

Lesson 07 Demo 06

Creating and Executing Scenario-Based Tests Using Generative AI

Objective: To create and execute comprehensive scenario-based test scripts using Generative AI tools, focusing on real-world user interactions within a payment gateway system to enhance efficiency and accuracy in software development

Tools required: VS Code, Microsoft Copilot, Eclipse, Selenium and Selenium Web Driver

Prerequisites: Basic knowledge and understanding of scenario-based testing and writing test scripts using Selenium

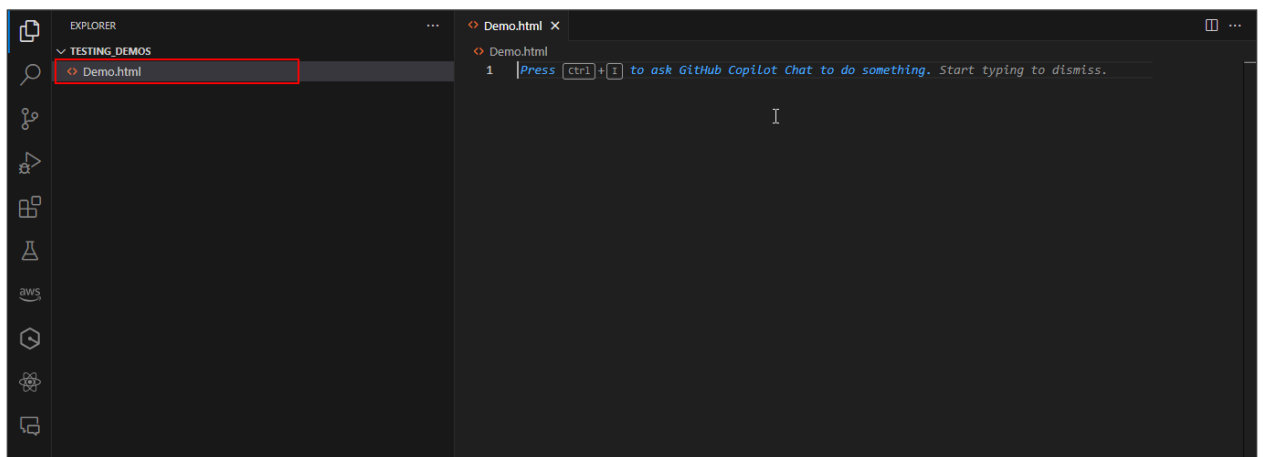
Steps to be followed:

1. Generate the test script using Microsoft Copilot
2. Execute the test script using Eclipse IDE and Selenium

Note: Please note that all the generative AI tools used in this exercise can produce varied outputs even when presented with similar prompts. Thus, you may get different outputs for the same prompt.

Step 1: Generate the test script using Microsoft Copilot

1.1 Open VS Code and create an HTML file named **Demo.html**



1.2 Add the following code to the **Demo.html** file:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Payment Gateway</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      margin: 0;
      padding: 0;
      background-color: #f7f7f7;
      display: flex;
      justify-content: center;
      align-items: center;
      height: 100vh;
    }

    .container {
      background-color: #ffffff;
      padding: 40px;
      border-radius: 8px;
      box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
      width: 400px;
      max-width: 100%;
      text-align: center;
    }

    h1 {
      margin-bottom: 20px;
      color: #333333;
    }

    label {
      display: block;
      margin-bottom: 5px;
      color: #555555;
      text-align: left;
    }

    input[type="text"],
    input[type="number"],
    select {
```

```

        width: calc(100% - 22px);
        padding: 10px;
        border: 1px solid #cccccc;
        border-radius: 4px;
        box-sizing: border-box;
        margin-bottom: 15px;
    }

    button {
        width: 100%;
        padding: 12px;
        background-color: #007bff;
        color: #ffffff;
        border: none;
        border-radius: 4px;
        cursor: pointer;
        font-size: 16px;
        transition: background-color 0.3s;
    }

    button:hover {
        background-color: #0056b3;
    }

    .error-message {
        color: #ff0000;
        margin-top: 5px;
        text-align: left;
    }
}
</style>
</head>
<body>
    <div class="container">
        <h1>Secure Payment Gateway</h1>
        <form id="paymentForm" onsubmit="return validatePayment()">
            <label for="cardNumber">Card Number:</label>
            <input type="text" id="cardNumber" placeholder="Enter your card number"
required>
            <div class="error-message" id="cardNumberError"></div>

