1.) What are the differences between ANT and Maven.

- ANT is low level build tool

- MAVEN is high level build tool

- Need more time build and deployment

- Very less time required

2.) How do you create a jar/war file in Maven?

With the help of pom.xml

3.)What is the difference between mvn deploy and mvn install?

mvn install: This command builds the project and deploys it to the local Maven repository. The local repository is usually located in the user's home directory under the ".m2" folder.

mvn deploy: This command deploys the project artifacts to a remote repository.such as Nexus or Jfrog

4.)Can you explain Maven's lifecycle? [Give one line explanation about each phase during your interview]

validate - validate the project is correct and all necessary information is available
initialize - initialize build state, such as setting properties

compile - compile the source code of the project

Surefire report - generates a report of the results of running unit tests during the build process.

package - take the compiled code and package it in its distributable format, such as a JAR or WAR

install - install the package into the local repository, for use as a dependency in other projects locally

5.)What is Maven? Why we use Maven?

Maven is a popular open-source build automation tool used primarily for Java projects. It is high level compile and build tool

Maven uses a declarative approach to project configuration, with a central pom.xml file that defines the project structure, dependencies, and build process.

6.)While building the project, you get an error saying some jar file is missing. how do you add that?

If a required jar file is missing during the build process, you can add it to your Maven project by specifying it as a dependency in the pom.xml file.

In pom.xml Navigate to the `<dependencies>` section and add a new `<dependency>` tag for the missing jar file.

7.)What is groupId, artifactId, and Version in Maven?

- groupId: It identifies the project's group or organization, usually in reverse domain name notation. For example, com.facebook
- artifactId: It identifies the project or module name. For example, facebook.
- version: It identifies the version of the project or module. For example, 1.0-SNAPSHOT.

8.)What are the Maven co-ordinates?

GAV is co-ordinates of maven

- groupId: It identifies the group or organization that owns the artifact.
- artifactId: It identifies the artifact name or module.
- version: It identifies the specific version of the artifact.

9.)What are the mandatory attributes in pom.xml?

The minimum requirement for a POM are the following:

- project root
- modelVersion - should be set to 4.0.0
- groupId - the id of the project's group.
- artifactId - the id of the artifact (project)
- version - the version of the artifact under the specified group

10.)What is the difference between 1.0-SNAPSHOT(SNAPSHOT) version and 1.0-RELEASE(RELEASE) version.

SNAPSHOT version: This is a development version that is subject to frequent changes and updates.

RELEASE version: This is a stable version of the project or module that is considered ready for release.

11.)What is the default naming convention of an artifacts(jar/war) in Maven?

In Maven, the default naming convention for an artifact (JAR/WAR) is as follows:

<artifactId>-<version>.<packaging>

12.)How do you generate a site in Maven?

In Maven, you can generate a project site using the mvn site command. This command generates a project site in the target/site directory by default, which contains various reports and documents related to the project, including:

- Project information (description, licenses, contributors, etc.)
- Dependency report
- Test coverage report
- Code analysis report
- JavaDoc report
- Change log
- Source code

13.)How do you run a clean build in Maven?

By running **mvn clean install**

clean: Deletes the target directory, which contains the compiled classes, generated resources, and other build artifacts from the previous build.

install: Compiles the source code, generates the project artifacts (JAR, WAR, etc.), and installs them in the local repository (~/.m2/repository by default) for later use in other projects or modules.

14.)how do you add a dependency in Maven pom.xml?

you can add a dependency to a project by adding a <dependency> element to the dependencies section of the pom.xml file. The <dependency> element contains the following information about the dependency:

groupId: The group or organization that created the dependency.

artifactId: The name of the dependency artifact.

version: The version of the dependency artifact.

scope: The scope of the dependency (e.g., compile, test, provided, etc.).

type: The type of the dependency (e.g., JAR, WAR, etc.).

optional: Whether the dependency is optional (true/false).

exclusions: A list of exclusions for transitive dependencies.

15.)what is a plugin?

