MODULE 2

Dependency Management

You will be discussing about 'Dependency Management', how to use build repositories, managing transitive dependencies, dependency scope and discuss about build tools like Maven, Ant and Gradle.

Module Objectives

At the end of this module, you will be able to:

- Define build tools (Maven, Ant and Gradle)
- Understand different types of repositories
- Explain dependency management
- Learn how to identify the dependencies
- Discuss dependency scope
- Describe transitive dependencies

Facilitator Notes:

Explain the module objectives to the participants.

Module Topics

Let us take a quick look at the topics that we will cover in this module:

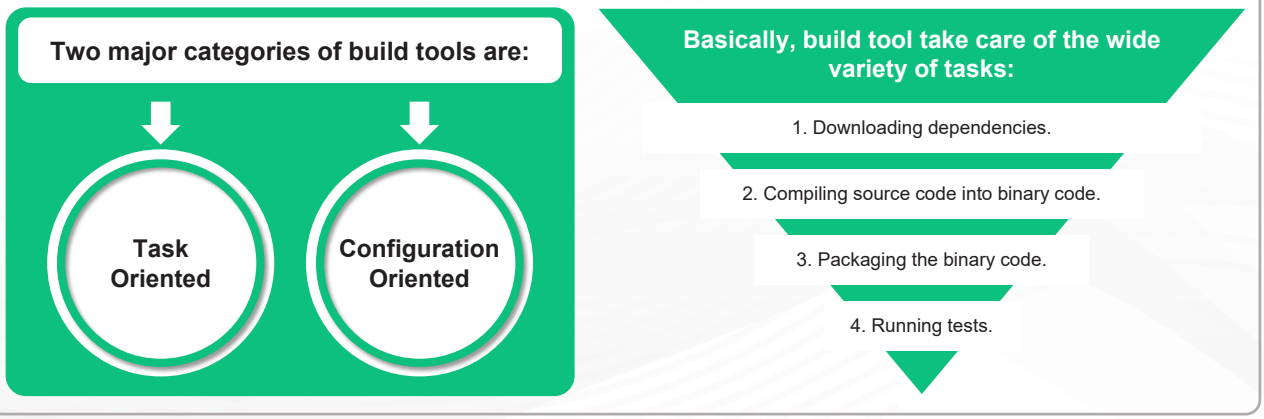
1. Build tools – Ant, Maven and Gradle
2. Using repositories
3. Dependency management
4. Dependency identification
5. Dependency scope
6. Transitive dependencies

Facilitator Notes:

Inform the participants about the topics that they will be learning in this module.

1. Build Tools

- Build tools are applications that convert source code to executable applications.
- Build tools take care of the compilation sequence of the code.



Facilitator Notes:

Explain details about build tools.

The process of code compilation is very important for high-level programming to create a software application. Modern build tools enable workflow processing by downloading the source code, compiling the code, deploying executables using distributed build technologies.

The type of build tools is necessary to decide which one is required for which type. Build tools can be categorized as:

- **Task-oriented** – Rake/Gulp/Make/Ant

  Tasks are the units of your build script that actually execute the build operations for your project. Some of the tasks are mkdir, copy, delete, javac, Javadoc and jar

- **Configuration Oriented** – Gradle/Maven

  Configuration such as compiling the source code, generating generate-sources etc.

The most popular build tools and maven and ant. Both the build tools can be executed in build tools like Eclipse or Netbeans or IntelliJ. In the next section, we will discuss Maven, Ant and Gradle.

1.1 Build Tools - Ant

Following are the key details of build tool Ant:

- Ant is an open source build tool from Apache tomcat project from early 2000.
- Ant (Another Neat Tool) was created by James Duncan Davidson.
- Ant is primarily used to build Java applications.
- Ant can also be used to build non Java applications like C, C++.
- Ant users can develop their own Antlibs which contains Ant tasks and types.
- Ant tools are a replacement of Unix Make build tool, which was created due to a number of issues with Make tool.
- Ant uses XML file format to describe the build process and its dependencies.

Facilitator Notes:

Give an overview of the build tool Ant.