            <label for="cardHolderName">Cardholder Name:</label>
            <input type="text" id="cardHolderName" placeholder="Enter cardholder name"
required>
            <div class="error-message" id="cardHolderNameError"></div>

```

```

<label for="expiryDate">Expiry Date:</label>
<input type="text" id="expiryDate" placeholder="MM/YYYY" required>
<div class="error-message" id="expiryDateError"></div>

<label for="cvv">CVV:</label>
<input type="text" id="cvv" placeholder="Enter CVV" required>
<div class="error-message" id="cvvError"></div>

<label for="amount">Amount:</label>
<input type="number" id="amount" placeholder="Enter amount" required>
<div class="error-message" id="amountError"></div>

<label for="currency">Currency:</label>
<select id="currency" required>
  <option value="USD">USD</option>
  <option value="EUR">EUR</option>
  <option value="GBP">GBP</option>
</select>

<button type="submit">Pay Now</button>
</form>
</div>

<script>
function validatePayment() {
  var cardNumber = document.getElementById('cardNumber').value;
  var cardHolderName = document.getElementById('cardHolderName').value;
  var expiryDate = document.getElementById('expiryDate').value;
  var cvv = document.getElementById('cvv').value;
  var amount = document.getElementById('amount').value;

  var cardNumberError = document.getElementById('cardNumberError');
  var cardHolderNameError = document.getElementById('cardHolderNameError');
  var expiryDateError = document.getElementById('expiryDateError');
  var cvvError = document.getElementById('cvvError');
  var amountError = document.getElementById('amountError');

  // Simple validation for demonstration purposes
  var isValid = true;

  if (!/^\d{16}$/.test(cardNumber)) {
    cardNumberError.textContent = "Please enter a valid card number.";
    isValid = false;
  } else {

```

```

        cardNumberError.textContent = "";
    }

    if (cardHolderName.trim() === "") {
        cardHolderNameError.textContent = "Please enter the cardholder name.";
        isValid = false;
    } else {
        cardHolderNameError.textContent = "";
    }

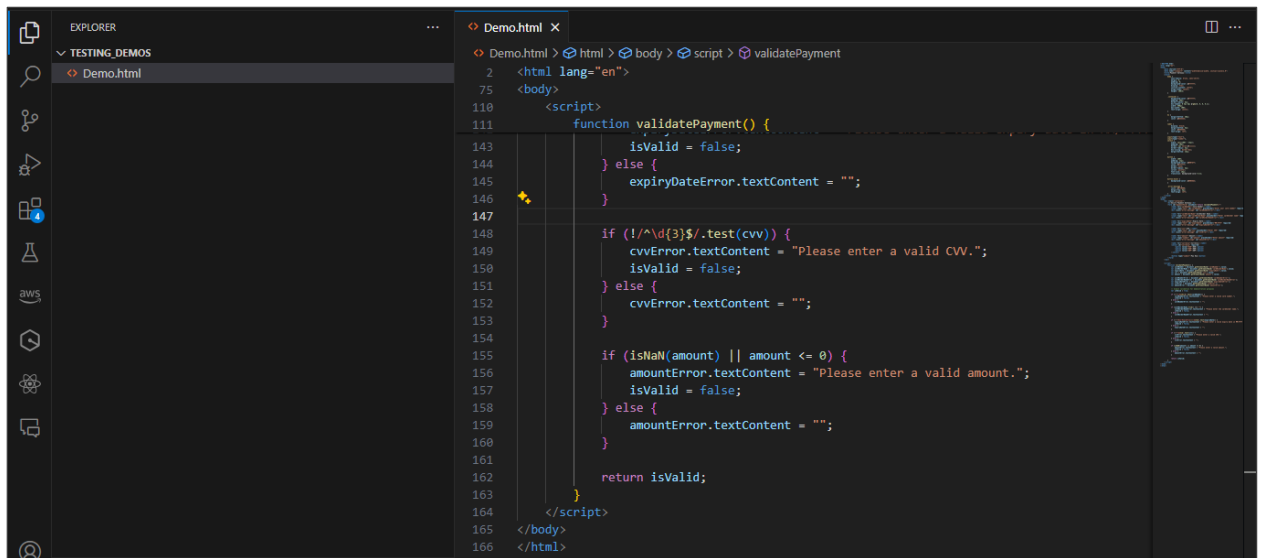
    if (!/^(0[1-9]|1[0-2])\d{4}$/.test(expiryDate)) {
        expiryDateError.textContent = "Please enter a valid expiry date in MM/YYYY
format.";
        isValid = false;
    } else {
        expiryDateError.textContent = "";
    }

