Bachelor of Technology

Computer Engineering Sem : 7

01CE0717 – DevOps Essentials

Unit : 2 - Compile and Build Using Maven and Gradle

Content

- Introduction
- Installation of Maven, POM files Maven Build lifecycle, Build phases(compile build, test, package) Maven Profiles,
- Maven repositories(local, central, global), Maven plugins,
- Maven create and build Artifacts,
- Dependency management,
- Installation of Gradle, Understand build using Gradle

Introduction

- Maven is a project management tool.
- It is based on **P**roject **O**bject **M**odel (POM).
- This tool is used for build, dependency and documentation.
- Maven simplifies and standardizes the project build process.
- Apache Maven is a popular build automation and project management tool used primarily for Java Projects.
- It can be used in building and managing the projects written in C#, ruby and other programming languages.

Introduction

Problems without Maven (Building Java Project without Maven)

- **Dependency management:**
  - To manage dependency manually (such as downloading library files, ensuring compatibility, manage version control)
  - It is time consuming and error prone.

- **Build automation:**
  - Manually create Jar and War files, and its dependencies set up manually

- **Standardized project structure:**
  - Maven creates a standardized project structure which can be easily grasped by the developers.
  - Without Maven, It is difficult to set up and maintain the standard project structure that could be followed by all the team members.

Introduction

Problems without Maven (Building Java Project without Maven)

- **IDE integration:**
  - Many popular IDE provide integration with Maven which allows easy to import, build and manage the whole project.
  - Without Maven, we need to configure the IDE manually.

- **Lack of Plugins and community support:**
  - Maven has support for plugins that can simplify various tasks such as deployment and documentation generation.

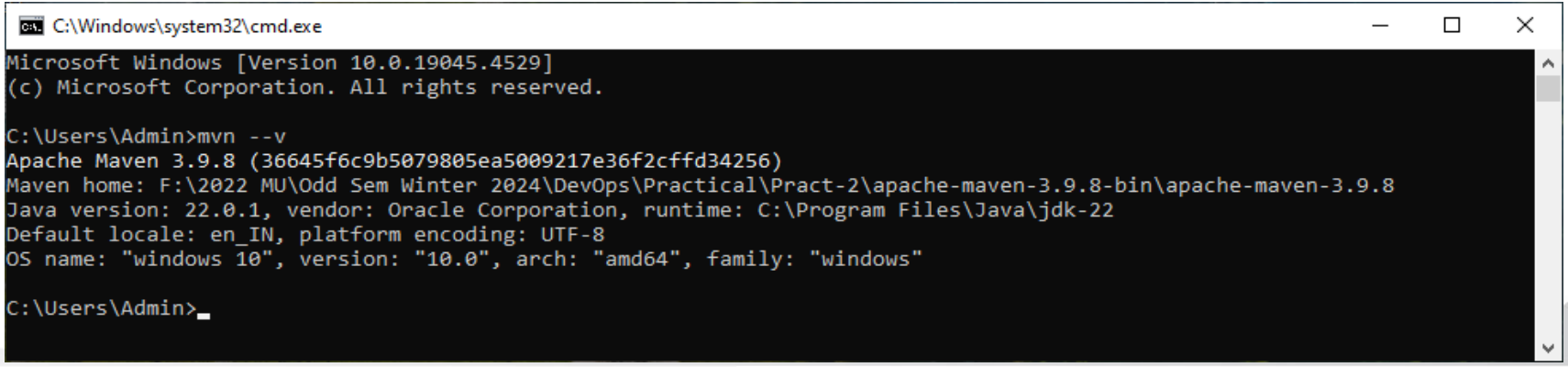
Introduction

Features of Maven

- Simple project setup
- Superior dependency management (Automatic updating)
- Ability to easily write plugins
- To build any number of projects
- Work with multiple projects at the same time
- Maven encourages the use of a central repository of JARs and other dependencies. Maven comes with a mechanism that can be used to download any JARs required for building your project from a central JAR repository.