Features of Apache Ant:

- Ant is the most complete Java build and deployment tool available.
- Ant can handle platform specific properties and neutral to the platform.
- Ant can be used to perform platform specific tasks like modifying the time of a file using ‘touch’ command.

- Ant scripts use plain XML language.
- Ant is used to automate repetitive tasks.
- Ant has many predefined tasks.
- Ant do not have a life cycle.
- The ant scripts are not reusable.
- Ant is less preferred than Maven.
- Ant provides an interface to develop custom tasks.

A sample build file in ANt is given below:

```
<?xml version="1.0"?>
<project name="Hello" default="compile">
    <target name="clean" description="remove intermediate files">
        <delete dir="classes"/>
    </target>
    <target name="clobber" depends="clean" description="remove all artifact files">
        <delete file="hello.jar"/>
    </target>
    <target name="compile" description="compile the Java source code to class files">
        <mkdir dir="classes"/>
        <javac srcdir="." destdir="classes"/>
    </target>
    <target name="jar" depends="compile" description="create a Jar file for the application">
        <jar destfile="hello.jar">
            <fileset dir="classes" includes="**/*.class"/>
            <manifest>
                <attribute name="Main-Class" value="HelloProgram"/>
            </manifest>
        </jar>
    </target>
</project>
```

Facilitator Notes:

Show the sample Ant build file.

You can refer the above picture to see sample build file of Ant.

In the above example first task clean removes intermediate files, second task will delete the helloo. jar, third class compiles the java source code from source dir and fourth class will create hello.jar file by compiling the source code.

1.2 Build Tools - Maven

Following are the key details of maven build tool:

1. Maven is an open source software build tool primarily used for Java-based applications.
2. It became part of the Apache software foundation project in 2003.
3. Maven was released in several versions – 1.0, 2.0 and 3.0 .
4. Maven was created in 2002 by Jason Van Zyl.
5. Maven can be used with other programming languages like Ruby and C#.
6. Maven depends on POM (Project Object Model) written in xml that has details about the project.

Facilitator Notes:

Give an introduction about maven build tool.

Key features of Maven include:

- An easy, and standard way to build projects where unnecessary details are hidden
- A standard strategy is followed while building any project
- Automatic dependency management
- Multiple projects can be simultaneously managed
- Necessary plugins and libraries are automatically downloaded from Maven repositories

Most commonly used build commands are as follows:

Facilitator Notes:

Explain details about commonly used build commands in maven.

Maven build commands that are widely used are discussed below:

- **Validate:** It validates the entire project and downloads required dependencies.
- **Compile:** Compiles the source code of the project.
- **Test:** It will just test the source code using test framework and to run this commands deploy or packaging is not required.
- **Package:** Packages the compiled source code into executable application like jar.
- **Install:** This command will install the package into the local repository.
- **Deploy:** It copies the final package into a remote repository.

1.2.1 Maven POM

A sample POM file in Maven is given below:

```
<project>
  <!-- model version is always 4.0.0 for Maven 2.x POMs -->
  <modelVersion>4.0.0</modelVersion>

  <!-- project coordinates, i.e. a group of values which
       uniquely identify this project -->

  <groupId>com.mycompany.app</groupId>
  <artifactId>my-app</artifactId>
  <version>1.0</version>

  <!-- library dependencies -->

  <dependencies>
    <dependency>

      <!-- coordinates of the required library -->

      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>

      <!-- this dependency is only used for running and compiling tests -->

      <scope>test</scope>

    </dependency>
  </dependencies>
</project>
```

Facilitator Notes:

Show the sample maven POM file.

Maven depends on POM file to build the project. You can refer the above picture to see sample POM file of maven.

In the above picture unit tests are executed for com.mycompany.app group ID.

Basic Elements of a POM file:

project: It is the root element of pom.xml file.

modelVersion: It is the sub element of project. It specifies the modelVersion. It should be set to 4.0.0.

groupId: It is the sub element of project. It specifies the id for the project group.

artifactId: It is the sub element of project. It specifies the id for the artifact (project). An artifact is something that is either produced or used by a project. Examples of artifacts produced by Maven for a project include: JARs, source and binary distributions, and WARs.

version: It is the sub element of project. It specifies the version of the artifact under given group.