    if (!/^\d{3}$/.test(cvv)) {
        cvvError.textContent = "Please enter a valid CVV.";
        isValid = false;
    } else {
        cvvError.textContent = "";
    }

    if (isNaN(amount) || amount <= 0) {
        amountError.textContent = "Please enter a valid amount.";
        isValid = false;
    } else {
        amountError.textContent = "";
    }

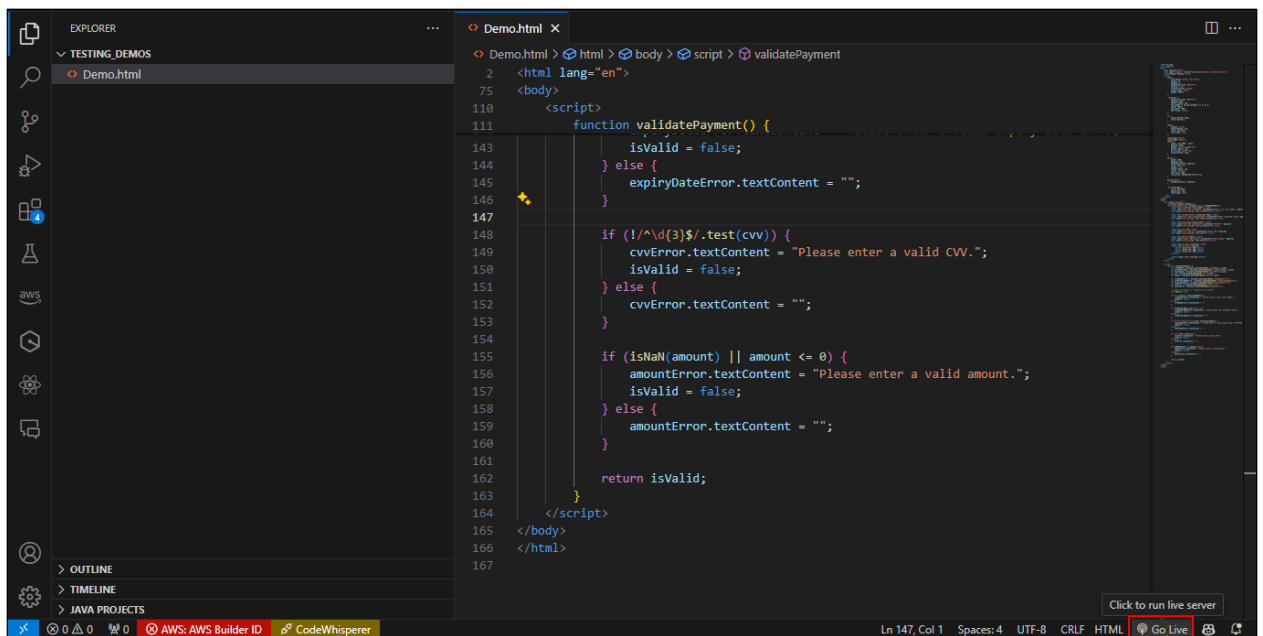
    return isValid;
}
</script>
</body>
</html>

```

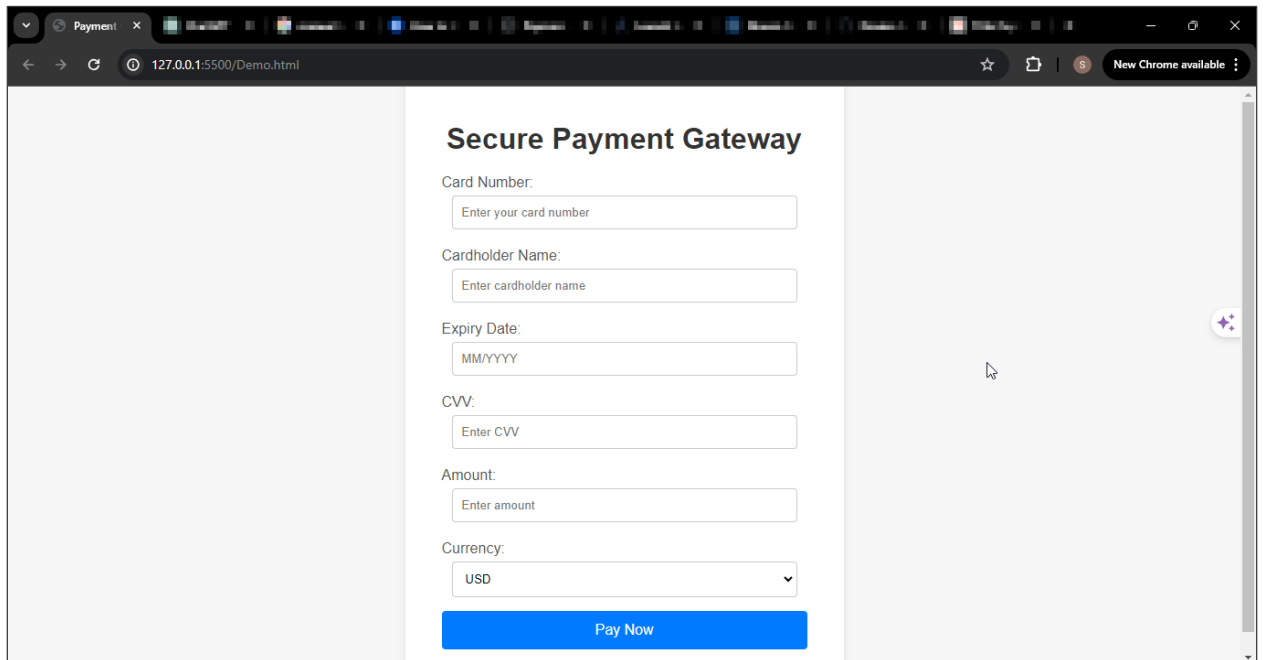


