in Maven allow for the customization of build settings for different environments or scenarios.
- They enable developers to define specific configurations, dependencies, and plugins based on conditions specified in the POM file or external properties files.
- Profiles provide flexibility to adapt the build process to different deployment environments or build variations.

- The Maven architecture promotes consistency, automation, and reusability in the build process.
- It simplifies dependency management, provides a standardized project structure, and supports customization through plugins and profiles.
- By leveraging the Maven architecture, developers can streamline the build process and efficiently manage project dependencies and configurations.

Local Vs Central Vs Remote Repository:

A. Local Repository:

- A local repository is a storage location on the developer's machine where Maven caches and stores downloaded artifacts (dependencies, plugins, and project-specific libraries).
- When Maven resolves and downloads artifacts for the first time, it stores them in the local repository.
- The local repository is specific to each developer's machine and is typically located in the .m2 directory in the user's home directory.
- The local repository allows Maven to work offline, as it can retrieve artifacts from the cache instead of downloading them again.

B. Central Repository:

- The Central Repository, also known as Maven Central, is a public repository maintained by the Maven community.
- It is a remote repository accessible to all Maven users.
- Maven Central hosts a vast collection of open-source libraries and artifacts that can be easily downloaded and used in Maven projects.
- When Maven needs to resolve an artifact that is not available in the local repository, it checks the Central Repository.
- If the artifact is found, it is downloaded and stored in the local repository for future use.

C. Remote Repository:

- A remote repository is any repository that is not the local repository and is hosted on a remote server.
- Remote repositories can be public or private and may serve specific purposes or contain organization-specific artifacts.
- Remote repositories are configured in the project's pom.xml file or in the user's Maven settings file (settings.xml).
- Maven can retrieve artifacts from remote repositories when they are not available in the local repository.

- Developers can configure multiple remote repositories to retrieve artifacts from different sources or custom repositories.

The key differences between the local, central, and remote repositories are:

a. Local Repository:
- It is specific to each developer's machine and stores artifacts downloaded for a particular project.
- It allows Maven to work offline and improves build performance by avoiding redundant downloads.

b. Central Repository:
- It is a public remote repository accessible to all Maven users.
- It contains a vast collection of open-source artifacts and libraries.
- Maven automatically checks the Central Repository for artifacts not found in the local repository.

c. Remote Repository:
- It can be any repository hosted on a remote server, whether public or private.
- Remote repositories can be configured in the project's pom.xml or the user's settings file.
- They allow developers to retrieve artifacts from sources other than the local or Central Repository, such as custom or organization-specific repositories.

- In summary, the local repository serves as a cache of artifacts downloaded for a specific project, while the Central Repository is a public repository containing a wide range of open-source artifacts.

- Remote repositories, on the other hand, can be public or private and provide flexibility to retrieve artifacts from various sources beyond the local or Central Repository.

4. Maven Plugin

   Information

- In Maven, plugins are key components that extend the functionality of the build process.
- They provide specific tasks and goals that can be executed during different phases of the build lifecycle.
- Plugins allow developers to automate various build-related activities, such as compiling code, running tests, packaging applications, generating documentation, and deploying artifacts.
- Maven has a vast ecosystem of plugins that cover a wide range of development tasks.

- Here are some important points about Maven plugins:

A. Plugin Configuration:

- Plugins are configured in the project's POM (Project Object Model) file.
- Developers can specify the plugins they want to use and configure their behavior by providing parameters, goals, and execution instructions.
- Plugin configuration is done within the <plugins> element in the POM file.

B. Lifecycle Bindings:

- Plugins can be bound to specific phases of the build lifecycle.
- This means that when a particular phase is executed, the associated plugins and their goals are automatically triggered.
- For example, the compile phase can be bound to the compiler plugin, which will compile the source code when the compile phase is executed.

C. Plugin Goals:

- Goals represent specific tasks that a plugin can perform.
- Plugins can have multiple goals, each serving a distinct purpose.
- For example, the compiler plugin has a compile goal to compile the source code and a testCompile goal to compile the test code.