1.3 Build Tools - Gradle

Following are the key details of build tool Gradle:

- Gradle is an open source tool release with the Apache license.
- Gradle scripts are written in Groovy or Kotlin DSL instead of xml.
- Gradle uses Directed Acyclic Graph(DAG) to decide the order of tasks to be executed.
- Gradle uses flexible conventions.
- Gradle uses robust and powerful dependency management.

Facilitator Notes:

Give an overview of the build tool Gradle.

Following are the features of Gradle:

- Highly customizable: Gradle is modeled in such a way that extensible and customizable in the most ways.
- Fast: Gradle reuses output from previous executions and completes tasks quickly, it processes only inputs that is changed, and executes tasks in parallel.
- Powerful: Gradle is the official build tool for Android.
- Gradle plugins can provide users with flexible conventions in a certain context. Refer the below examples:
    - It defines the directory src/main/java as the default source directory for compilation.
    - The output directory for compiled source code and other artifacts (like the JAR file) is build.

- Getting help on Gradle - https://docs.gradle.org/current/userguide/userguide.html#getting_help
    - Forum: On Gradle forum, we can get quick help from core contributors and community members.
    - Training: Gradle developers provide free, web-based training every month.
- Gradle has built-in support for dependency management. It provides multiple options like file based dependencies, custom depedency scopes, and 3rd party depedency scope etc.

A sample configuration file in Gradle is given below:

```
apply plugin: 'java'
apply plugin: 'checkstyle'
apply plugin: 'findbugs'
apply plugin: 'pmd'

version = '1.0'

repositories {
    mavenCentral()
}

dependencies {
    testCompile group: 'junit', name: 'junit', version: '4.11'
    testCompile group: 'org.hamcrest', name: 'hamcrest-all', version: '1.3
}
```

Facilitator Notes:

Show the sample configuration file of Gradle.

You can refer the above picture to see the sample configuration of Gradle.

In the above sample screenshot Gradle file will generate checkstyle, findbugs and pwd reports for the source code.

What did You Grasp?

3. Which build tool uses POM to build the project?
   - **A) Maven**
   - **B) Ant**
   - **C) Gradle**
   - **D) Make**

4. In which build phase, Unit test reports are recorded?
   - **A) Build Identification**
   - **B) Build Definition**
   - **C) Build Execution**
   - **D) Build Reporting**

Facilitator Notes:

Tell the participants that they will be going through a knowledge check question.

Answer:

1. b. Ant

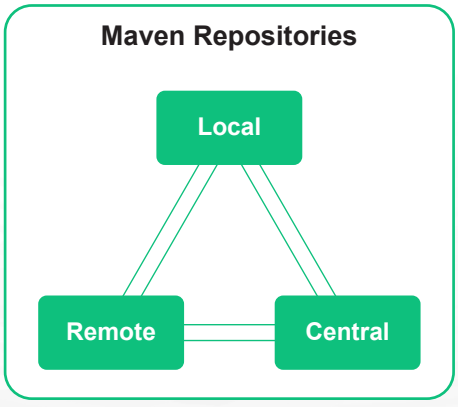
2. c. Gradle

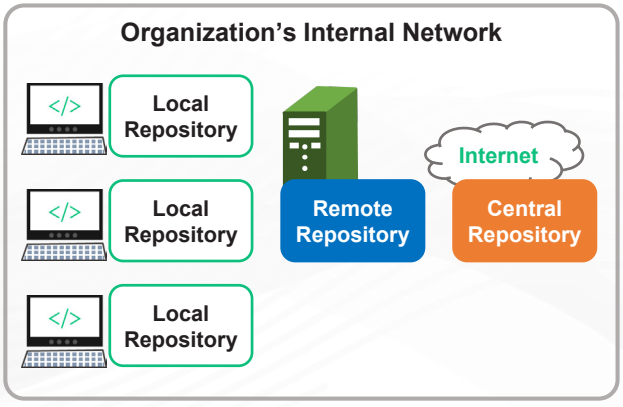
3. a. Maven

4. d. Build Reporting

2. Using Repositories

A repository is a location where all the libraries, project jars, plugins or project specific artifacts are stored. In this study guide we will be discussing maven repositories:





Facilitator Notes:

Give an introduction about different types of source code repositories in maven.

