EXERCISE -1

1. Add maven-jar-plugin: Configure the plugin in pom.xml to specify the

2. Create Main Class: Write a simple MainClass with a main class . main method that outputs a message.

3. Package with Maven: Run mvn clean package to package the project into a JAR file.

4. Run the JAR: Use java -jar target/your-project-name.jar to run the packaged JAR and print the output.

Steps to Package the Project as a JAR and Run a Main Class

1. Add maven-jar-plugin to pom.xml : To package your Maven project as a JAR file and specify the pom.xml

Add the following configuration to your POM.XML

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-jar-plugin</artifactId>
  <version>3.1.0</version>
  <configuration>
    <!-- Specify the main class to be executed -->
    <archive>
      <manifestEntries>
        <Main-Class>org.example.Main</Main-Class> <!-- Replace with your actual main class -->
      </manifestEntries>
    </archive>
  </configuration>
</plugin>
```

2. Create a Main Class: In your src/main/java directory, create a class with a main method. For example, create a com.example :

```
package com.example;

public class Main {
  public static void main(String[] args) {
    System.out.println("Hello, World!");
  }
}
```

3. Package the Project into a JAR: After configuring the plugin and creating the MainClass , run the following Maven command to build the project and package it into a JAR file

Run the below command in IntelliJ Git Terminal

mvn clean package

4. Run the JAR File: Once the JAR is created, you can run it with the following command:

java -jar target/your-project-name.jar

Build and Run a Simple Java Application

Step 1: Modify build.gradle (Groovy DSL)

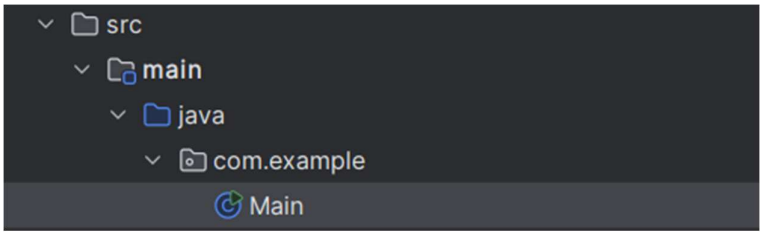
```
plugins {
    id 'application'
}

repositories {
    mavenCentral()
}

dependencies {
    testImplementation 'org.junit.jupiter:junit-jupiter:5.8.1'
}

task copyWebsite(type: Copy) {
    from 'src/main/resources'
    into 'docs'
}
tasks.named('run'){
    dependsOn 'copyWebsite'
}
application {
    mainClass = 'com.example.Main'
}
jar {
    manifest {
        attributes 'Main-Class': 'com.example.Main'  // This tells Java where to start execution
    }
}
```

Step 2: Create Main.java in src/main/java/com/example




```
package com.example;

public class Main {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

Step 3: Build and Run the Project

In IntelliJ IDEA, open the Gradle tool window (View → Tool Windows → Gradle).

Click Tasks > application > run .

Or run from terminal:

gradle run

 Hosting a Static Website on GitHub Pages

Step 1: Create a /docs Directory Create docs inside the root folder (not in src).

Add your HTML, CSS, and images inside /docs .

Step 2: Modify build.gradle to Copy Website Files (This is optional)

```
task copyWebsite(type: Copy) {
   from 'src/main/resources'
   into 'docs'
}
```

Step 3: Commit and Push to GitHub

1.git add .

2 git commit -m "Deploy website using Gradle"

3 git push origin main

Step 4: Enable GitHub Pages Go to GitHub Repo → Settings → Pages. Select the /docs folder as the source.

Your website will be hosted at

```
https://yourusername.github.io/repository-name/
```

Testing the Website using Selenium & TestNG in IntelliJ IDEA Step 1: Add Selenium & TestNG Dependencies in build.gradle

Step 1: Add Selenium & TestNG Dependencies in build.gradle

```
dependencies {
   testImplementation 'org.junit.jupiter:junit-jupiter:5.8.1'
}
```

```
test {
 useTestNG()
}
```

Step 2: Write a Test Script (src/test/java/org/test/WebpageTest.java)

```
import static org.testng.Assert.assertTrue;
```

```
public class WebpageTest {

private static WebDriver driver;

@BeforeTest

 public void openBrowser() throws InterruptedException {

driver = new ChromeDriver();

driver.manage().window().maximize();

Thread.sleep(2000);

driver.get("https://sauravsarkar-codersarcade.github.io/CA-GRADLE/");

}

@Test 24 public void titleValidationTest(){

String actualTitle = driver.getTitle();

String expectedTitle = "Tripillar Solutions";

Assert.assertEquals(actualTitle, expectedTitle);

assertTrue(true, "Title should contain 'Tripillar'");

}

@AfterTest

public void closeBrowser() throws InterruptedException {

Thread.sleep(1000);

driver.quit();

}

}
```

Step 3: Run the Tests Open the Gradle tool window in IntelliJ.

Click Tasks > verification > test . “Recommended“ Or

run from terminal:

```
gradle test // Fails sometimes due to terminal issues
```

Packaging a Gradle Project as a JAR

Step 1: Modify build.gradle for JAR Packaging

```
application {
   mainClass = 'com.example.Main'
}
jar {
   manifest {
      attributes 'Main-Class': 'com.example.Main'  // This tells Java where to start execution
   }
}
```

Step 2: Build and Package the JAR

```
gradle jar
```

Step 3: Run the JAR

```
java -jar build/libs/<my-gradle-project>.jar
```

POM.XML

```
<?xml version="1.0" encoding="UTF-8"?>
<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.example</groupId>
  <artifactId>SVITMVNTOGRDL</artifactId>
  <version>1.0-SNAPSHOT</version>

  <properties>
    <maven.compiler.source>17</maven.compiler.source>
    <maven.compiler.target>17</maven.compiler.target>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.8.1</version>
        <configuration>
          <source>1.8</source>  <!-- Java version -->
          <target>1.8</target>
        </configuration>
      </plugin>

      <!-- JAR Plugin -->
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-jar-plugin</artifactId>
        <version>3.2.0</version>
        <configuration>
          <archive>
            <manifest>
              <addClasspath>true</addClasspath>
              <mainClass>com.example.Main</mainClass> <!-- Replace with your main class -->
            </manifest>
          </archive>
        </configuration>
      </plugin>
    </plugins>
  </build>

</project>
```

BUILD.GRADLE

```
plugins {
    id 'java'
}

group = 'com.example'
version = '1.0-SNAPSHOT'

repositories {
    mavenCentral()
}

dependencies {
    testImplementation 'junit:junit:4.13.2'
}

jar {
    manifest {
        attributes(
                'Main-Class': 'com.example.Main'
        )
    }
}
```