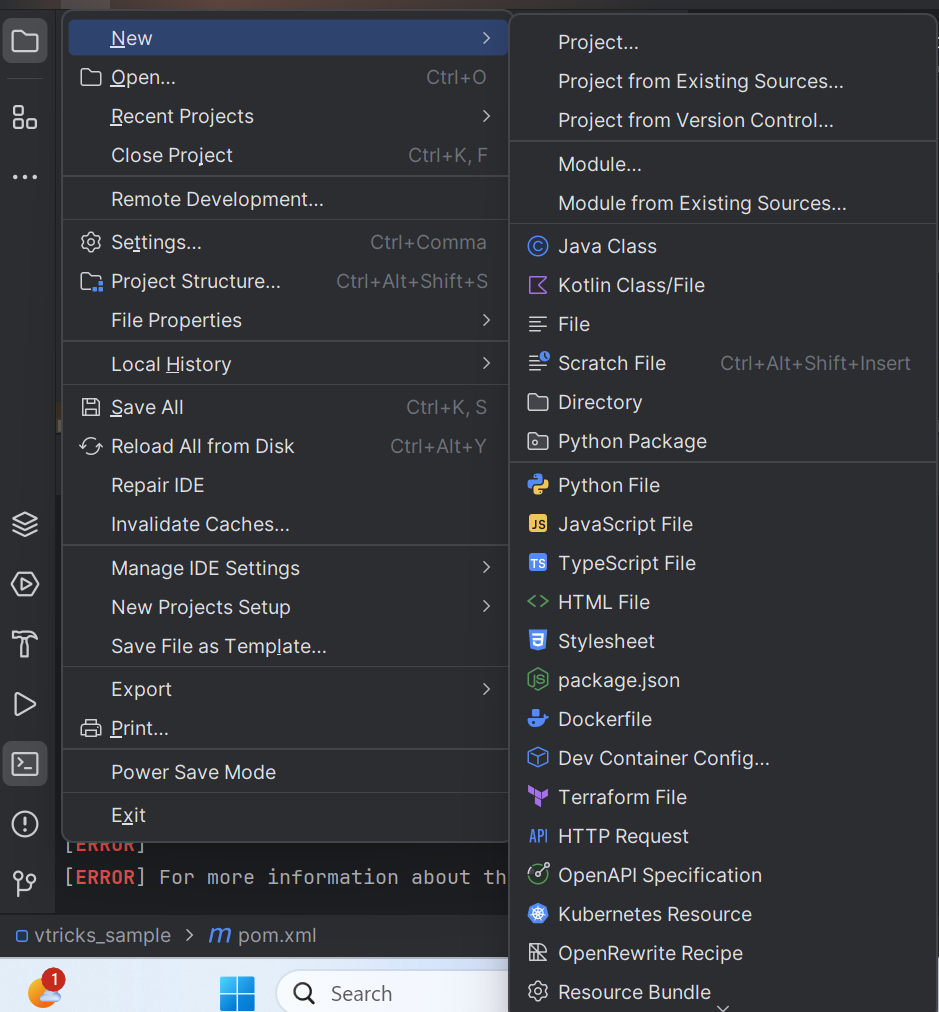
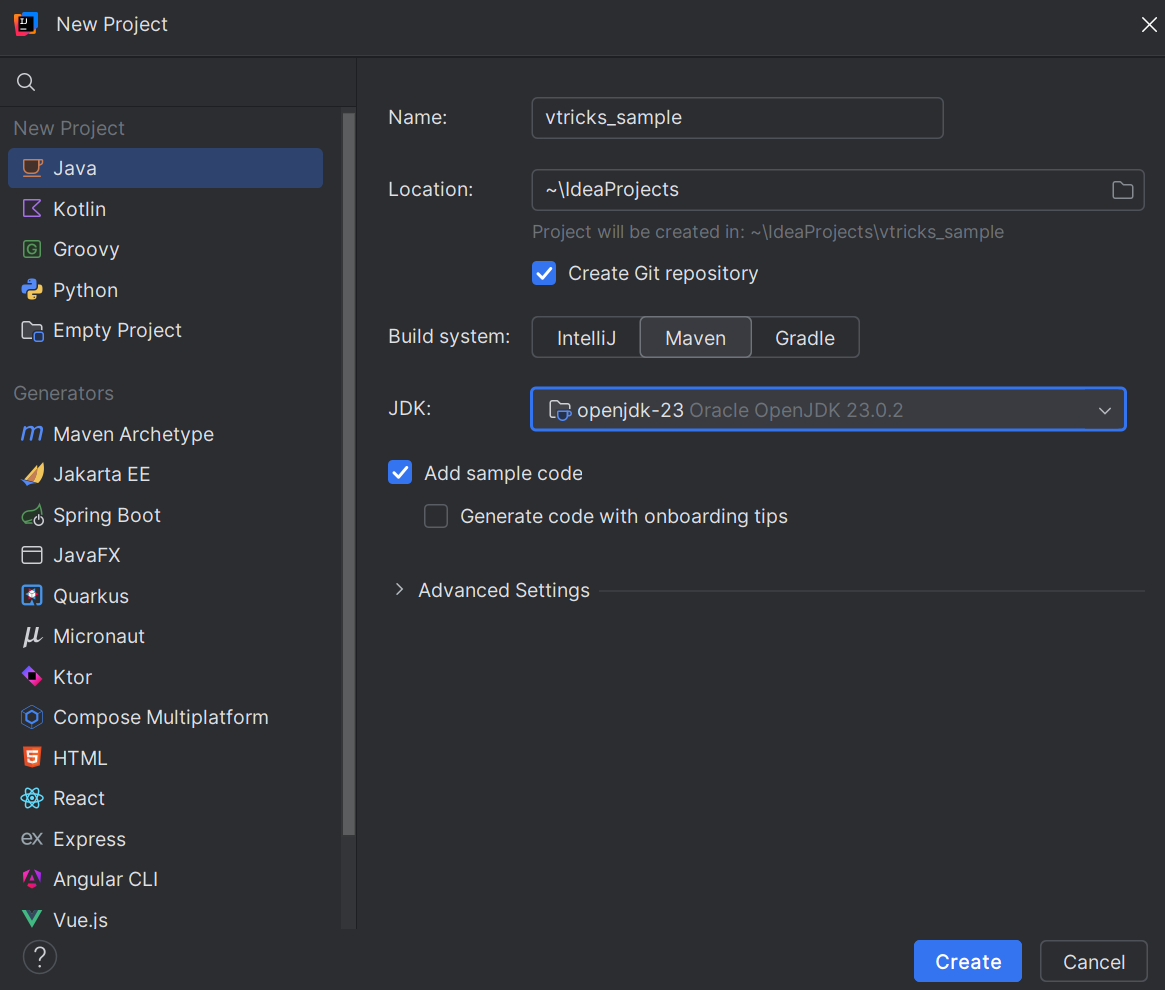
Experiment 2 : Working with Maven:Creating a maven Project, Understanding the POM File,Dependency Management and Plugins