A Maven repository stores project artifacts, dependencies and retrieve artifacts from a repository. Maven Repository act as highly configurable proxies between public repositories and your organization and it also provides a destination to deploy your artifacts.

Maven is useful because it can automatically manage large dependency trees, especially transitive dependencies where Plugin A depends on Application B, which depends on Shared Library C. Maven will quickly and easily retrieve all the dependencies that are required for the project, and compile environment in a few steps.

2.1 Maven Repository - Local

Following are the key details of maven local repository:

- The local copy of dependencies.
- It is created after executing the first mvn command on the system.
- Caches remote downloads.
- Temporary build artifacts that are not released will be stored in the local repository.
- Location of local repository is user_home/.m2/repository

Facilitator Notes:

Discuss about maven local repository.

Maven local repository is created after executing the first command of maven. It downloads the remote artifacts and stores them in the repository. If a build is executed again artifacts from the local repository will be used instead of downloading them again.

The default location of the local repository is USER_HOME/.m2/repository. However, the location of the repository can be changed by editing the configuration file. You should mention absolute path for the local repository in your configuration file. Please refer below example :

${user_home}/.m2/settings.xml – Configuration file
<settings>
...
<localRepository>/path/to/repo/</localRepository>
...
</settings>

2.2 Maven Repository - Central

Following are the key details of maven central repository:

- It is provided by the Maven central community.
- Most commonly used libraries can be found here.
- If maven is unable to find dependency in the local repository it will search in the central repository.
- Configuration is not required to use the central repository.
- Internet connection is required to access central repository.

Facilitator Notes:

Discuss about maven central repository.

Maven central repository is maintained by maven community and it largely contains most commonly used dependencies.The Internet connection is required to download the dependencies.

The URL of the central repository is https://search.maven.org/#browse . Using this URL a developer can search for required dependencies on the central repository.

2.3 Maven Repository - Remote

Facilitator Notes:

Discuss about maven remote repository.

Sometimes for specific development, the company uses its own repository to store the libraries.

The following POM declaration is an example of specifying a remote repository location:

```
<project ...>
   <dependencies>
      <dependency>
         <groupId>com.company.common-lib</groupId>
         <artifactId>common-application-lib</artifactId>
         <version>1.0.0</version>
      </dependency>
   <dependencies>
   <repositories>
      <repository>
         <id>companyname.library1</id>
         <url>http://download.companyname.org/maven2/library</url>
      </repository>
      <repository>
         <id>companyname.library2</id>
         <url>http://download.companyname.org/maven2/library2</url>
      </repository>
   </repositories>
</project>
```

2.4 Maven Dependency Search Sequence

When a maven command is executed, it follows a particular sequence to search for the dependency:

Build fails if it is unable to find the dependency in the above repositories

Facilitator Notes:

Discuss about maven dependency search sequence.

Maven follows a particular sequence to search the libraries. It involves:

- **Step 1** − It initially searches the local repository for the dependency. It the dependency is found, it continues with the build execution. If the dependency is not found, then it proceeds to step 2 for further execution.
- **Step 2** − If the dependency is not found in the local repository maven will search central repository. If the required dependency is found, it will be copied to the local repository for build execution. If the dependency is not found, then it will search in the remote repository if configured or the build fails.
- **Step 3** − If the remote repository is mentioned in pom.xml, then maven starts searching the remote repository for the dependency that is not found in the central repository. If the dependency is found, it is copied to the local repository for further processing or the build fails.
- **Step 4** − If the dependency is not found in any of the repository maven will stop processing the build and throws an error – "Required dependency not found. Build Failed".

What did You Grasp?