```
2 <html lang="en">
75 <body>
110 <script>
111     function validatePayment() {
143         isValid = false;
144     } else {
145         expiryDateError.textContent = "";
146     }
147
148     if (/^\d{3}$/.test(cvv)) {
149         cvvError.textContent = "Please enter a valid CVV.";
150         isValid = false;
151     } else {
152         cvvError.textContent = "";
153     }
154
155     if (isNaN(amount) || amount <= 0) {
156         amountError.textContent = "Please enter a valid amount.";
157         isValid = false;
158     } else {
159         amountError.textContent = "";
160     }
161
162     return isValid;
163 }
164 </script>
165 </body>
166 </html>
```

1.3 Click on the **Go Live** button to start the live server as shown in the screenshot below:



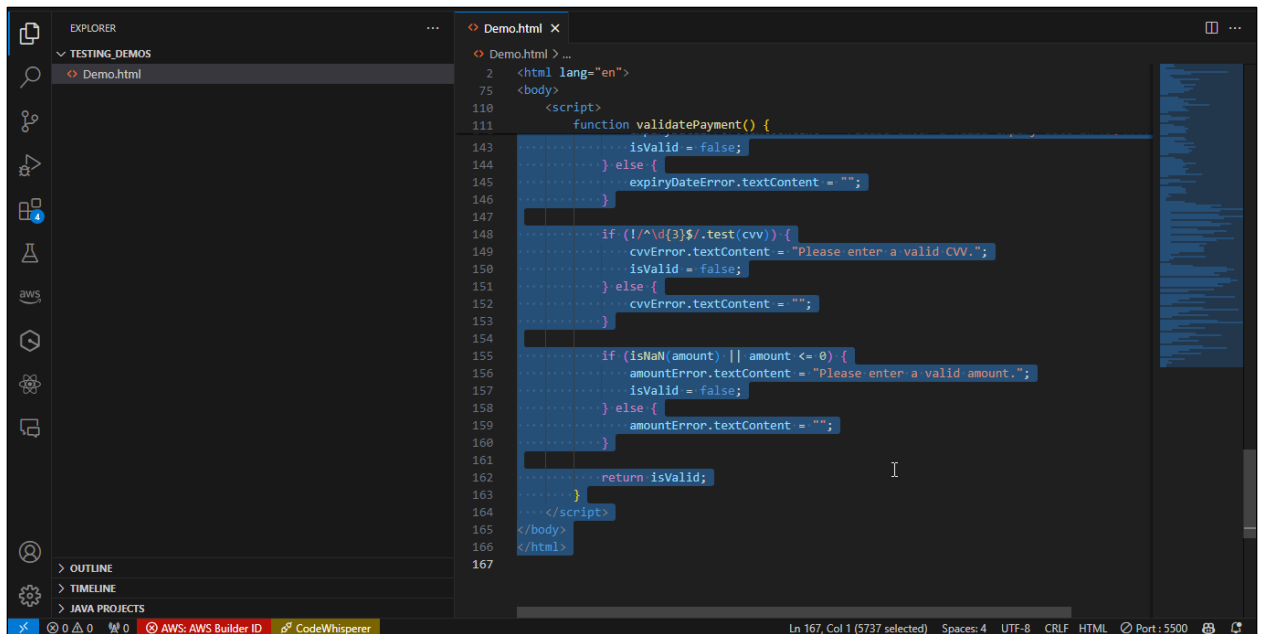
The following page will appear:



The screenshot shows a web browser window with the address bar displaying "127.0.0.1:5500/Demo.html". The page title is "Secure Payment Gateway". The form contains the following fields and controls:

- Card Number:** A text input field with the placeholder text "Enter your card number".
- Cardholder Name:** A text input field with the placeholder text "Enter cardholder name".
- Expiry Date:** A text input field with the placeholder text "MM/YYYY".
- CVV:** A text input field with the placeholder text "Enter CVV".
- Amount:** A text input field with the placeholder text "Enter amount".