Installation of Maven

- Prerequisite
  - JDK 8 or above
- Download from
  - https://maven.apache.org/download.cgi
- Set Environment variable
  - …\apache-maven-3.9.8-bin\apache-maven-3.9.8\bin
- Check version
  - $mvn --v



POM files Maven Build lifecycle

https://maven.apache.org/guides/getting-started/maven-in-five-minutes.html

- **Creating a Project**
  - mvn archetype:generate -DgroupId=com.mycompany.app -DartifactId=my-app -DarchetypeArtifactId=maven-archetype-quickstart -DarchetypeVersion=1.4 -DinteractiveMode=false

- you will notice the following standard project structure.

```
my-app
|-- pom.xml
`-- src
    |-- main
    |   `-- java
    |       `-- com
    |           `-- mycompany
    |               `-- app
    |                   `-- App.java
    `-- test
        `-- java
            `-- com
                `-- mycompany
                    `-- app
                        `-- AppTest.java
```

POM files Maven Build lifecycle

- **Creating a Project**
  - mvn archetype:generate -DgroupId=com.mycompany.app -DartifactId=my-app -DarchetypeArtifactId=maven-archetype-quickstart -DarchetypeVersion=1.4 -DinteractiveMode=false

archetype - refers to a template for generating a new project
groupId – is a unique identifier for your project's group, typically following the reverse domain name convention (e.g., com.example.myapp)
D – is a flag used to define system properties or configuration parameters
artifactId - is a unique identifier for a project within a group. It is used to define the name of the artifact (e.g., a JAR, WAR, or other packaged output) that will be produced by the project.
archetypeArtifactId - artifact ID of the archetype
archetypeVersion – version of the archetype
interactiveMode – false means no interact with the user //(Batch Mode)

POM files Maven Build lifecycle

- **Creating a Project**
  - mvn archetype:generate -DgroupId=com.mycompany.app -DartifactId=my-app -DarchetypeArtifactId=maven-archetype-quickstart -DarchetypeVersion=1.4 -DinteractiveMode=false //without interactive
  - **mvn archetype:generate** //with Interactive mode

POM files Maven Build lifecycle

https://maven.apache.org/guides/getting-started/maven-in-five-minutes.html

- **POM Files**
  - Stands for Project Object Model
  - It is XML file which is named as pom.xml
  - It is the core of a project's configuration in Maven.
  - It is a single configuration file that contains the majority of information required to build a project in just the way you want.
  - It contains information related to the project and configuration information such as
    - dependencies,
    - plugins
    - source directory,
    - goals and so on.
  - Maven reads pom.xml file to accomplish its configuration and operations.

POM files Maven Build lifecycle

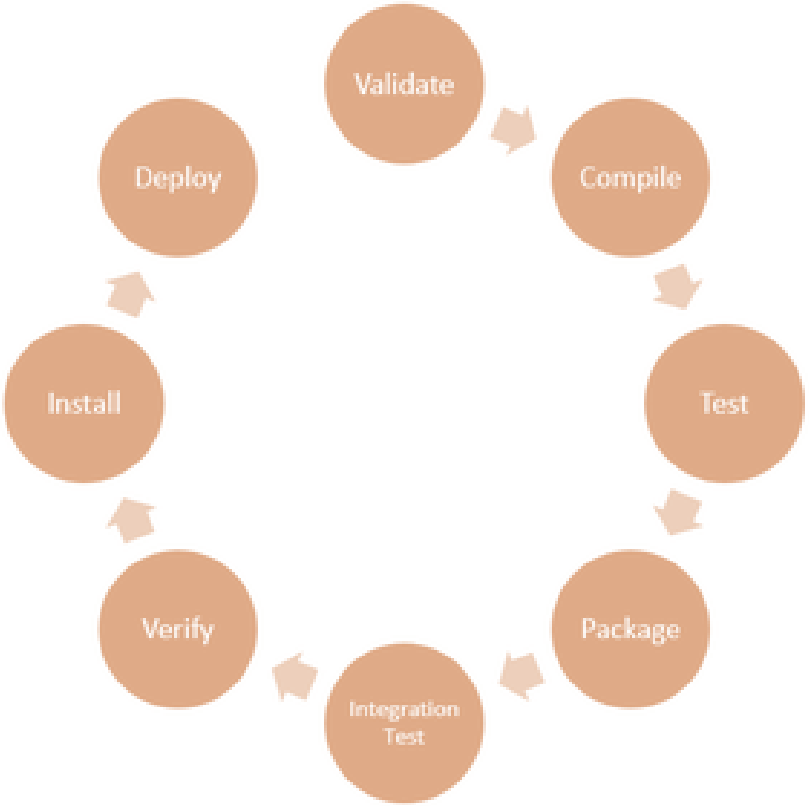
- **POM Files**
  - **Elements of pom.xml file**
    - **project** : root element; top level element
    - **modelVersion** : It indicates the version model in which the current pom.xml is using.
    - **groupId**: It indicates the unique identifier of the organization or group that created project.
    - **arifactId**: It indicates the unique base name of the primary artifact being generated by this project. The primary artifact for a project is typically a JAR file. Secondary artifacts like source bundles also use the artifactId as part of their final name.
    - **version** : It indicates the version of the artifact generated by the project.