1. What is the default repository for maven?
   - **A) Remote repository**
   - **B) Local Repository**
   - **C) Central Repository**
   - **D) None of the above**

2. State True or False
   It is not possible to change the default repository of maven.
   - **A) True**
   - **B) False**

3. If maven is unable to find the dependency in the local repository, it will search in which repository?
   - **A) Remote repository**
   - **B) Local Repository**
   - **C) Central Repository**
   - **D) None of the above**

4. Remote repository is maintained by :
   - **A) Maven**
   - **B) Developer**
   - **C) Organization**
   - **D) Remote users**

Facilitator Notes:

Tell the participants that they will be going through a knowledge check question.

Answer:

1. b. Local Repository
2. b. False
3. c. Central Repository
4. c. Organization

Group Discussion

Facilitator Notes:

Divide the students into groups and discuss about Maven repository types.

We've so far seen about the different types of repository used by maven. Form different groups and each group should talk about one concept in detail along with analogies or examples to show your understanding.

3. Dependency Management

Following are the key details of dependency management:

- It's a mechanism to centralize dependency information.
- Each component publishes the output to a common repository for usage of other dependency modules.
- Each child module is added as an external definition to master project. This enabled automatic checkout of child modules automatically while building the master.
- This is useful when we have the multiple set of projects.

Facilitator Notes:

Discuss about dependency management and its advantages.

Most of the Java programs depend on other projects or open source frameworks to function properly. Is it difficult to download all the dependencies manually and keep a track of the dependency version as we use them in the project. In this module, we will discuss about maven as it is widely used build tool.

Maven provides an easy way to declare the project dependencies in external pom file and it automatically downloads the dependencies and allows the user to use those dependencies in the project. This feature solves the project dependency issue to a great extent.

By using this command: **mvn dependency:tree,** you can see all the dependencies in your project.

3.1 Example of Dependency Management: Parent POM

The below screenshot of a parent POM file shows dependency management.

```
<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>
    <groupId>com.java.code.skaas</groupId>
    <artifactId>java-code-skaas-parent</artifactId>
    <version>1.0.1</version>
    <packaging>pom</packaging>

    <dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>log4j</groupId>
                <artifactId>log4j</artifactId>
                <version>1.2.17</version>
            </dependency>
        </dependencies>
    </dependencyManagement>
</project>
```

Facilitator Notes:

Discuss about pom.xml of parent module.

If you are working on multi-module project, you can specify the artifact version in a parent project and it will be automatically inherited by the child projects. This makes it easier to maintain the dependencies in the entire project.

If two projects are using the same dependency, you can add it in <dependencyManagement> tag inside the parent pom, extend two poms with the parent pom and this makes child poms simpler.

In the above diagram, log4j dependency is added in the parent pom. Child modules can simply inherit the same version as described in the parent pom.

3.2 Example of Dependency Management: Child POM-a

The below screenshot of a child POM file shows dependency management.

```
<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>
    <artifactId>java-code-skaas-a</artifactId>
    <version>1.0.1</version>
    <packaging>pom</packaging>
    <parent>
        <groupId>com.java.code.skaas</groupId>
        <version>1.0.1</version>
        <artifactId>aca-parent</artifactId>
    </parent>

    <dependencies>
        <dependency>
            <groupId>log4j</groupId>
            <artifactId>log4j</artifactId>
        </dependency>
    </dependencies>

</project>
```

Facilitator Notes:

Discuss about the reference of log4j dependency in of child pom-a.

In the above diagram child pom-a did not specifically define the version of log4j. It is inherited from the parent module.

Notice that we now have added the parent section. This section allows us to specify which artifact is the parent of our POM. And we do so by specifying the fully qualified artifact name of the parent POM. With this setup, our module can now inherit some of the properties of our parent POM.

3.3 Example of Dependency Management: Child POM-b

The below screenshot of a child POM file shows dependency management.

```
<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>
    <artifactId>java-code-skaas-b</artifactId>
    <version>1.0.1</version>
    <packaging>pom</packaging>
    <parent>
        <groupId>com.java.code.skaas</groupId>
        <version>1.0.1</version>
        <artifactId>aca-parent</artifactId>
    </parent>

    <dependencies>
        <dependency>
            <groupId>log4j</groupId>
            <artifactId>log4j</artifactId>
        </dependency>
    </dependencies>

</project>
```


Facilitator Notes:

Discuss about the reference of log4j dependency in of child pom-b.

