**What is Maven?**

Maven is a build automation tool used primarily for Java projects. It helps developers manage dependencies, compile code, and build artifacts.

**What is Maven Global Server?**

The Maven Global Server is a public repository that contains JAR files for many popular Java libraries. Developers can connect to the Maven Global Server to download the JAR files they need for their projects.

**What is Maven Local Repository?**

The Maven Local Repository is a directory on the developer's machine where Maven stores JAR files that have been downloaded from the Maven Global Server. Maven will automatically download JAR files to the Maven Local Repository the first time they are needed, and then use them from the local repository for subsequent builds.

**Why use Maven?**

Maven has a number of benefits, including:

* **Dependency management:** Maven helps developers manage the dependencies for their projects. This includes automatically downloading the JAR files needed for the project, and ensuring that the correct versions of the dependencies are used.
* **Build automation:** Maven automates the build process for Java projects. This includes compiling the code, running unit tests, and creating artifacts.
* **Reproducibility:** Maven builds are reproducible, meaning that the same build will produce the same results on any machine.

**Configuring Maven**

To configure Maven, you need to install the Maven software and set up some environment variables.

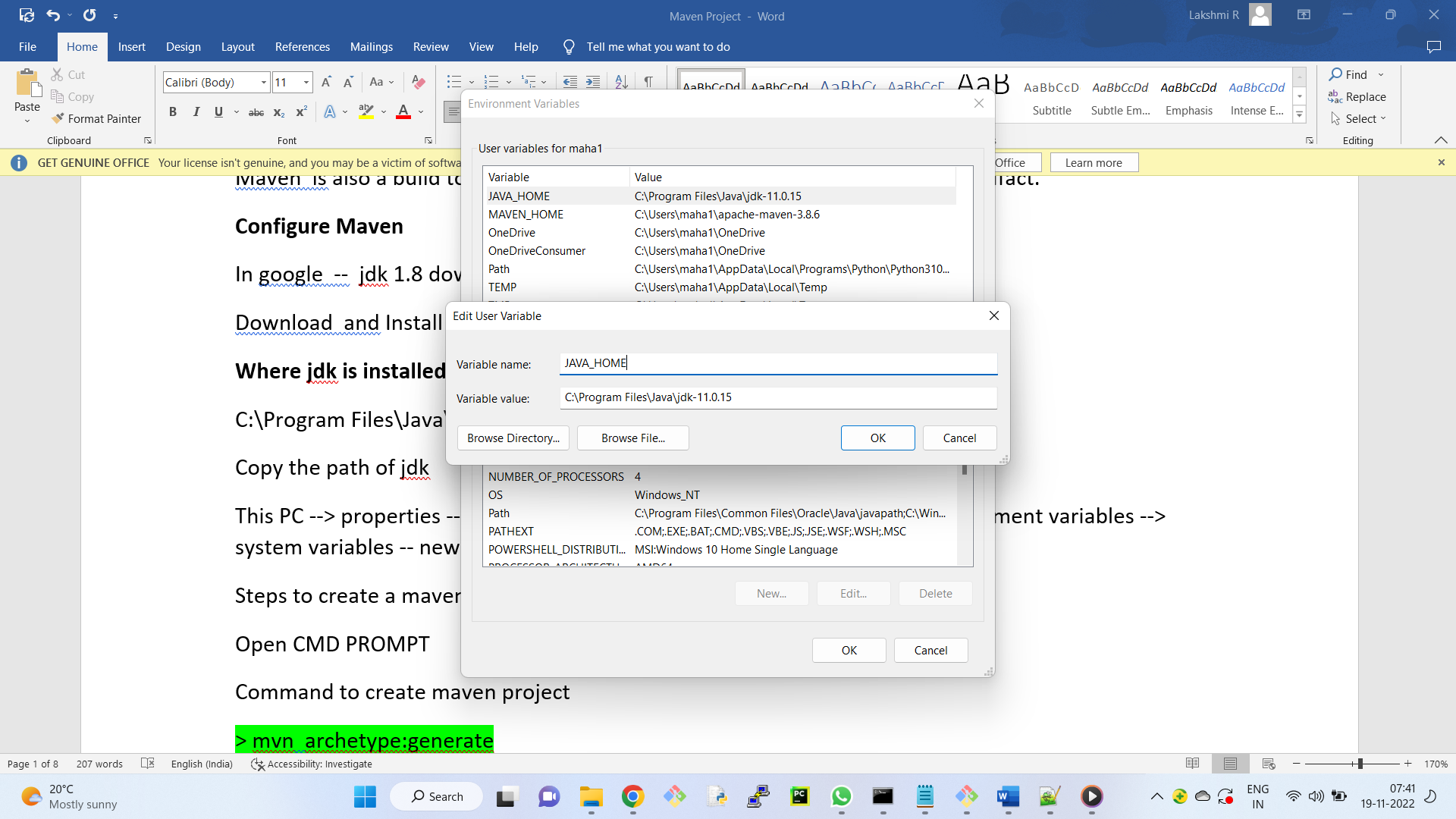
Browse JDK installation on google. Download and Install JDK