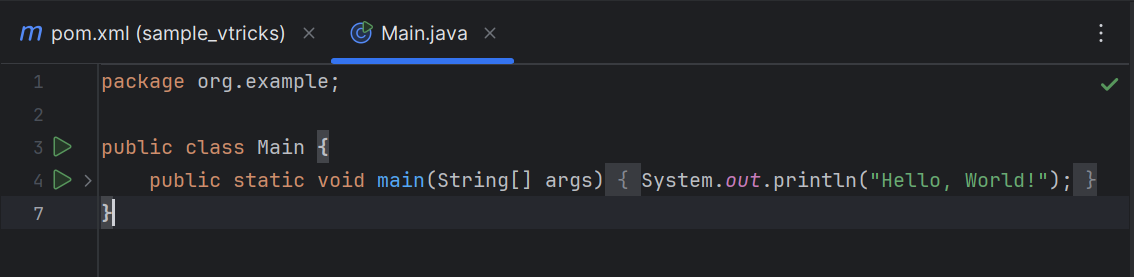
In intellij IDEA click on the NEW and Project



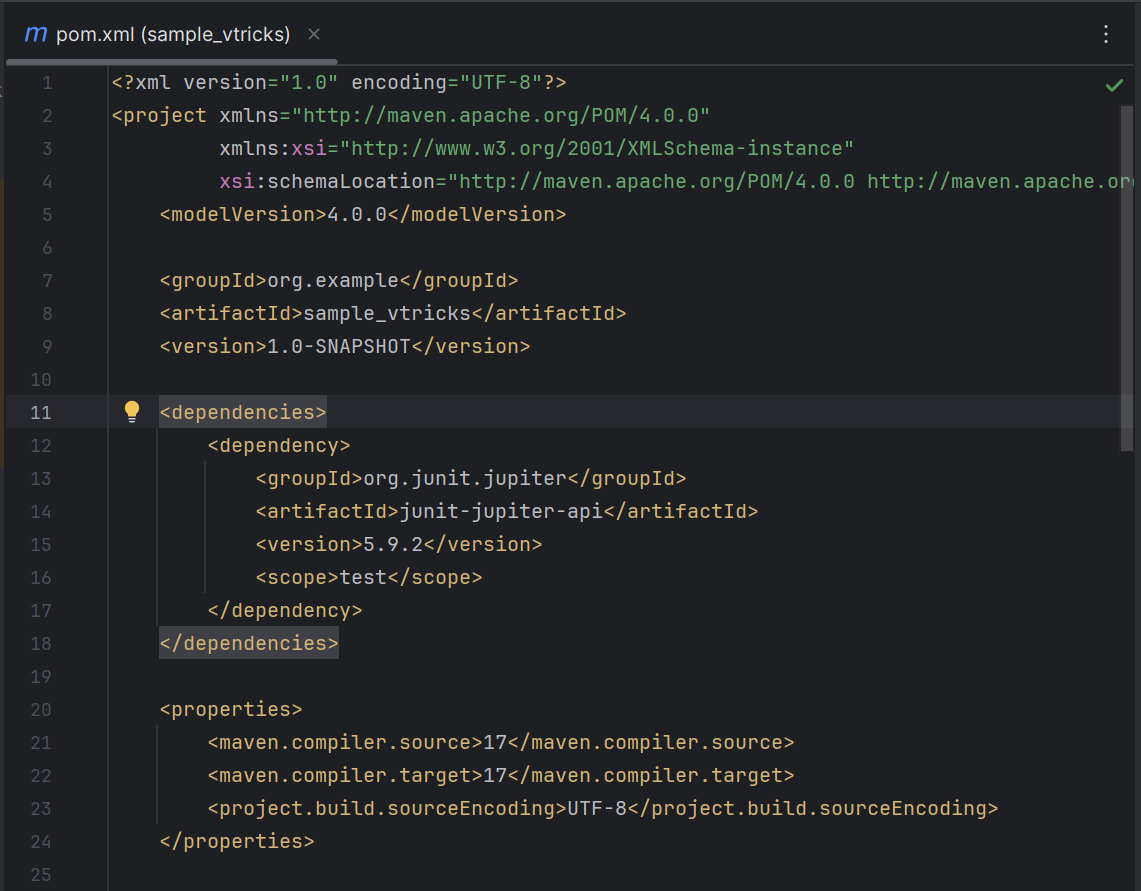
Provide Name for the Project, choose Build system as Maven and then click on Create