In the above diagram child pom-a did not specifically define the version of log4j. It is inherited from the parent module.

4. Dependency Identification

Dependency Identification

- Dependencies can be classified based on a number of criteria like whether the dependency exists between within the project or outside of the project.
- Analyzes the dependencies of this project and determines which are: used and undeclared, unused and declared and used and declared.
- Based on dependency type, dependencies can fall into one of the following three groups:
  - Internal dependencies
  - External dependencies

Facilitator Notes:

Explain the participants about dependency identification.

The relationship between processes or tasks are called dependencies. Keeping track of all the dependencies and managing them efficiently is important for project planning, tracking, schedule and execution.

Based on the type, dependencies can be classified into one of the following:

- Internal dependencies are required between two tasks or within the same project. The project team has complete control over internal dependency management.
- External dependencies are required by other projects. Project teams do not have much control over these dependencies.

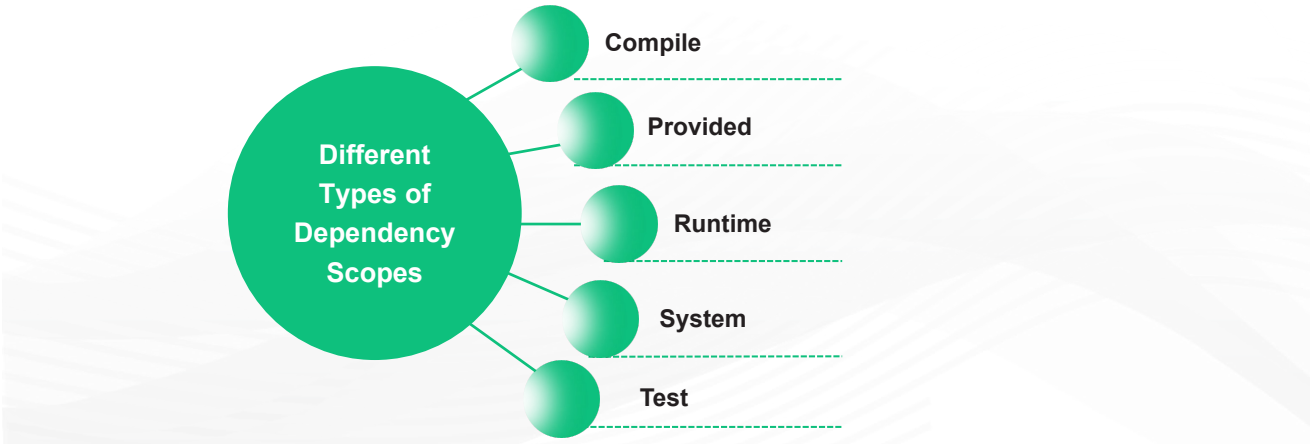
Each project manager needs to identify the dependencies for their project. This can be achieved by the project manager sitting down, reviewing the plan and documenting potential dependencies.

A better approach is to run a dependency identification session. This typically will involve getting a group of people into a workshop who understand the project and / or subject matter.

All dependencies should be captured into the defined dependency tool and MUST include who is responsible for delivering the dependency to the project.

5. Dependency Scope

The dependency scope defines in build phase or which environment you want to use an artifact and it affects the classpath used for an artifact.



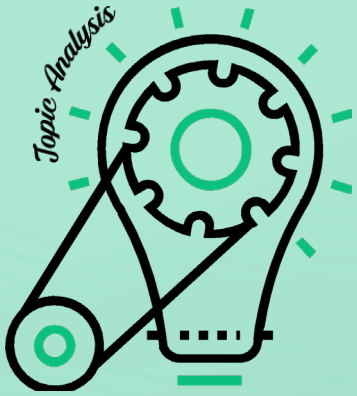
Facilitator Notes:

Discuss the project dependency scope.

Take note of the key features of the module.

- **Compile:** In the compile phase, dependencies are available in the classpath of the project. All the dependencies are propagated to dependent projects.
- **Provided:** If the JDK or the application carries the dependency this phase is used. This phase is available on test compile classpath.
- **Runtime:** This scope is used to indicate that the dependency is required only for execution and not at compile time. This dependency is used in the test and runtime, but not in compile time.
- **Test:** The dependency declared in this phase will be available only on test and execution classpaths only.
- **System:** This phase is used to use a local jar on the filesystem. The repository will not be searched.