POM files Maven Build lifecycle

- **POM Files**
  - **Additional Elements**
    - **name** : It indicates the display name used for the Maven project.
    - **url** : It indicates where the project's site can be found.
    - **dependencies:** The POM lists all the external libraries and dependencies that the project relies on during the build process and runtime.
    - **dependency:** It define a dependency. It is used inside dependencies.
    - **plugins**: Maven allows developers to use various plugins to extend its functionality during the build process. The POM defines which plugins are applied to the project and the respective configurations.
    - **scope** : It is used to define the scope for this maven project.

POM files Maven Build lifecycle

- **Maven Build Lifecycle**



POM files Maven Build lifecycle

- **Maven Build Lifecycle**
  - **validate:** It confirms that all the data necessary for the build is available.
  - **Compile:** compile the source code of the project.
  - **test:** test the compiled source code using a suitable unit testing framework.
  - **package:** It take the compiled code and package it in its distributable format, such as a JAR.
  - **verify:** run any checks on results of integration tests to ensure quality criteria are met.
  - **install:** install the package into the local repository, for use as a dependency in other projects locally.
  - **deploy:** Final package is deployed into a remote repository.

POM files Maven Build lifecycle

- **Maven Goals**
  - In Maven, each phase is a sequence of goals. The purpose of each goal in Maven is to perform in specific task.
  - When we build our Maven project, we need to specify the Goal.
  - Some of the popular goals are –
    - validate
    - compile
    - test
    - package
    - install
    - deploy
  - The build phases are executed sequentially. Any maven build phases that come before the specified phase is also executed. For example, if we run "***mvn package***" then it will execute compile, test and package phases of the project.

POM files Maven Build lifecycle

- **Built-in Build Life Cycle Phase**

  Maven comes with 3 built-in build life cycles as shown below:

  1) **Clean:** cleans up artifacts created by prior builds;

     It performs the cleaning operation in which it deletes the build directory name target and its contents for a fresh build and deployment. To perform this operation we use command ***mvn clean***

  2) **Default:** complete deployment of the project

  3) **Site:** This phase handles the generating java documentation of the project.

Maven Profiles

- A Maven profile is a way to customize the build process in Maven, the popular build automation tool for Java projects. Profiles allow you to define a set of configurations and settings that can be activated under specific conditions, such as for different environments (development, testing, production) or different platforms.
- Different Types of Build Profiles:
  1) **Per Project :** Defined in the POM itself (pom.xml)
  2) **Per User/Developer :** Defined in the Maven-settings (%USER_HOME%/.m2/setting.xml)
  3) **Global :** Defined in the global Maven-settings (%M2_HOME%/conf/settings.xml)

Maven Repositories

- Maven repository is a directory where all the project jars, plugins, library jars or any other project related artifacts are stored and these can be accessed by Maven easily.

- There are three Types of repositories in Maven:
  1) **Local repository**
  2) **Central repository**
  1) **Remote repository**

Maven Repositories

- There are three Types of repositories in Maven:
  1) **Local repository**
     - Stored on the developer's machine
     - Acts as a cache of downloaded dependencies
       - When we execute any Maven project that requires dependencies, Maven downloads these dependencies from central or remote repository and store them on developer's machine
     - Typically located in the ***".m2"*** directory in the user's home directory
     - Example location: "C:\Users\Admin\.m2\repository"

Maven Repositories

- There are three Types of repositories in Maven:
  - **2) Central repository**
    - Maven central can be accessed from https://repo.maven.apache.org/maven2.
    - Maintained by the Maven community.
    - Whenever we execute our Maven project then, required dependencies are first searched in the local repository, if those dependencies are not present in the local repository then those are downloaded from the central repository and are stored in the local repository for the future use.