**Where JDK is installed?**

C:\Program Files\Java\jdk-11.0.15

Copy the path of JDK

This PC --> properties -- Advanced system settings --> advanced tab---> Environment variables --> system variables – new



Variable name - JAVA\_HOME

Variable value - C:\Program Files\Java\jdk-11.0.15

Ok

Now, go to "path" variable --- Edit -- click on new button

C:\Program Files\Java\jdk11.0.15\bin

Ok

++++++++++++

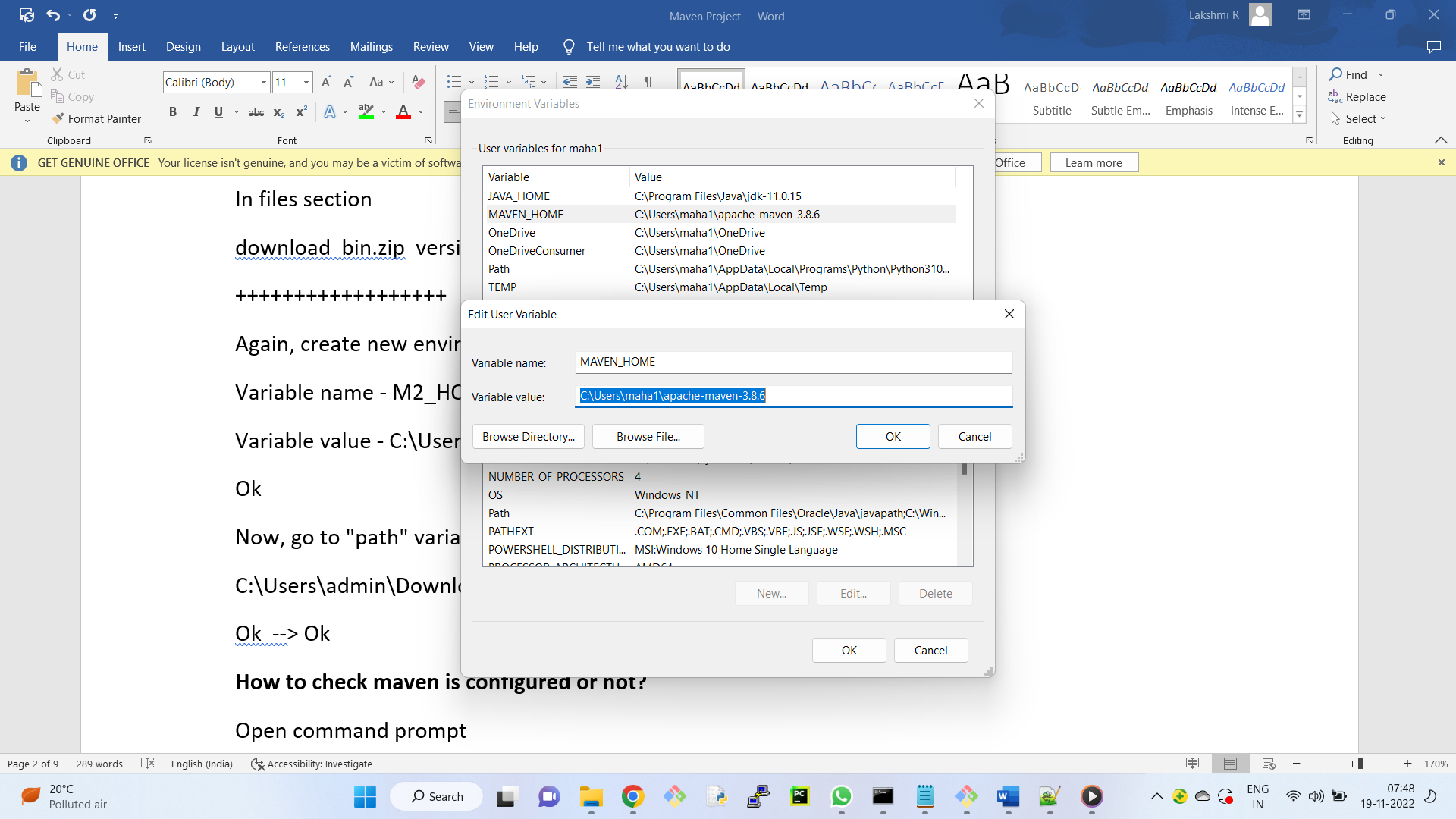
Download Apache Maven

In files section

download bin.zip version and extract

++++++++++++++++++

Again, create new environment variable



Variable name - MAVEN\_HOME

Variable value - C:\Users\maha1\apache-maven-3.8.6

Ok

Now, go to "path" variable --- Edit -- click on new button

C:\Users\admin\Downloads\apache-maven-3.8.6\bin

Ok --> Ok

**How to check maven is configured or not?**

Open command prompt

> mvn --version

++++++++++++++

**What is an artifact?**

An artifact in Maven is a file that is produced by the Maven build process. It can be a JAR file, a WAR file, or any other type of file. Artifacts are identified by their coordinates, which are made up of Group Id, Artifact Id, Version, Packaging (JAR, WAR, EAR), Classification (optional) which is a string that can be used to distinguish different versions of the same artifact.