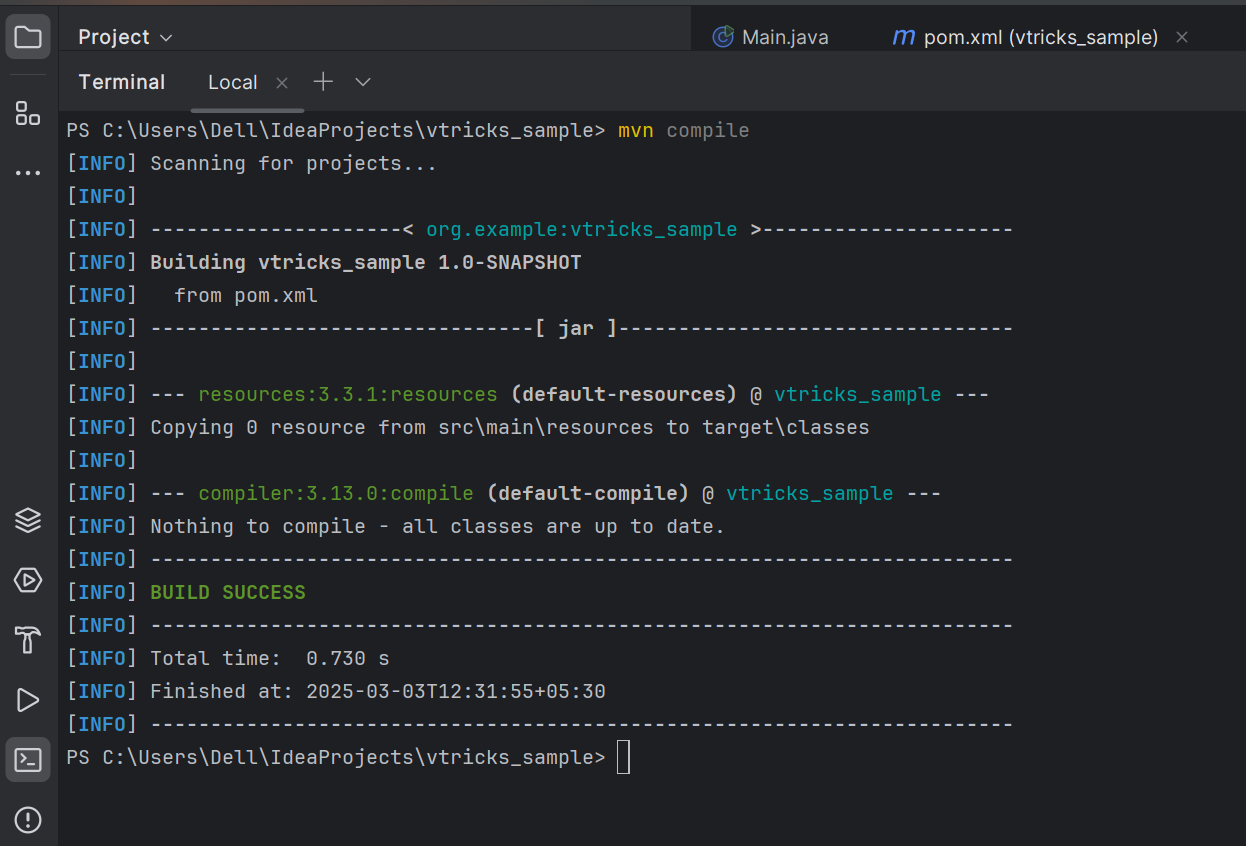
It will create a default main.java and pom.xml file