Maven Repositories

- There are three Types of repositories in Maven:
  3) **Remote repository**
     - Third-party repositories or internal repositories managed by organizations
     - Used for hosting custom, private or third-party dependencies not available in the central repository.
     - Defined in the project's "pom.xml" file or maven settings.

- Maven repositories are essential for managing project dependencies efficiently, ensuring consistent builds, and promoting the reuse of libraries and components in Java projects.

Maven Plugins

- Maven plugins are core components of the Apache Maven build automation tool.
- They provide additional tasks and goals that can be executed during the build process of a Maven project.
- Plugins are used to perform various build and project management operations, such as compiling code, packaging binaries, running tests, generating documentation, and deploying artifacts.
- Types of Maven Plugins
  1. Build Plugins
  2. Reporting Plugins

Maven Plugins

- Types of Maven Plugins
  1. **Build Plugins :**
     - These are used to execute tasks during the build process.
     - These plugins are declared inside <build> element.

     - Compiler Plugin: Compiles the project's source code.
     - Surefire Plugin: Runs the unit tests of a project.
     - Jar Plugin: Packages the compiled code into a JAR file.
     - War Plugin: Packages web applications into a WAR file.
     - Assembly Plugin: Creates distributions with the project binaries and dependencies.

Maven Plugins

- Types of Maven Plugins
  2. **Reporting Plugins :**
     - These are executed at the time of site generation.
     - These plugins are declared inside the element <reporting>.

     - Site Plugin: Generates a project website.
     - Javadoc Plugin: Generates Java API documentation.
     - PMD Plugin: Integrates PMD (a source code analyzer) to generate reports.

Maven Plugins

- Core Maven Plugins
  - clean : clean up after the build
  - compiler : Compiles Java sources
  - deploy : Deploy the built artifact to the remoter repository
  - failsafe : Run the JUnit integration tests in an isolated classloader
  - install : Install the built artifact into the local repository
  - resources : Copy the resources to the output directory for including in the JAR
  - site : Generate a site for the current project.
  - surefire : Run the JUnit unit tests in an isolated classloader
  - verifier : Useful for integration tests – it verifies the existence of certain conditions

Maven Plugins

- Benefits of Maven Plugins
  - **Modularity:** Adds modularity to the build process, allowing various tasks to be performed through plugins.
  - **Reusability:** Enables the reuse of common build logic across different projects.
  - **Customizability:** Plugins can be configured and customized as needed in the ‘pom.xml’.
  - **Extensibility:** New plugins can be created and added to Maven repositories, extending Maven's capabilities.

- Maven plugins are essential tools that extend Maven's functionality, allowing you to automate and manage various aspects of your project's build lifecycle.

Dependency Management

- In Maven, a dependency is just another archive-JAR, ZIP, and so on-which our current project needs in order to compile, build, test, and/or run.
- These project dependencies are collectively specified in the pom.xml file, inside of a ***<dependencies>*** tag.
- When we run a maven build or execute a maven goal, these dependencies are resolved and then loaded from the local repository.
- If these dependencies are not present in the local repository, then Maven will download them from a remote repository and cache them in local.
- Dependency management is an important activity because to build a product we need several library files, plugins or some external dependencies. We have to be sure that all these components are compatible for building our product. This task can be taken care by dependency management.

Dependency Management

- Example:
- Transitive dependency
  - $mvn dependency:tree
- Excluding dependency
  - <exclusion> tag

Gradle

Introduction to Gradle

- Open source build automation tool

- The build automation tool is a tool that automates the creation of software build. Build automation is the process of automating the retrieval of source code, compiling it into binary code, executing automated tests and publishing it into a shared, centralized repository.

- Gradle is a general purpose build tool and its main focus is Java Projects.