**What is a Maven number?**

A Maven number is a unique identifier for an artifact. It is made up of the following three parts:

* **Group ID:** This is the identifier for the organization that created the artifact.
* **Artifact ID:** This is the identifier for the artifact itself.
* **Version:**This is the version of the artifact.

**What is a group ID? (is nothing but project structure)**

The group ID in Maven is a unique identifier for the organization that created the artifact. It is typically a reverse domain name, such as com.example

The group ID is used to structure Maven projects in a logical way. For example, a company might have the following group ID:

com.example

Within this group ID, the company could create different artifacts for different projects, such as:

com.example:my-artifact

com.example:my-other-artifact

This structure makes it easy to find and manage artifacts.

**Example:**XML

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

<version>5.3.23</version>

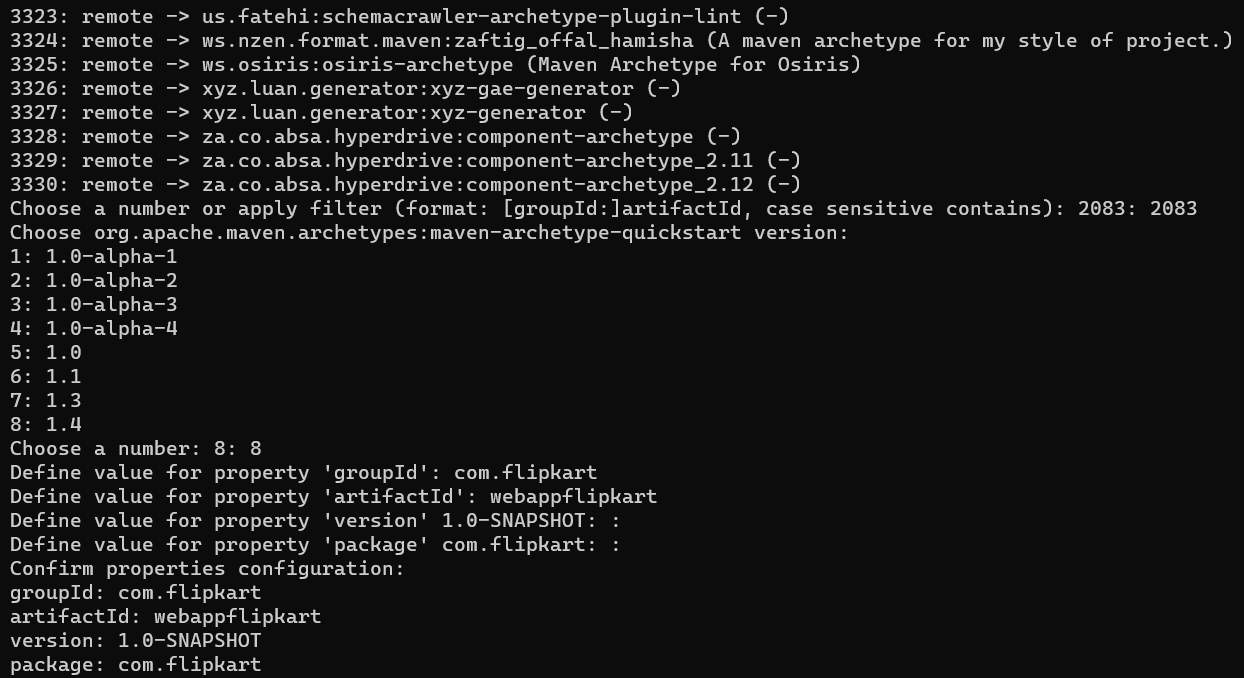
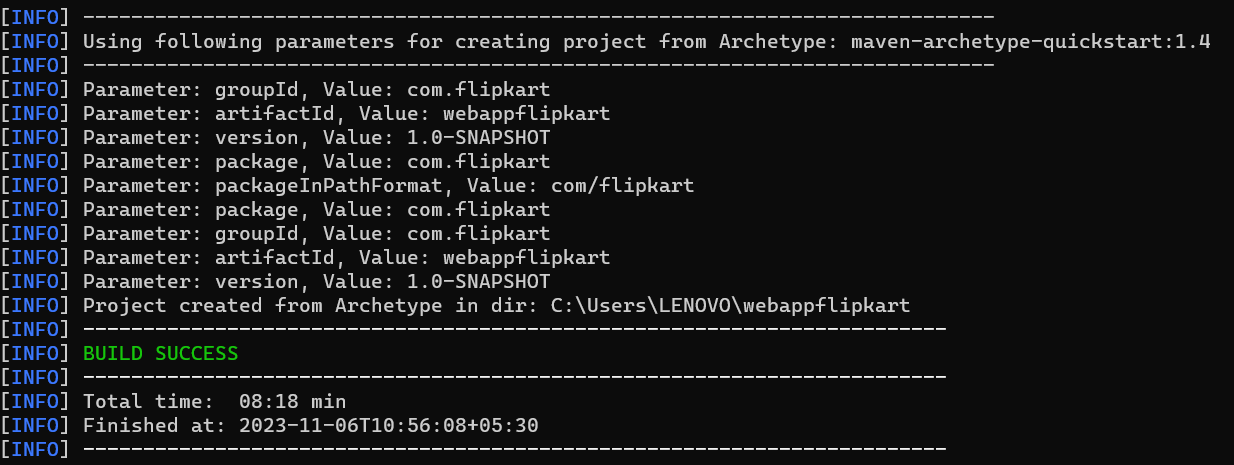
</dependency>