D. Plugin Execution:

- Plugins can be executed during the build process by specifying the goal and the phase in which the goal should be executed.
- Maven executes plugins and their goals based on the order of the defined build phases and the specified goals within each phase.

E. Default Plugins:

- Maven comes with a set of default plugins that are automatically available without explicit configuration.
- These plugins handle common tasks such as compiling code, running tests, generating reports, packaging applications, and more.
- Examples of default plugins include the compiler plugin, surefire plugin for test execution, jar plugin for creating JAR files, and deploy plugin for deploying artifacts.

F. Custom Plugins:

- Maven allows developers to create their own custom plugins to extend the build process and perform specific tasks.
- Custom plugins can be developed using the Maven Plugin API and customized to meet specific project requirements.
- They can be used to integrate external tools, automate custom processes, or perform specialized build tasks.

- Plugins are a powerful feature of Maven that enable automation and customization of the build process.
- By leveraging existing plugins or creating custom plugins, developers can streamline the build process, improve productivity, and enhance the functionality of their Maven projects.

5. Maven Archetypes

Information

- Maven archetypes are project templates or blueprints that provide a predefined structure, configuration, and initial set of files for creating new Maven projects.
- Archetypes serve as a starting point for developers, helping them quickly set up new projects with the required dependencies and build configurations.
- Maven provides a wide range of archetypes for different project types, frameworks, and technologies.

A. Archetype Structure:

- An archetype defines the structure and contents of a new Maven project.
- It includes the project's directory structure, configuration files (such as pom.xml), and initial source code or resources.
- The archetype provides a baseline structure that developers can modify and expand upon to build their application.

B. Archetype Catalogs:

- Maven archetypes are organized into catalogs, which are XML files containing information about available archetypes.
- These catalogs are hosted by Maven repositories and can be accessed by developers to discover and use archetypes.

- The default catalog is the central Maven archetype catalog, but additional catalogs can be configured in Maven settings to access custom or external archetypes.

C. Project Generation:

- To create a new project from an archetype, developers use the Maven command-line interface or an IDE with Maven integration.
- They specify the desired archetype, provide project-specific details (such as group ID, artifact ID, version), and Maven generates the project structure and initial files based on the archetype.

D. Customizing Archetypes:

- Archetypes can be customized to suit specific project requirements.
- Developers can modify the archetype's files, configurations, and templates to align with their project's needs.
- Customized archetypes can be shared within teams or organizations to ensure consistent project structures and configurations.

E. Popular Archetypes:

- Maven offers a wide variety of archetypes catering to different project types and technologies.
- Some popular archetypes include:

i. 'maven-archetype-quickstart': Basic archetype for creating a simple Java project.
ii. 'maven-archetype-webapp': Archetype for creating a Java web application.
iii. 'maven-archetype-jar': Archetype for creating a Java library project.
iv. 'maven-archetype-springboot': Archetype for creating a Spring Boot application.
v. 'maven-archetype-android': Archetype for creating an Android application.

- Archetypes simplify the process of setting up new Maven projects by providing a standardized starting point.
- They help developers avoid repetitive configuration and ensure consistency across projects.
- With archetypes, developers can quickly bootstrap new projects and focus on the core development tasks without spending much time on project setup.

6. Maven Commands

Information

- Maven provides a set of commands that you can use to interact with your Maven projects.
- These commands are executed using the command-line interface or within an integrated development environment (IDE) with Maven integration.
- Here are some commonly used Maven commands:

i. mvn clean:

- Cleans the build directory by deleting the target folder, which contains the compiled classes, generated files, and artifacts from previous builds.
- This command ensures a clean build environment.

ii. mvn compile:

- Compiles the source code of the project.
- It compiles the Java source files and any other compiled resources, placing the compiled classes in the target directory.

iii. mvn test:

- Runs the project's unit tests.
- This command executes the test cases in the test source directory and reports the test results.

iv. mvn package:

- Packages the project into a distributable format.
- It creates the artifact, such as a JAR or WAR file, according to the project's packaging configuration.
- The packaged artifact is placed in the target directory.

v. mvn install:

- Installs the project's artifact into the local repository.
- It copies the packaged artifact to the local repository, making it available for other projects on the same machine to use as a dependency.

vi. mvn deploy:

- Deploys the project's artifact to a remote repository or artifact repository manager.
- It uploads the artifact to a remote repository so that it can be shared with other developers or used in a production environment.

vii. mvn clean install:

- Performs a clean build and installs the artifact to the local repository in a single command.
- This command is often used to ensure a clean build and make the latest version of the artifact available locally.

viii. mvn dependency:tree:

- Displays the dependency tree of the project.
- It shows the hierarchical structure of dependencies, including the transitive dependencies, helping to identify potential conflicts or issues.

ix. mvn archetype:generate:

- Generates a new Maven project based on an archetype.
- This command prompts for input to select the archetype, project details (such as group ID, artifact ID, and version), and creates the project structure.

x. mvn help:

- Displays the list of available Maven commands and provides information about a specific command or plugin.

- It is a useful command to get help and learn more about Maven.

- These are just a few examples of commonly used Maven commands.
- Maven provides a wide range of commands and options to support various build and project management tasks.
- You can explore more commands and their options in the Maven documentation or by running mvn --help to get a complete list of available commands.

7. Setting Up Maven Application

Information

Installation of Maven and setting up the Build Path :

- To install Maven and set up the build path, you need to follow these steps:

A. Install Maven:

- Download the Maven distribution from the Apache Maven website (https://maven.apache.org/download.cgi).
- Extract the downloaded archive to a directory of your choice.
- Add the bin directory of the extracted Maven distribution to your system's PATH variable.

i. On Windows:

- Open the System Properties dialog.
- Go to the "Advanced" tab and click on "Environment Variables."
- In the "System variables" section, select the "Path" variable and click "Edit."
- Add the path to the Maven bin directory (e.g., C:\apache-maven-3.8.4\bin) at the end of the variable value. Ensure to separate it from the existing entries with a semicolon (;).
- Click "OK" to save the changes.
- Open a new command prompt and type mvn --version to verify that Maven is correctly installed and the path is set up.

ii. On macOS or Linux:

- Open a terminal.
- Run the following command to open the .bash_profile file (or .zshrc if using Zsh):

```
nano ~/.bash_profile
```

- Add the following line at the end of the file:

```
export PATH="/path/to/apache-maven-3.8.4/bin:$PATH"
```

- Replace '/path/to' with the actual path to the Maven bin directory.
- Press Ctrl + X, then Y, and Enter to save the changes and exit the editor.
- Restart the terminal or run source ~/.bash_profile (or source ~/.zshrc) to apply the changes.
- Type mvn --version in the terminal to verify that Maven is correctly installed and the path is set up.

B. Set Up the Build Path in Your IDE:

- If you are using an Integrated Development Environment (IDE) such as Eclipse, IntelliJ IDEA, or NetBeans, you need to configure the build path to include the Maven dependencies.
- Open your IDE and open the project you want to configure.
- Locate the build path settings in your IDE. This is typically found in the project settings or properties.
- Add the Maven dependencies to the build path:

i. In Eclipse:

- Right-click on the project and select "Properties."
- Go to "Java Build Path" > "Libraries" tab.

- Click on "Add Library" > "Maven Managed Dependencies" and click "Finish."
- Click "Apply" or "OK" to save the changes.

ii. In IntelliJ IDEA:

- Right-click on the project and select "Open Module Settings."
- Go to "Libraries" and click on the "+" button to add a new library.
- Select "From Maven..." and search for your project's Maven dependencies.
- Select the desired dependencies and click "OK."
- Click "Apply" or "OK" to save the changes.

iii. In NetBeans:

- Right-click on the project and select "Properties."
- Go to "Libraries" > "Compile" tab.
- Click on "Add Dependency" and search for your project's Maven dependencies.
- Select the desired dependencies and click "Add Dependency."
- Click "OK" to save the changes.
- With Maven installed and the build path set up in your IDE, you can start using Maven to manage dependencies and build your projects effectively.