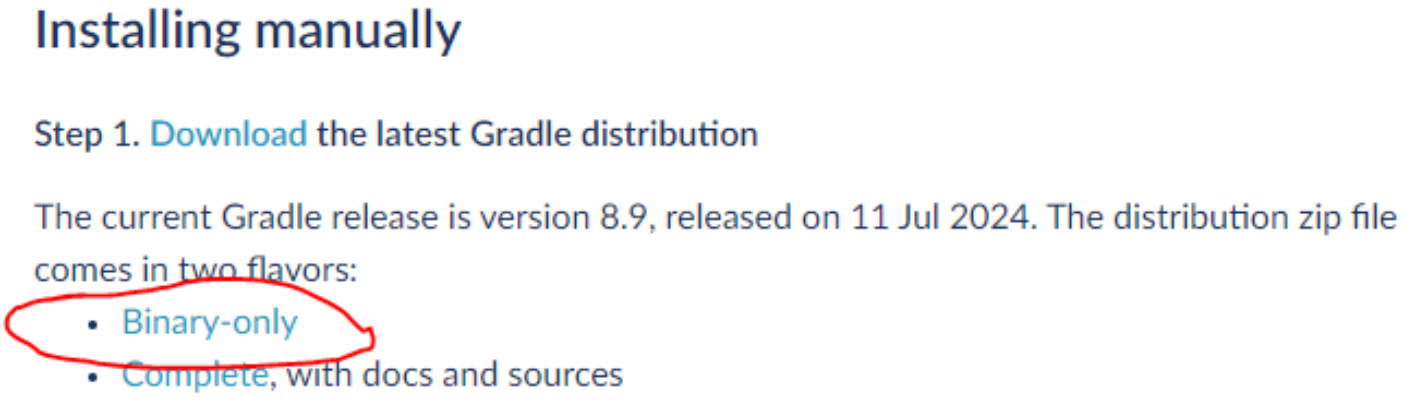
Gradle

Features of Gradle

- Free and Open Source tool
- High Performance
- Better Support compare to Ant tool
- Scaling : It can increase the productivity, from simple and single project to huge multi-project builds.
- Multi-project builds
- IDE Support: It has support for several IDEs. Gradle also generates the required solution files to load a project into Visual Studio.