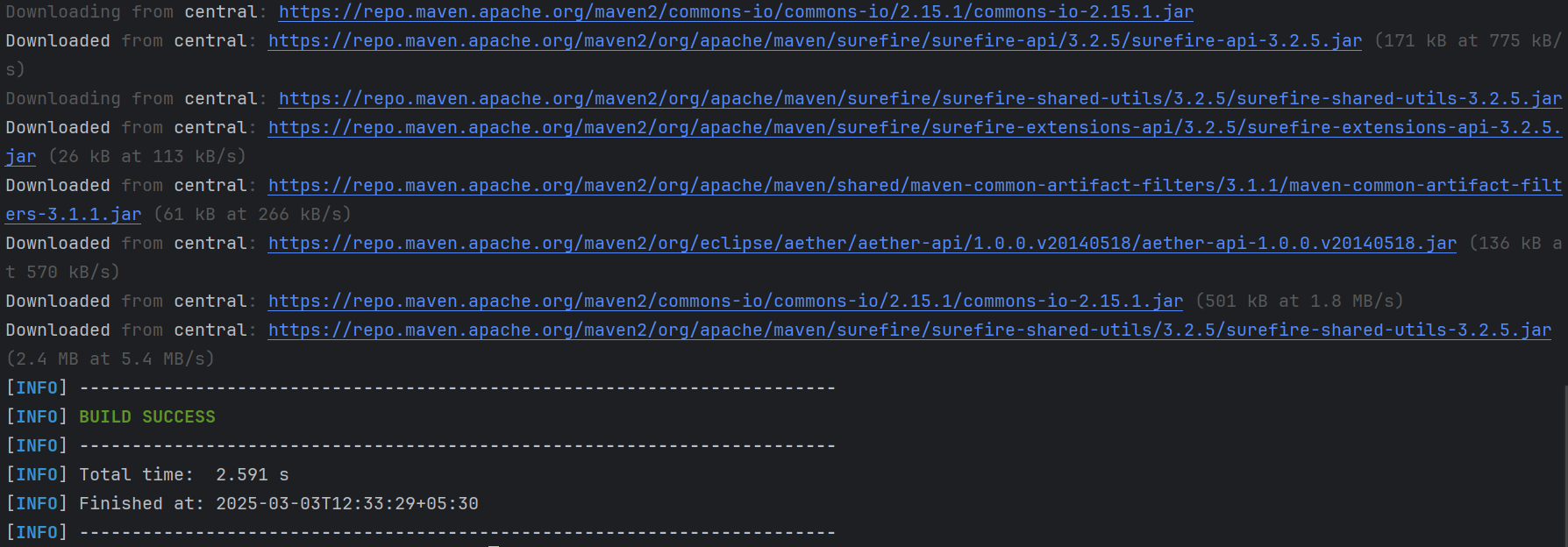
Here is the pom.xml file



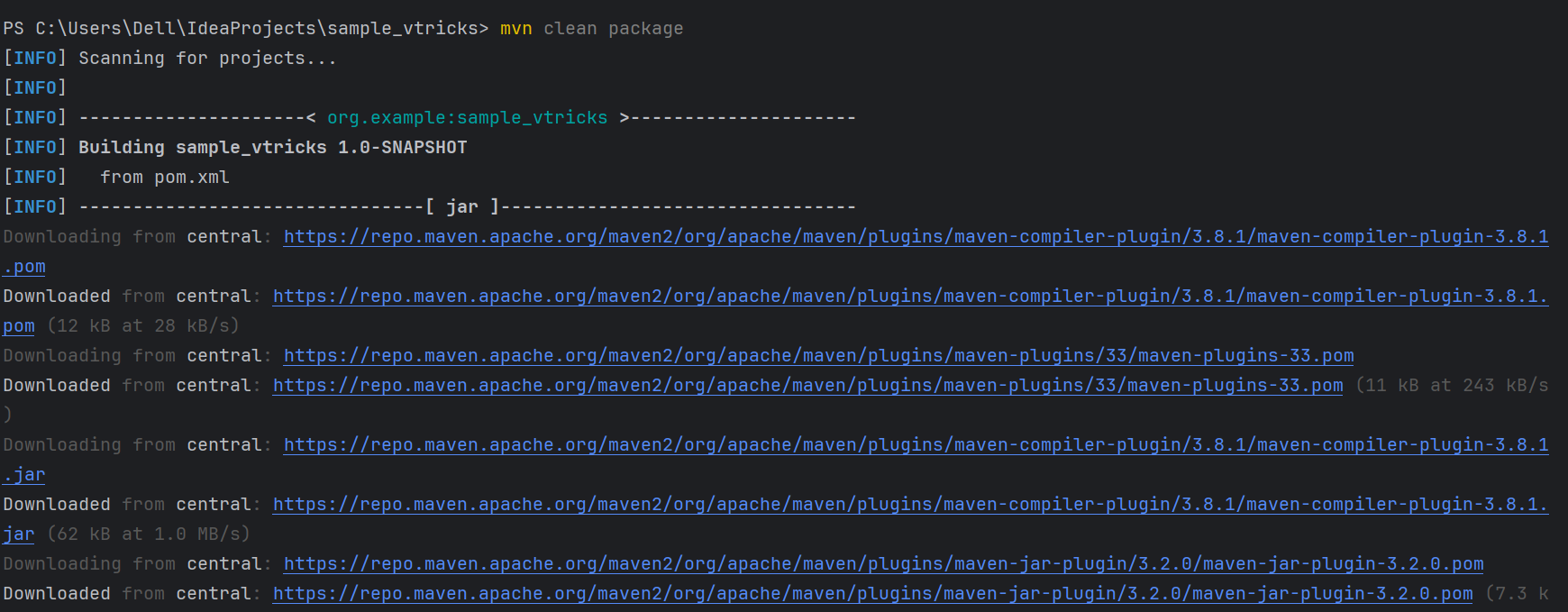
In terminal compile the maven file using command mvn compile