Setting up Maven Application :

- To set up a Maven application, you need to follow these steps:

i. Install Maven:

- Download the Maven distribution from the Apache Maven website (https://maven.apache.org/download.cgi).
- Extract the downloaded archive to a directory of your choice.
- Add the bin directory of the extracted Maven distribution to your system's PATH variable.

ii. Create a Maven Project:

- Open a command-line interface or terminal.
- Navigate to the directory where you want to create your Maven project.
- Run the following command to create a new Maven project:

```
mvn archetype:generate -DgroupId=com.example
-DartifactId=my-app
-DarchetypeArtifactId=maven-archetype-quickstart
-DinteractiveMode=false
```

- This command creates a new Maven project using the maven-archetype-quickstart archetype with the specified Group ID (com.example) and Artifact ID (my-app).
- Once the command completes, you will have a new directory named my-app (or the specified Artifact ID) containing the project structure and initial files.

iii. Build and Test the Project:

- Navigate to the project's directory:

```
cd my-app
```

- Build the project using the following command:

```
mvn clean install
```

- This command will compile the source code, run the tests, and package the project into a JAR file.
- The JAR file will be located in the target directory.
- Verify that the build was successful without any test failures or errors.

iv. Import the Project into an IDE (Optional):

- If you are using an Integrated Development Environment (IDE) such as Eclipse, IntelliJ IDEA, or NetBeans, you can import the Maven project into your IDE for easier development and management.
- Open your IDE and choose the option to import an existing Maven project.
- Select the directory where your Maven project is located (my-app in this example).
- Follow the prompts and let the IDE configure the project based on the Maven settings.

v. Start Developing Your Application:

- Open the project in your preferred IDE and start writing your application code.
- Add dependencies to your project by specifying them in the project's pom.xml file. Maven will automatically download the required dependencies from the Maven Central Repository.
- Configure the build process, plugins, and additional settings in the pom.xml file as needed.

- By following these steps, you can set up a Maven application and start developing your project with the benefits of Maven's build and dependency management capabilities.

Building and Running Console based Application and Dynamic Web Application :

- To build and run a console-based application and a dynamic web application using Maven, follow these steps:

  A. Building and Running a Console-based Application:

  i. Create a Maven Project:

- Use the Maven command-line tool or an IDE to create a new Maven project.
- For example, in the command-line interface, navigate to the desired directory and run:

```
mvn archetype:generate -DgroupId=com.example
-DartifactId=my-console-app
-DarchetypeArtifactId=maven-archetype-quickstart
-DinteractiveMode=false
```

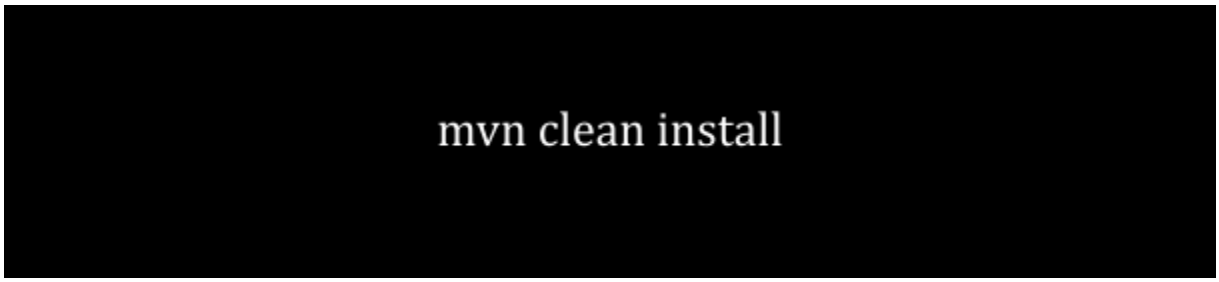
- This command creates a new Maven project named "my-console-app" in the specified group ID ("com.example") using the "maven-archetype-quickstart" archetype.