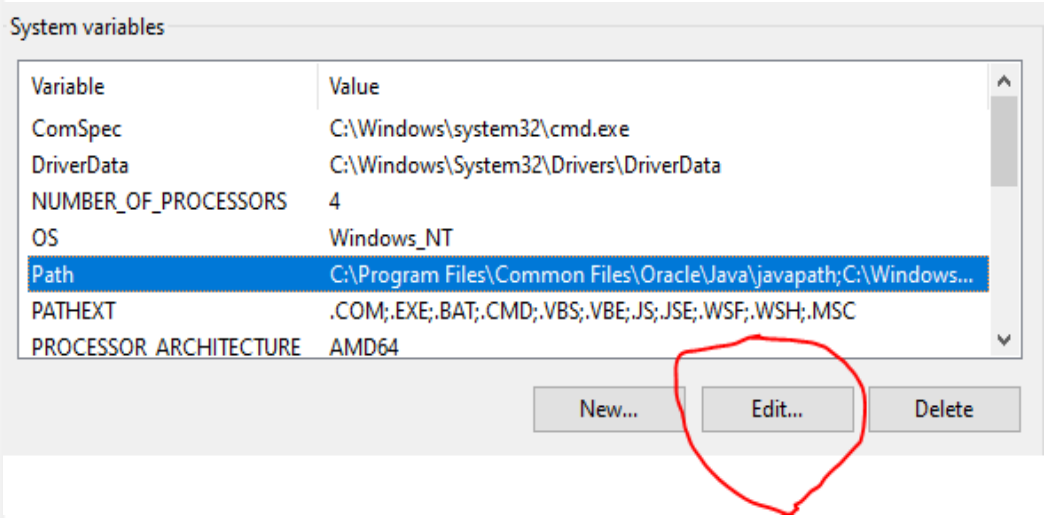
Installation of Gradle

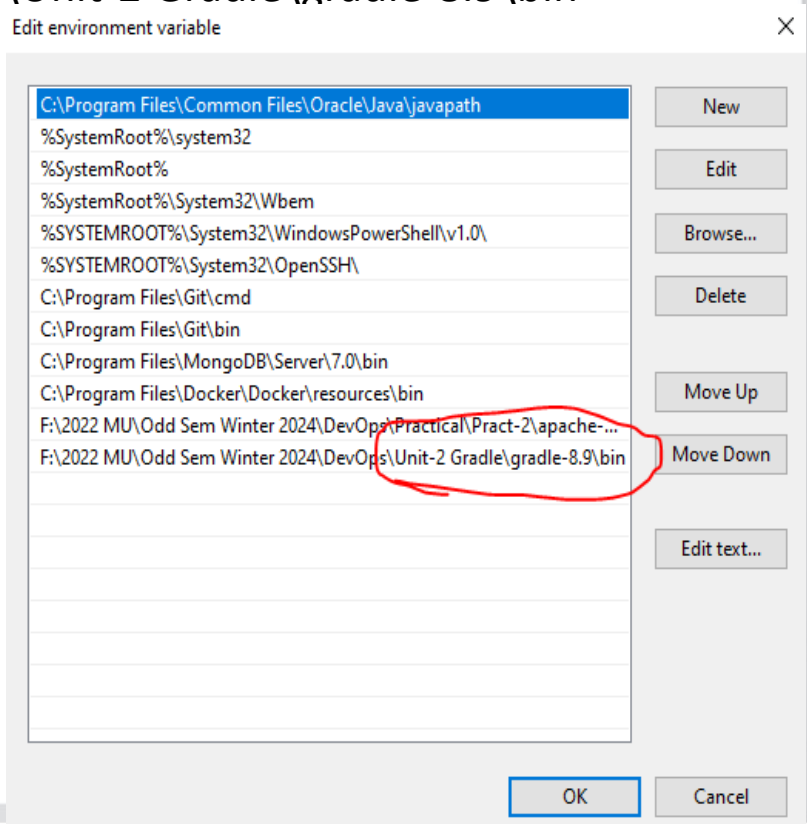
- Prerequisite
  - JDK 8 or above
- Download from
  - https://https://gradle.org/install/
- Configure the Environment variable.



Installation of Gradle

- Set Environment variable
  - F:\2022 MU\Odd Sem Winter 2024\DevOps\Unit-2 Gradle\gradle-8.9\bin





Installation of Gradle

- Open CMD and issue the command **gradle –v.** It will display the following screen.

```
C:\Users\Admin>gradle -v

Welcome to Gradle 8.9!

Here are the highlights of this release:
 - Enhanced Error and Warning Messages
 - IDE Integration Improvements
 - Daemon JVM Information

For more details see https://docs.gradle.org/8.9/release-notes.html


------------------------------------------------------------
Gradle 8.9
------------------------------------------------------------

Build time:    2024-07-11 14:37:41 UTC
Revision:      d536ef36a19186ccc596d8817123e5445f30fef8

Kotlin:        1.9.23
Groovy:        3.0.21
Ant:           Apache Ant(TM) version 1.10.13 compiled on January 4 2023
Launcher JVM:  22.0.1 (Oracle Corporation 22.0.1+8-16)
Daemon JVM:    C:\Program Files\Java\jdk-22 (no JDK specified, using current Java home)
OS:            Windows 10 10.0 amd64

C:\Users\Admin>
```

Understand Build using Gradle

Core Concepts

- **Project:** Gradle project represents the application that can be deployed to the staging environment.