Plugins are defined in the pom.xml file using the <plugin> element, which specifies the plugin's group, artifact ID, version, and configuration parameters. For example, to use the Maven Compiler Plugin to compile Java source code, you would add the following configuration to your pom.xml file

16.) What is the default path of artifacts in local repository?

the default path of artifacts in the local repository depends on the operating system:

- On Windows, the default path is C:\Users\{user}\.m2\repository.
- On Unix-based systems (Linux, macOS), the default path is ~/.m2/repository.
-

17.)Where maven stores the built artifacts?

the built artifacts (such as JAR files, WAR files, etc.) are stored in the target directory of the project's root directory by default.

18.) How do you create a project in the Maven?

To create a new project in Maven, you can use the archetype:generate goal, which creates a new Maven project from an archetype. An archetype is a template project that defines the project structure, build configuration, and initial source code for a specific type of application.

$mvn archetype:generate -DgroupId=com.flipkart -DartifactId=flipkart -Dversion=1.0-SNAPSHOT -DinteractiveMode=false

19.) What are the different binary repositoris we have? Which one you are using for your project?

There are several binary repositories that are commonly used for Java projects, including:

1. Apache Maven Central Repository: This is the default repository for Apache Maven and contains a large number of open-source Java libraries and frameworks.
2. JCenter: This is a popular binary repository maintained by JFrog and hosts a large number of open-source Java libraries and frameworks.
3. Google Maven Repository: This repository is hosted by Google and contains various libraries and artifacts for Android development.
4. Sonatype Nexus Repository: This is a powerful binary repository manager that can be used to host both open-source and private artifacts.

5. Artifactory: This is another popular binary repository manager that can be used to host both open-source and private artifacts.

20.)How do you customize the name of your artifact(jar/war) in Maven?

By default, Maven uses the artifactId, version, and packaging elements in the pom.xml file to generate the name of the artifact. The name of the artifact is typically in the format of <artifactId>-<version>.<packaging> (e.g., my-app-1.0-SNAPSHOT.jar).

To customize the name of the artifact, you can use the finalName element in the build section of the pom.xml file.

21.) What do you mean by transitive dependency in Maven and can you explain how maven resolves it?

a transitive dependency is a dependency that is required by another dependency of your project, rather than being directly declared in your project's pom.xml file.

22.) What is the significance/importance of scope parameter in dependency section?

The scope parameter in the dependency section of a Maven project's pom.xml file specifies the scope of the dependency, which determines when and where the dependency is available during the build process.

23.) What are the different scope's we have in Maven?

1. compile: This is the default scope. Dependencies declared with compile scope are available at compile time and runtime. They are included in the final artifact (JAR or WAR file).
2. provided: Dependencies declared with provided scope are provided by the environment (such as a web container or application server) and are not included in the final artifact. They are available at compile time and runtime.
3. runtime: Dependencies declared with runtime scope are not available at compile time but are available at runtime. They are not included in the final artifact.
4. test: Dependencies declared with test scope are used only for testing purposes and are not included in the final artifact.
5. system: Dependencies declared with system scope are similar to provided scope, but the difference is that you have to provide the dependency JAR file explicitly with a system path. They are not included in the final artifact.

24.)What inside the target folder?

The target folder is created by Maven during the build process, and it is where Maven stores the compiled class files, test classes, and other build artifacts.

Inside the target folder, you may find the following files and folders:

1. classes: This folder contains the compiled class files (i.e., .class files) of your project's main Java source code.
2. test-classes: This folder contains the compiled class files of your project's test Java source code.
3. generated-sources: This folder contains any generated source code that was created during the build process.
4. generated-test-sources: This folder contains any generated test source code that was created during the build process.
5. maven-archiver: This folder contains the archive files (e.g., JAR, WAR) that were created by Maven during the build process.
6. maven-status: This folder contains status files that are used by Maven to keep track of which files have been built and which files need to be rebuilt.
7. surefire-reports: This folder contains the test reports that were generated by the Surefire plugin.
8. failsafe-reports: This folder contains the test reports that were generated by the Failsafe plugin.
9. site: This folder contains the generated site documentation that was created by Maven during the site phase.