  ii. Write Console Application Code:

- Open the project in your preferred IDE or a text editor.
- In the src/main/java directory, you'll find the generated Java source file App.java. Replace its contents with your console application code.
- Write the necessary logic and functionality for your console-based application.

iii. Build the Console Application:

- Open a command-line interface or terminal.
- Navigate to the root directory of your Maven project (my-console-app in this example).
- Run the following command to build the application:



- Maven will compile your source code, run any tests, and package the application into a JAR file.

iv. Run the Console Application:

- After a successful build, navigate to the target directory within your project.
- Run the JAR file using the 'java' command, providing the fully qualified name of the main class:

- Replace my-console-app-1.0-SNAPSHOT.jar with the actual JAR file name generated by Maven.

B. Building and Running a Dynamic Web Application:

i. Create a Maven Project:

- Use the Maven command-line tool or an IDE to create a new Maven project.
- For example, in the command-line interface, navigate to the desired directory and run:

```
mvn archetype:generate -DgroupId=com.example
-DartifactId=my-web-app
-DarchetypeArtifactId=maven-archetype-webapp
-DinteractiveMode=false
```

- This command creates a new Maven project named "my-web-app" in the specified group ID ("com.example") using the "maven-archetype-webapp" archetype.

ii. Write Dynamic Web Application Code:

- Open the project in your preferred IDE or a text editor.
- Add the necessary HTML, CSS, JavaScript, and server-side code to the appropriate directories (src/main/webapp) for your dynamic web application.

iii. Build the Web Application:

- Open a command-line interface or terminal.

- Navigate to the root directory of your Maven project (my-web-app in this example).
- Run the following command to build the application:

```
mvn clean install
```

- Maven will compile your source code and package the web application into a WAR (Web Application Archive) file.

iv. Deploy and Run the Web Application:

- After a successful build, the generated WAR file will be located in the target directory within your project.
- Deploy the WAR file to a web server or application server of your choice (e.g., Apache Tomcat, Jetty).
- Start the web server or application server.
- Access the deployed web application using the appropriate URL in a web browser.

- By following these steps, you can build and run both console-based applications and dynamic web applications using Maven.
- Maven simplifies the build and dependency management process, allowing you to focus on writing your application's logic and functionality.

Spring boot application using Spring Initializr :

- To create a Spring Boot application using the Spring Initializr, follow these steps:

a. Visit the Spring Initializr website:
- Go to https://start.spring.io/ in your web browser.

b. Configure the project:
- Group: Enter the package name for your application (e.g., com.example).
- Artifact: Enter the name of your application (e.g., my-spring-boot-app).
- Dependencies: Select the desired dependencies for your application. You can choose from a wide range of options, such as Web, Data JPA, Security, etc., depending on your project requirements.

c. Generate the project:
- Once you have configured the project, click the "Generate" button.
- The Spring Initializr will create a ZIP file containing the generated project.

d. Download and extract the project:

- Download the ZIP file and extract its contents to a directory of your choice.

e. Import the project into your IDE:
- Open your preferred Integrated Development Environment (IDE) such as IntelliJ IDEA, Eclipse, or Visual Studio Code.
- Import the project as a Maven project by selecting the extracted project directory.
- Let the IDE import the project and set up the necessary dependencies.

f. Start developing:

- Once the project is imported, you can start developing your Spring Boot application.
- The main application class is typically named 'Application.java' and resides in the root package.
- Add your application logic, controllers, services, and other components as needed.

g. Build and run the application:

- Use your IDE's build tools or Maven commands to build the application.
- Run the main application class to start the Spring Boot application.
- By default, the application will start an embedded web server (usually Tomcat or Jetty) and listen on a specific port.

h. Test the application:

- Access the application's endpoints using a web browser or API testing tool.
- Verify that the application is running correctly and responding to requests.