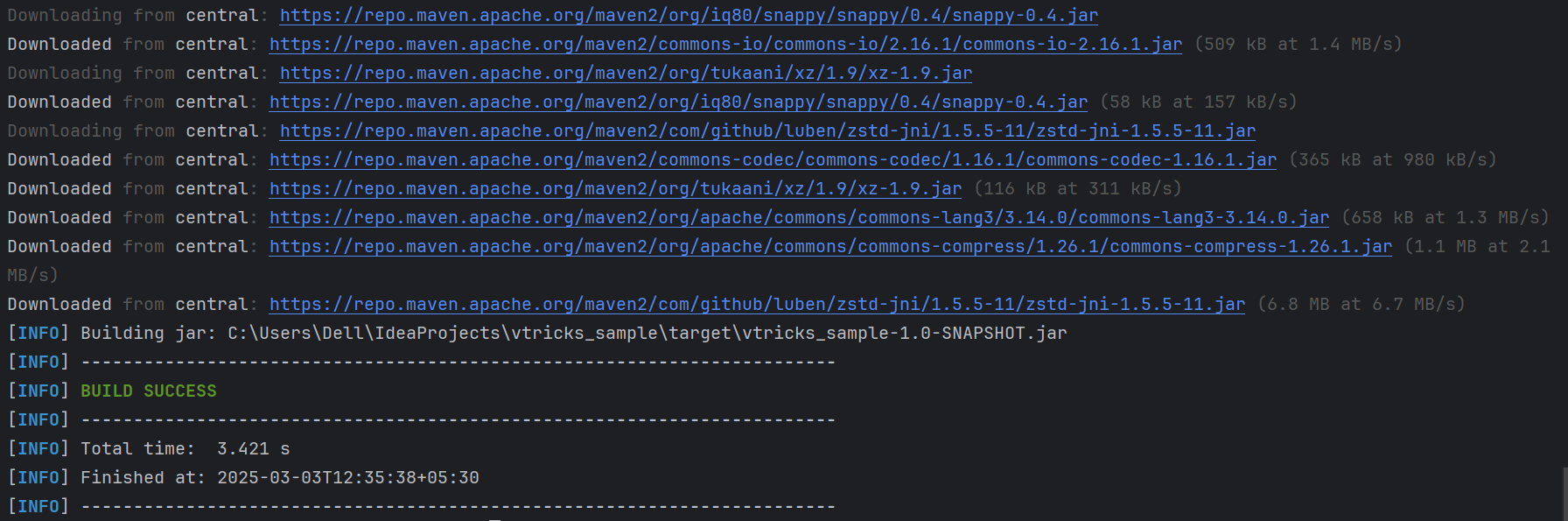


After compile test it using the mvn test



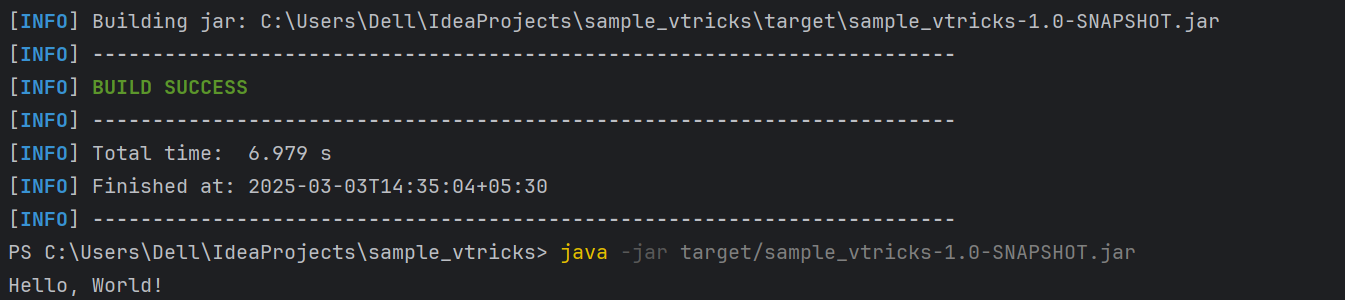
After testing we need to pack it using mvn clean package, it will create a jar file.





Here we can see the jar file created

To test the jar file use cd to the target folder wher you will get a jar file copy the jar file name and test it by java -jar target/sample\_vtricks-1.0-SNAPSHOT.jar it will give the output



pom.xml file

<?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>org.example</groupId>

<artifactId>sample\_vtricks</artifactId>

<version>1.0-SNAPSHOT</version>

<dependencies>

<dependency>

<groupId>org.junit.jupiter</groupId>

<artifactId>junit-jupiter-api</artifactId>

<version>5.9.2</version>

<scope>test</scope>

</dependency>

</dependencies>

<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>17</source>

<target>17</target>

</configuration>

</plugin>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-jar-plugin</artifactId>

<version>3.2.0</version>

<configuration>

<archive>

<manifest>

<mainClass>org.example.Main</mainClass>

</manifest>

</archive>

</configuration>

</plugin>

</plugins>

</build>

</project>

The provided pom.xml file is the Project Object Model (POM) configuration file for a Maven project. It defines project dependencies, build configuration, and other important settings for a Java project.