In this example, the group ID is org.springframework. This tells Maven to look for the spring-core artifact in the org.springframework repository.

**Creating a Maven Project**

To create a Maven project, you can use the following command:

mvn archetype:generate

This command will prompt you to choose a type of project to create. Once you have chosen a type, Maven will create a new project directory with the necessary files.  
  
In our file location we can find src folder and pom.xml file. Inside the src folder there are you can see main and test folders.

Code created by the developer will be in main folder.

Code created by the developer for unit testing will be in test folder.

**POM -- stands for project Object model.**

It is an xml file, which will store all the external API information.

Open **GitBash**

change directory to webappflipkart

$ git init

$ git status

$ git add .

$ git commit -m "first commit"

**Adding 3rd Party APIs to Maven Project**

To add a 3rd party API to your Maven project, you need to add a dependency to your pom.xml file.

**https://search.maven.org/** -- maven global server

Developer will search for the required API here. For instance, we can choose mysql and google map.

copy the dependency tag and paste in pom.xml

Once you have added the dependency to your pom.xml file, you need to run the following command:

mvn compile

This command will download the API from Maven Global server to Maven local repository.

**Building and Packaging the Maven Project**

To build and package your Maven project, you can use the following command:

mvn package

This command will compile the code, run unit tests, and create an artifact. The artifact will be a JAR file in the target/ directory that contains your project's code and dependencies.

Open git bash

Go to working directory

$ git status

$ git add .

$ git commit -m "second commit"

Copy code in App.java, lets create a new java file, save as Sample.java

package com.flipkart;

public class Sample

{

    public static void main( String[] args )

    {

        System.out.println("Hello World");

    }

}

Now, we need to compile the java files . In command prompt

> mvn compile

Observation:

A new folder target is created in webappflipkart, we can see the class files.

Similarly

goto src/test/java/com/flipkart

create another java file with the name SampleTest.java

Add one line of code at the end ( for our reference )

System.out.println("Testing Passed");

package com.flipkart;

import static org.junit.Assert.assertTrue;

import org.junit.Test;

public class SampleTest

{

    @Test

    public void shouldAnswerWithTrue()

    {

        assertTrue( true );

        System.out.println("Testing Passed");

    }

}

Now, we need to convert the unit testing code into .class file, for that we have command

> mvn test

Observation:

In folder target , test-classes will be created , which contains .class files



To create Artifact

> mvn package

Where is the location of the artifact?

In your target folder you can see webappflipkart-1.0-SNAPSHOT.jar