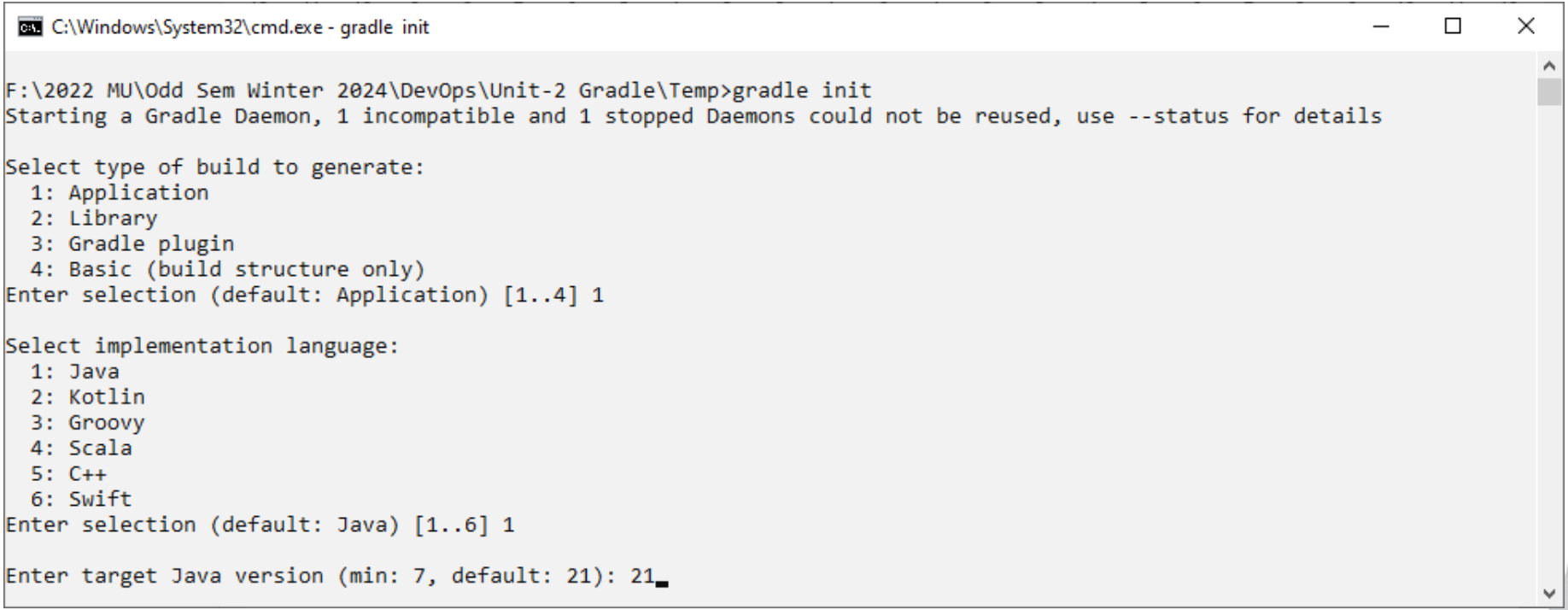
- **Task:** A task referes to a piece of work performed by a build. For example – the task can be creating a Jar file, compiling classes, or making JavaDoc.

- **Build Script:** Every Gradle build represents one or more projects. This build script is written using the domain specification language called groovy. This script is saved as ***build.gradle***.

Creating Gradle Project using CMD

Marwadi University
Marwadi Chandarana Group
NAAC A+

$gradle init



Creating Gradle Project using CMD

C:\Windows\System32\cmd.exe - gradle init

```
Enter target Java version (min: 7, default: 21): 21

Project name (default: Temp): Test

Select application structure:
  1: Single application project
  2: Application and library project
Enter selection (default: Single application project) [1..2] 1

Select build script DSL:
  1: Kotlin
  2: Groovy
Enter selection (default: Kotlin) [1..2] 1

Select test framework:
  1: JUnit 4
  2: TestNG
  3: Spock
  4: JUnit Jupiter
Enter selection (default: JUnit Jupiter) [1..4] 1

Generate build using new APIs and behavior (some features may change in the next minor release)? (default: no) [yes, no]
 no

<-------------> 0% EXECUTING [11m 51s]
> :init
```

Creating Gradle Project using CMD

Marwadi University
Marwadi Chandarana Group
NAAC A+

C:\Windows\System32\cmd.exe

```
Enter selection (default: Single application project) [1..2] 1

Select build script DSL:
  1: Kotlin
  2: Groovy
Enter selection (default: Kotlin) [1..2] 1

Select test framework:
  1: JUnit 4
  2: TestNG
  3: Spock
  4: JUnit Jupiter
Enter selection (default: JUnit Jupiter) [1..4] 1

Generate build using new APIs and behavior (some features may change in the next minor release)? (default: no) [yes, no]
 no


> Task :init
Learn more about Gradle by exploring our Samples at https://docs.gradle.org/8.9/samples/sample_building_java_applications.html

BUILD SUCCESSFUL in 12m 8s
1 actionable task: 1 executed
F:\2022 MU\Odd Sem Winter 2024\DevOps\Unit-2 Gradle\Temp>
```