- Currency:** A dropdown menu currently showing "USD".
- Pay Now:** A blue button.

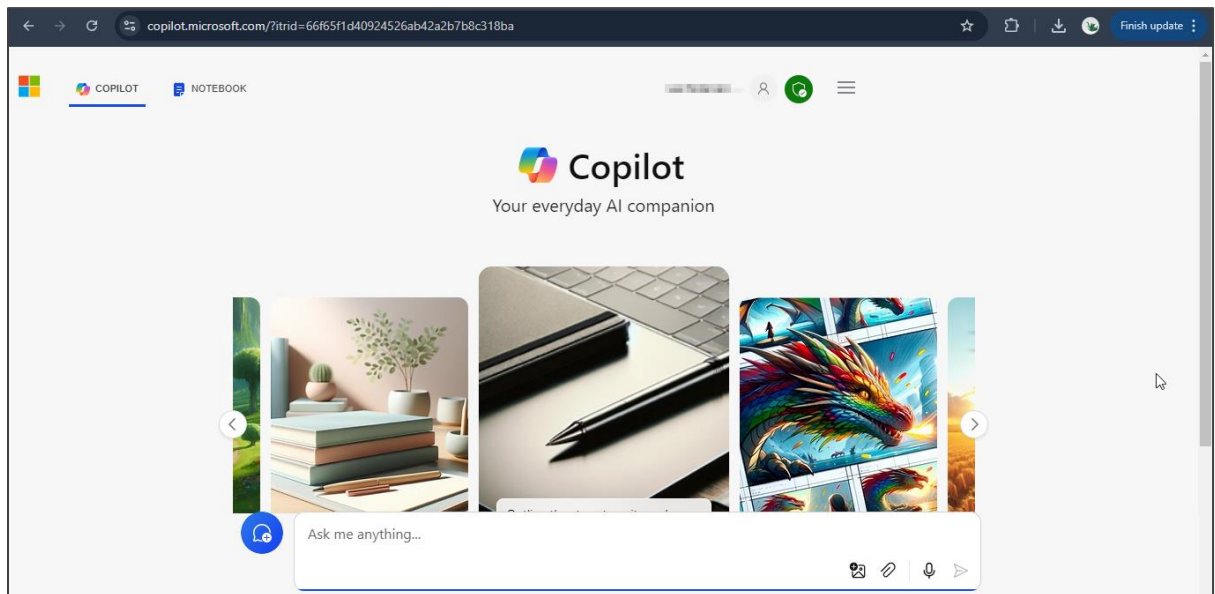
1.4 Navigate to VS Code, select the entire code, and copy the code as shown in the screenshot below:



The screenshot shows the VS Code editor with the file "Demo.html" open. The code is as follows:

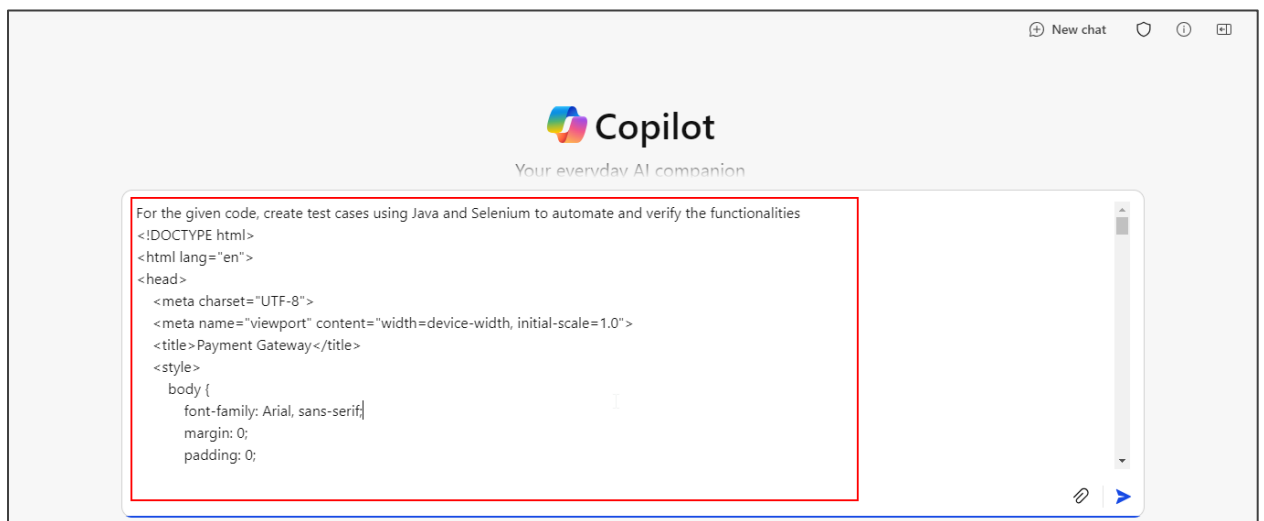
```
2 <html lang="en">
75 <body>
110 <script>
111     function validatePayment() {
143         isValid = false;
144     } else {
145         expiryDateError.textContent = "";
146     }
147
148     if (/^\d{3}$/.test(cvv)) {
149         cvvError.textContent = "Please enter a valid CVV.";
150         isValid = false;
151     } else {
152         cvvError.textContent = "";
153     }
154