What did You Grasp?



1. In which phase dependency is not required for compilation, but required for execution?
   - **A) Compile**
   - **B) Provided**
   - **C) Runtime**
   - **D) Test**
   - **E) System**

2. In which phase the artifact is not searched in the repository?
   - **A) Compile**
   - **B) Provided**
   - **C) Runtime**
   - **D) Test**
   - **E) System**

3. In which phase the dependency is needed at run time?
   - **A) Compile**
   - **B) Provided**
   - **C) Runtime**
   - **D) Test**
   - **E) System**

4. In which phase repository is not searched for a jar file?
   - **A) Compile**
   - **B) Provided**
   - **C) Runtime**
   - **D) Test**
   - **E) System**

Facilitator Notes:

Tell the participants that they will be going through a knowledge check question.

Answer:

1. c. Runtime
2. e. System
3. c. Runtime
4. e. System

6. Transitive Dependencies

Following are the key details of transitive dependencies:

- The dependencies of the direct dependencies are called transitive dependencies.
- There is no limit on the number of levels that dependencies can be downloaded from.
- Maven is extremely powerful because of the transitive dependency feature.
- With this feature, the dependency graph can grow quietly large.
- In a real-world program, the dependency tree might contain hundreds of JAR files with multiple levels of transitive dependencies.

Facilitator Notes:

Discuss about transitive dependencies in detail

- Transitive dependency automatically downloads the specific libraries that own dependencies require. If the project is large the dependency graph can grow large.
- Whenever the direct dependencies are changed, run **mvn dependency:tree** to find out what has changed with transitive dependencies.
- Transitive dependency avoids the need to discover and mention libraries that your own dependencies require, that means if the project is dependent on few libraries and those libraries are dependent on different libraries, you only specify the project's dependencies and this feature takes care of all other dependencies.
- There will be no limit of the number of level dependencies can be gathered from, and it will only create a problem if a cyclic dependency is discovered.

6.1 Features of Transitive Dependencies

Following features of transitive dependency helps us to handle the large graph of included libraries:

Facilitator Notes:

Discuss about the features of transitive dependencies.

Let's discuss features of transitive dependency that help in managing the dependencies efficiently in a large infrastructure:

- **Dependency mediation:** This feature decides which version of a dependency will be used when multiple versions of an artifact are encountered. If the dependency version of the two modules is declared in the same depth, in that case first declaration wins.
- **Dependency Management:** With this feature, the version of artifacts can be specified to be used when they are required in transitive dependencies.

- **Dependency Score:** Only the appropriate dependencies required for the current build is included.
- **Excluded Dependencies:** If project A depends on project B, and project B depend on project C, the project A can explicitly exclude project C as a dependency using the exclusion element.
- **Optional Dependencies:** If project B depends on project C, and the owner of Project B can mark project C as optional dependency, using the optional element. When project A depends on project B, A will depend only on B and not on B's optional dependency C.

What did You Grasp?

1. In maven transitive dependency there is a limit in number of levels that the dependency can be downloaded?
   A) True
   B) False

Facilitator Notes:

Tell the participants that they will be going through a knowledge check question.

Answer:

1. b. False

Group Discussion

Facilitator Notes:

Divide the students into different groups and discuss about dependency management concepts in detail.

Form different groups and discuss about dependency management and the different concepts associated with it. Each group can summarize the key points of one of the concepts.

In a nutshell, we learnt:

1. Build tools are applications that convert source code to executable applications.
2. Maven is the widely used build tool for build java based applications.
3. A repository is a location where all the libraries, project jars, plugins or project specific artifacts are stored.
4. Maven has three types of repositories – Local, Remote and Central.
5. Dependency management is a mechanism to centralize dependency information.
6. The dependencies of the direct dependencies are called transitive dependencies.

Facilitator Notes:

Share the module summary with the audience.

Ask the participants if they have any questions. They can ask their queries by raising their hands.