### Understanding the pom.xml File

#### 1. ****Project Information:****

* <groupId>:
  + This is the identifier for the group or organization that is managing the project. In this case, it is org.example.
* <artifactId>:
  + This represents the unique identifier for the artifact (project). Here, it is sample\_vtricks.
* <version>:
  + This indicates the version of the artifact. In this case, it's set to 1.0-SNAPSHOT. The SNAPSHOT version implies that this is a development version and may change.

#### 2. ****Dependencies Section:****

The <dependencies> section is where you define the external libraries that the project needs. In this case:

<dependency>:

groupId: The group identifier for the dependency. Here, it is org.junit.jupiter, which is the group for JUnit 5 libraries.

artifactId: The name of the artifact. Here, it's junit-jupiter-api, which is the API module for JUnit 5.

version: The version of the library. In this case, version 5.9.2 is used.

scope: Specifies when the dependency is available. Here, it is test, meaning the JUnit library is only available during test compilation and runtime.

3. Properties Section:

**<properties>**

**<maven.compiler.source>17</maven.compiler.source>**

**<maven.compiler.target>17</maven.compiler.target>**

**<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding></properties>**

**<maven.compiler.source> and <maven.compiler.target>:**

These define the version of the Java compiler to be used. In this case, both are set to 17, indicating that the project is written for Java 17.

**<project.build.sourceEncoding>:**

This specifies the character encoding for the project. Here, it is set to UTF-8, ensuring that all files are read and written with UTF-8 encoding.

4. Build Section:

The <build> section configures the build process. Here, two plugins are defined:

a) Maven Compiler Plugin:

**<plugin>**

**<groupId>org.apache.maven.plugins</groupId>**

**<artifactId>maven-compiler-plugin</artifactId>**

**<version>3.8.1</version>**

**<configuration>**

**<source>17</source>**

**<target>17</target>**

**</configuration></plugin>**

maven-compiler-plugin:

This plugin is responsible for compiling Java sources.

<source> and <target>: These specify the source and target Java versions for the compilation process. Both are set to 17, ensuring that the code is compiled using Java 17 features and bytecode compatibility.

b) Maven Jar Plugin:

**<plugin>**

**<groupId>org.apache.maven.plugins</groupId>**

**<artifactId>maven-jar-plugin</artifactId>**

**<version>3.2.0</version>**

**<configuration>**

**<archive>**

**<manifest>**

**<mainClass>org.example.Main</mainClass>**

**</manifest>**

**</archive>**

**</configuration></plugin>**

maven-jar-plugin:

This plugin is used to package the project into a JAR (Java ARchive) file.

The <mainClass> element is used to define the entry point for the application. In this case, the org.example.Main class is specified, meaning it is the class with the public static void main(String[] args) method to be executed when the JAR file is run.

**Summary of Key Sections:**

Dependencies: The junit-jupiter-api dependency is included with the test scope, meaning it will only be used for testing.

Compiler Settings: The source and target versions for the compiler are set to Java 17.

Plugins:

The maven-compiler-plugin ensures Java 17 compatibility during compilation.

The maven-jar-plugin allows for packaging the application as a JAR and defines the main class for the JAR.

**Dependency Management**

The pom.xml manages dependencies through Maven's dependency management system. Maven downloads and caches the dependencies from remote repositories and ensures that the correct versions are used. Dependencies can be declared in the <dependencies> section, and their scope (such as test, compile, runtime, etc.) determines when they are included in the build process.

In this pom.xml, JUnit is included for testing, and Maven ensures that the correct version of the JUnit API is available during testing.

**Plugins**

Plugins in Maven automate tasks during the build lifecycle.

Compiler Plugin: Ensures that the code is compiled with the correct version of Java.

Jar Plugin: Packages the project as a JAR file and allows specifying the main class for execution.

Maven plugins help control the flow of the build process, such as compiling code, packaging it, running tests, and generating reports. Each plugin can be configured to perform specific tasks like setting compiler versions or including a specific main class in the generated JAR file.

This pom.xml file is a standard setup for a Java 17 project with unit tests using JUnit 5, packaged as a JAR file.