Creating Gradle Project using CMD

Marwadi University
Marwadi Chandarana Group
NAAC A+

$gradle tasks

C:\Windows\System32\cmd.exe

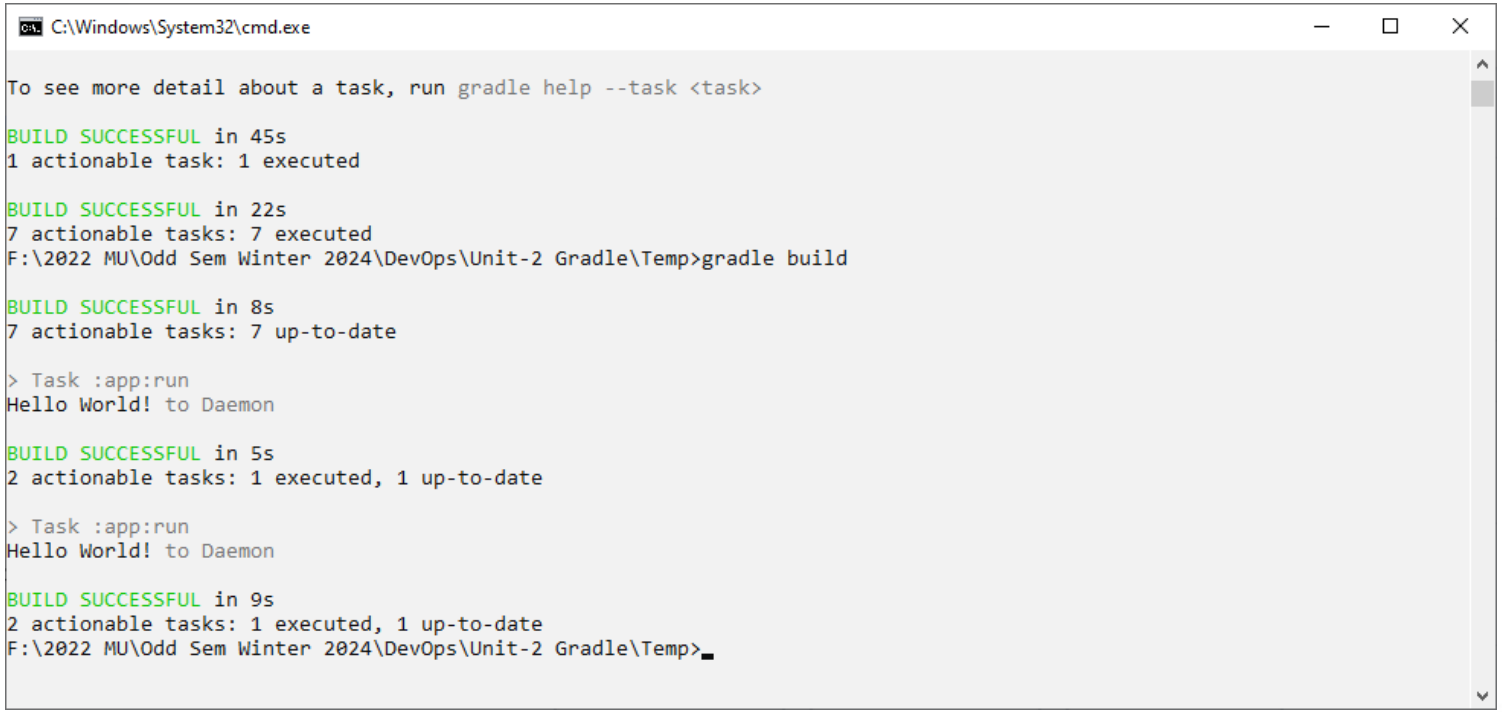
```
F:\2022 MU\Odd Sem Winter 2024\DevOps\Unit-2 Gradle\Temp>gradle tasks

> Task :tasks

------------------------------------------------------------
Tasks runnable from root project 'Test'
------------------------------------------------------------

Application tasks
-----------------
run - Runs this project as a JVM application

Build tasks
-----------
assemble - Assembles the outputs of this project.
build - Assembles and tests this project.
buildDependents - Assembles and tests this project and all projects that depend on it.
buildNeeded - Assembles and tests this project and all projects it depends on.
classes - Assembles main classes.
clean - Deletes the build directory.
jar - Assembles a jar archive containing the classes of the 'main' feature.
testClasses - Assembles test classes.

Build Setup tasks
-----------------
init - Initializes a new Gradle build.
```

Creating Gradle Project using CMD

Marwadi University
Marwadi Chandarana Group
NAAC A+

$gradle build

$gradle run



Understanding build using Gradle

Maven	Gradle
Maven is a tool used form generating Java based projects.	Gradle is a tool which can be used to develop domain-specific language projects.
It is uses the XML (pom.xml) to create project structure.	It uses Groovy/kotlin based Domain Specification Language (DSL) for creating the project structure.
Java compilation is compulsory for the Maven tool	Java compilation is not compulsory for the Gradle tool

THANK YOU