**5. Set Up Eclipse to Run Selenium Tests and Execute Selenium Test Case Using Java Project**

**Step 1: Install Eclipse**

1. Download and install Eclipse IDE.

**Step 2: Install Selenium WebDriver**

1. Create a new Java project in Eclipse:
   * Go to **File** > **New** > **Java Project**.
   * Name your project (e.g., SeleniumJavaProject).
2. Add Selenium WebDriver libraries:
   * Download Selenium WebDriver from Selenium HQ.
   * Extract the downloaded ZIP file.
   * In Eclipse, right-click your project > **Build Path** > **Configure Build Path**.
   * Go to the **Libraries** tab and click **Add External JARs**. Add the extracted Selenium JAR files.

**Step 3: Write a Selenium Test Case**

1. Create a new Java class:
   * Right-click on src > **New** > **Class** (e.g., GoogleSearchTest).

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.chrome.ChromeDriver;

import org.openqa.selenium.By;

public class GoogleSearchTest {

public static void main(String[] args) {

// Set the path for ChromeDriver

System.setProperty("webdriver.chrome.driver", "path/to/chromedriver.exe");

WebDriver driver = new ChromeDriver();

driver.get("https://www.google.com");

// Search for Selenium

driver.findElement(By.name("q")).sendKeys("Selenium");

driver.findElement(By.name("btnK")).click();

// Close the browser

driver.quit();

}

}

**Run the Test**: Right-click the class > **Run As** > **Java Application**.

### ****6. Set Up Eclipse to Run Selenium Tests and Execute Selenium Test Case Using Maven Project****

#### ****Step 1: Install Eclipse and Maven****

1. Ensure you have Eclipse IDE installed.
2. Install Maven (if not already done).

#### ****Step 2: Create Maven Project****

1. Go to **File** > **New** > **Maven Project**.
2. Select the workspace location and click **Next**.
3. Choose **Quick Start** and click **Next**.
4. Fill in the Group Id and Artifact Id, and click **Finish**.

#### ****Step 3: Add Selenium Dependencies in**** pom.xml

Add the following dependencies to your pom.xml file:

xml

Copy code

<dependencies>

<dependency>

<groupId>org.seleniumhq.selenium</groupId>

<artifactId>selenium-java</artifactId>

<version>4.0.0</version> <!-- Check for the latest version -->

</dependency>

<dependency>

<groupId>org.testng</groupId>

<artifactId>testng</artifactId>

<version>7.4.0</version> <!-- Check for the latest version -->

<scope>test</scope>

</dependency>

</dependencies>

#### ****Step 4: Write a Selenium Test Case****

1. Create a new Java class (e.g., GoogleSearchTest).
2. Add the following code:

java

Copy code

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.chrome.ChromeDriver;

import org.openqa.selenium.By;

import org.testng.annotations.AfterClass;

import org.testng.annotations.BeforeClass;

import org.testng.annotations.Test;

public class GoogleSearchTest {

WebDriver driver;

@BeforeClass

public void setUp() {

System.setProperty("webdriver.chrome.driver", "path/to/chromedriver.exe");

driver = new ChromeDriver();

}

@Test

public void searchGoogle() {

driver.get("https://www.google.com");

driver.findElement(By.name("q")).sendKeys("Selenium");

driver.findElement(By.name("btnK")).click();

}

@AfterClass

public void tearDown() {

driver.quit();

}

}

#### ****Step 5: Run the Test****

* Right-click on the project > **Run As** > **Maven test**.

### ****7. Set Up Eclipse to Run Selenium – TestNG Tests and Execute Selenium Test Case Using Maven Project with TestNG****

This process is similar to the previous step, as it’s already set up with TestNG.

* Ensure you have the TestNG dependency in your pom.xml as shown above.
* The code structure provided in ***step 6*** can be directly used here, as it already incorporates TestNG.

### ****8. Install Docker and Create Docker Image for Python Application****

#### ****Step 1: Install Docker****

* Follow the official instructions to install Docker for your OS from Docker Installation.

#### ****Step 2: Create a Python Application****

1. Create a directory for your Python application (e.g., python-app).
2. Inside the directory, create a Python script (app.py):

# app.py

from flask import Flask

app = Flask(\_\_name\_\_)

@app.route('/')

def hello():

return "Hello, World!"

if \_\_name\_\_ == '\_\_main\_\_':

app.run(host='0.0.0.0', port=5000)

Create a requirements.txt file for dependencies:

Flask

**Step 3: Create a Dockerfile**

1. Create a file named Dockerfile in the same directory:

# Dockerfile

FROM python:3.9

WORKDIR /app

COPY requirements.txt .

RUN pip install --no-cache-dir -r requirements.txt

COPY . .

CMD ["python", "app.py"]

**Step 4: Build the Docker Image**

* Open a terminal and navigate to your application directory, then run:

docker build -t python-app .

**Step 5: Run the Docker Container**

* To run the container:

docker run -p 5000:5000 python-app

### ****9. Install Docker and Create Docker Image for Java Application****

#### ****Step 1: Create a Java Application****

1. Create a directory for your Java application (e.g., java-app).
2. Inside the directory, create a simple Java application (HelloWorld.java):

// HelloWorld.java

public class HelloWorld {

public static void main(String[] args) {

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

}

}

1. Create a Dockerfile in the same directory:

# Dockerfile

FROM openjdk:11-jre-slim

COPY HelloWorld.java /usr/src/myapp/

WORKDIR /usr/src/myapp/

RUN javac HelloWorld.java

CMD ["java", "HelloWorld"]

#### ****Step 2: Build the Docker Image****

* Open a terminal, navigate to your application directory, then run:

docker build -t java-app .

#### ****Step 3: Run the Docker Container****

* To run the container:

docker run java-app

### ****10. Install Docker and Create Docker Image for Simple Web Application****

#### ****Step 1: Create a Simple Web Application****

1. Create a directory for your web application (e.g., simple-web-app).
2. Inside the directory, create an index.html file:

html

Copy code

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Simple Web App</title>

</head>

<body>

<h1>Hello, World!</h1>

</body>

</html>

1. Create a Dockerfile in the same directory:

# Dockerfile

FROM nginx:alpine

COPY index.html /usr/share/nginx/html

#### ****Step 2: Build the Docker Image****

* Open a terminal, navigate to your web application directory, then run:

docker build -t simple-web-app .

#### ****Step 3: Run the Docker Container****

* To run the container:

docker run -p 80:80 simple-web-app