- The Spring Initializr simplifies the process of setting up a Spring Boot application by generating the initial project structure and configuring the necessary dependencies.
- It provides a streamlined way to bootstrap a Spring Boot project, allowing you to focus on building your application's functionality.

Practical :

1. Creation of simple Java application using Maven

- To create a simple Java application using Maven, follow these steps:

a. Install Maven:

- Ensure that Maven is installed on your system. You can download Maven from the Apache Maven website (https://maven.apache.org/download.cgi) and follow the installation instructions.

b. Create a new Maven project:

- Open a command-line interface or terminal.
- Navigate to the directory where you want to create your project.
- Run the following command to create a new Maven project:

```
mvn archetype:generate -DgroupId=com.example
-DartifactId=my-java-app
-DarchetypeArtifactId=maven-archetype-quickstart
-DinteractiveMode=false
```

- This command creates a new Maven project with the Group ID ‘com.example’, Artifact ID my-java-app, and uses the ‘maven-archetype-quickstart’ archetype.

c. Structure and Code:

- Once the project is created, navigate into the project's directory:

```
cd my-java-app
```

- Open the project in your preferred IDE or a text editor.
- You'll find the source code directory structure under 'src/main/java'.
- Modify the' App.java' file to write your Java code. This is the entry point of your application.
- Add any additional Java classes or packages as needed.

d. Build the application:

- In the command-line interface or terminal, ensure you are in the root directory of your project (my-java-app).
- Run the following command to build the application:

```
mvn clean install
```

- Maven will compile your Java code and package the application into a JAR file.
- The JAR file will be located in the 'target' directory.

e. Run the application:

- After a successful build, navigate to the ‘target’ directory within your project.
- Run the JAR file using the ‘java’ command, providing the fully qualified name of the main class:

```
java -cp my-java-app-1.0-SNAPSHOT.jar
com.example.App
```

- Replace ‘my-java-app-1.0-SNAPSHOT.jar’ with the actual JAR file name generated by Maven.
- By following these steps, you can create a simple Java application using Maven.
- Maven helps in managing the build process and dependencies, allowing you to focus on writing the application logic.

2. Creation of Java Web application using Maven

   - To create a Java web application using Maven, follow these steps:

   a. Install Maven:
   - Ensure that Maven is installed on your system. You can download Maven from the Apache Maven website (https://maven.apache.org/download.cgi) and follow the installation instructions.

b. Create a new Maven project:

- Open a command-line interface or terminal.
- Navigate to the directory where you want to create your project.
- Run the following command to create a new Maven project:

```
mvn archetype:generate -DgroupId=com.example
-DartifactId=my-web-app
-DarchetypeArtifactId=maven-archetype-webapp
-DinteractiveMode=false
```

This command creates a new Maven project with the Group ID 'com.example', Artifact ID 'my-web-app', and uses the 'maven-archetype-webapp' archetype.

c. Structure and Code:

- Once the project is created, navigate into the project's directory:

```
cd my-web-app
```

- Open the project in your preferred IDE or a text editor.
- You'll find the source code directory structure under 'src/main/webapp' for web-related files (HTML, CSS, JavaScript, etc.).
- Modify or add web-related files to build your Java web application.
- Java code can be added to the 'src/main/java' directory. You can create servlets, controllers, filters, etc., based on your web application's needs.

d. Build the application:

- In the command-line interface or terminal, ensure you are in the root directory of your project (my-web-app).
- Run the following command to build the application:

```
mvn clean install
```

- Maven will compile your Java code and package the web application into a WAR (Web Application Archive) file.
- The WAR file will be located in the target directory.

e. Deploy and run the application:

- Deploy the generated WAR file to a web server or application server of your choice, such as Apache Tomcat or Jetty.

- Start the web server or application server.
- Access the deployed web application using the appropriate URL in a web browser.
- By following these steps, you can create a Java web application using Maven.
- Maven helps in managing the build process and dependencies, allowing you to focus on developing the web application's functionality and user interface.

3. Creation of Java Spring Boot Microservice using Maven

- To create a Java Spring Boot microservice using Maven, follow these steps:

a. Install Maven:
- Ensure that Maven is installed on your system. You can download Maven from the Apache Maven website (https://maven.apache.org/download.cgi) and follow the installation instructions.

b. Create a new Maven project:
- Open a command-line interface or terminal.
- Navigate to the directory where you want to create your project.
- Run the following command to create a new Maven project:

```
mvn archetype:generate -DgroupId=com.example
-DartifactId=my-microservice
-DarchetypeArtifactId=maven-archetype-quickstart
-DinteractiveMode=false
```

- This command creates a new Maven project with the Group ID com.example, Artifact ID my-microservice, and uses the maven-archetype-quickstart archetype.

c. Add Spring Boot dependencies:

- Open the project in your preferred IDE or a text editor.
- Open the pom.xml file in the root directory of your project.
- Add the necessary Spring Boot dependencies to the <dependencies> section.
- For example, you can add the following dependencies for a basic Spring Boot microservice:

```
<dependencies>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-web</artifactId>
</dependency>
</dependencies>
```

- You can add more dependencies depending on the requirements of your microservice, such as database connectivity, security, etc.

d. Write microservice code:

- Create the necessary Java classes in the src/main/java directory.
- Write your microservice logic using Spring Boot annotations and features.
- For example, you can create a 'RestController' class with '@RequestMapping' annotations to define your API endpoints.

e. Build and run the microservice:

- In the command-line interface or terminal, ensure you are in the root directory of your project (my-microservice).
- Run the following command to build the microservice:

```
mvn clean install
```

- Maven will compile your Java code, resolve dependencies, and package the microservice into a JAR file.

f. Run the microservice:

- After a successful build, navigate to the target directory within your project.
- Run the JAR file using the java command:

```
java -jar my-microservice-1.0-SNAPSHOT.jar
```

- Replace 'my-microservice-1.0-SNAPSHOT.jar' with the actual JAR file name generated by Maven.

g. Test the microservice:

- Access the endpoints defined in your microservice using tools like cURL or Postman.
- Verify that the microservice is running correctly and responding to requests.
- By following these steps, you can create a Java Spring Boot microservice using Maven.
- Maven simplifies the build and dependency management process, while Spring Boot provides a powerful framework for developing microservices.

4. Maven Command Demonstration to Build, Test and Package the projects

- Certainly! Here are some commonly used Maven commands to build, test, and package projects:

a. Clean the project:

```
mvn clean
```

- This command cleans the project by deleting the 'target' directory and removing any compiled artifacts and generated files.

b. Compile the project:

```
mvn compile
```

- This command compiles the source code of the project, including Java source files and other compiled resources.
- The compiled classes are placed in the 'target/classes' directory.

c. Run tests:

```
mvn test
```

- This command executes the tests in the project.
- Maven runs the unit tests and generates test reports, providing information on test coverage and results.

d. Package the project:

```
mvn package
```

- This command packages the project into a distributable format, such as a JAR or WAR file.
- The packaging configuration specified in the project's POM file determines the type of artifact created.

e. Install the project:

```
mvn install
```

- This command installs the project's artifact into the local repository.
- The packaged artifact is copied to the local repository, making it available for other projects on the same machine to use as a dependency.

f. Deploy the project:

```
mvn deploy
```

- This command deploys the project's artifact to a remote repository.
- It uploads the artifact to a remote repository, such as a Maven repository manager, making it accessible for other developers or deployment in production environments.

g. Generate project site documentation:

```
mvn site
```

- This command generates project documentation and reports, such as project information, project dependencies, test reports, and code coverage reports.
- The generated documentation is published in the ‘target/site’ directory.

- These are some of the commonly used Maven commands to build, test, and package projects.
- You can run these commands from the command-line interface or terminal in the project's root directory, where the project's POM file resides.
- Maven will execute the necessary tasks based on the command you run and the project's configuration in the POM file.