155     if (isNaN(amount) || amount <= 0) {
156         amountError.textContent = "Please enter a valid amount.";
157         isValid = false;
158     } else {
159         amountError.textContent = "";
160     }
161
162     return isValid;
163 }
164 </script>
165 </body>
166 </html>
167
```

1.5 Open Microsoft Copilot in Google Chrome

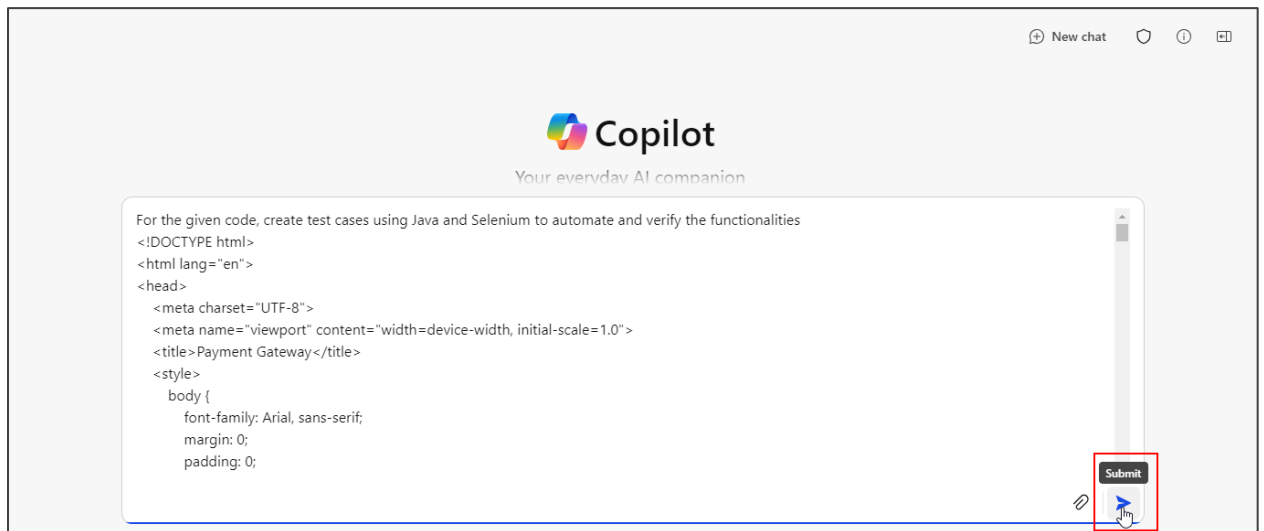


1.6 Add the following prompt in the **Ask me anything...** area and paste the code copied in step 1.4:

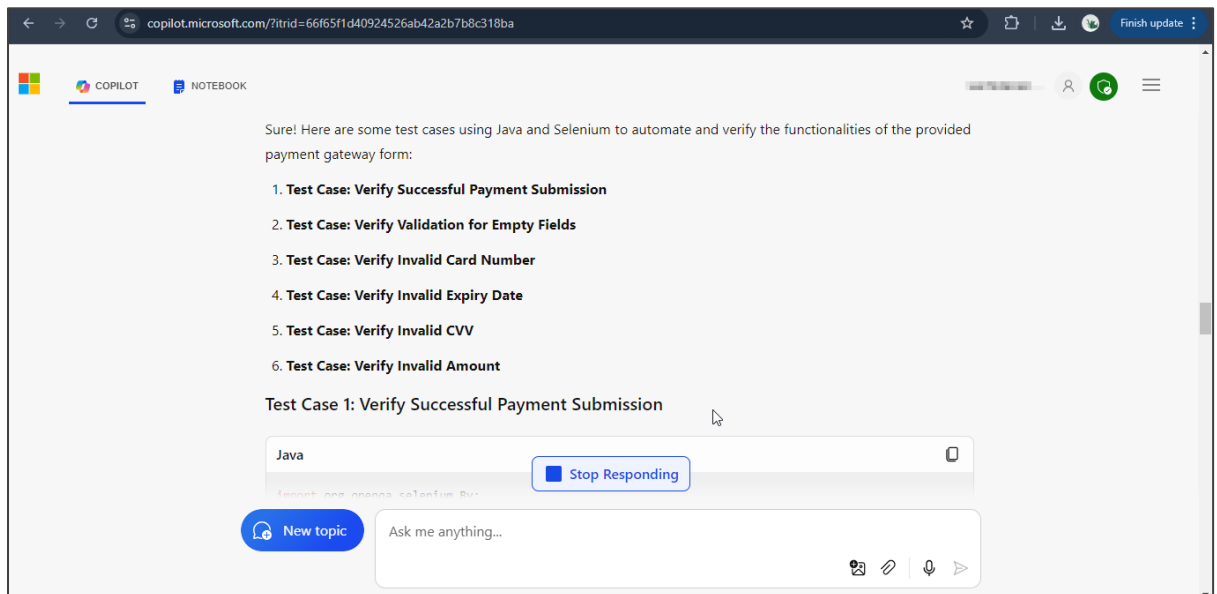
For the given code, create test cases using Java and Selenium to automate and verify the functionalities



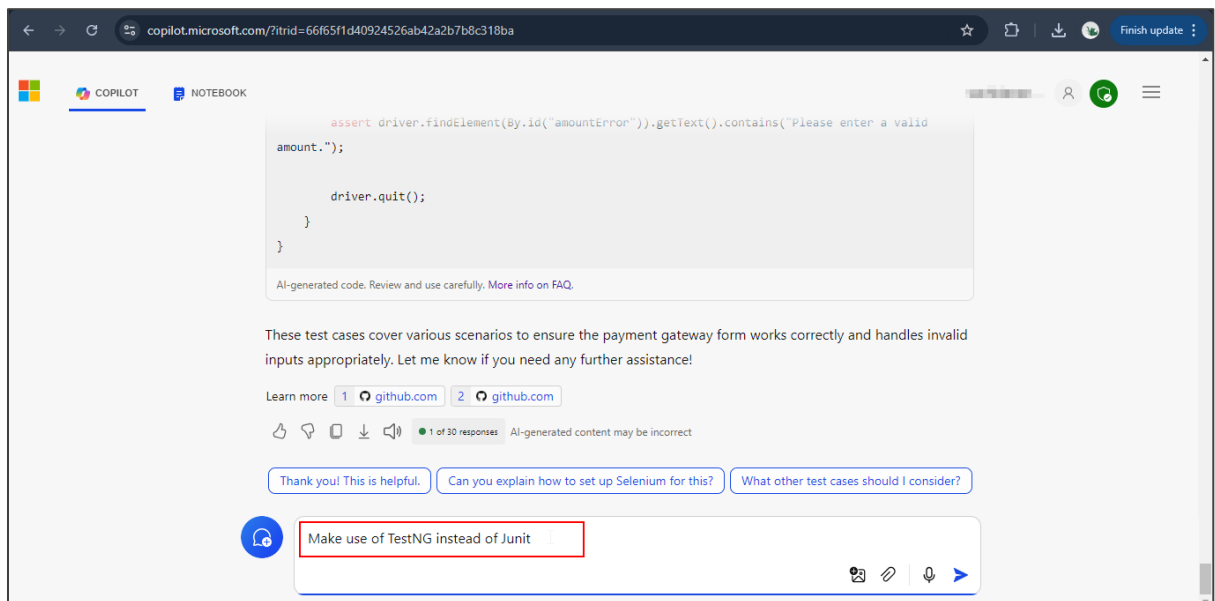
1.7 Click on the **Submit** button to generate the test cases



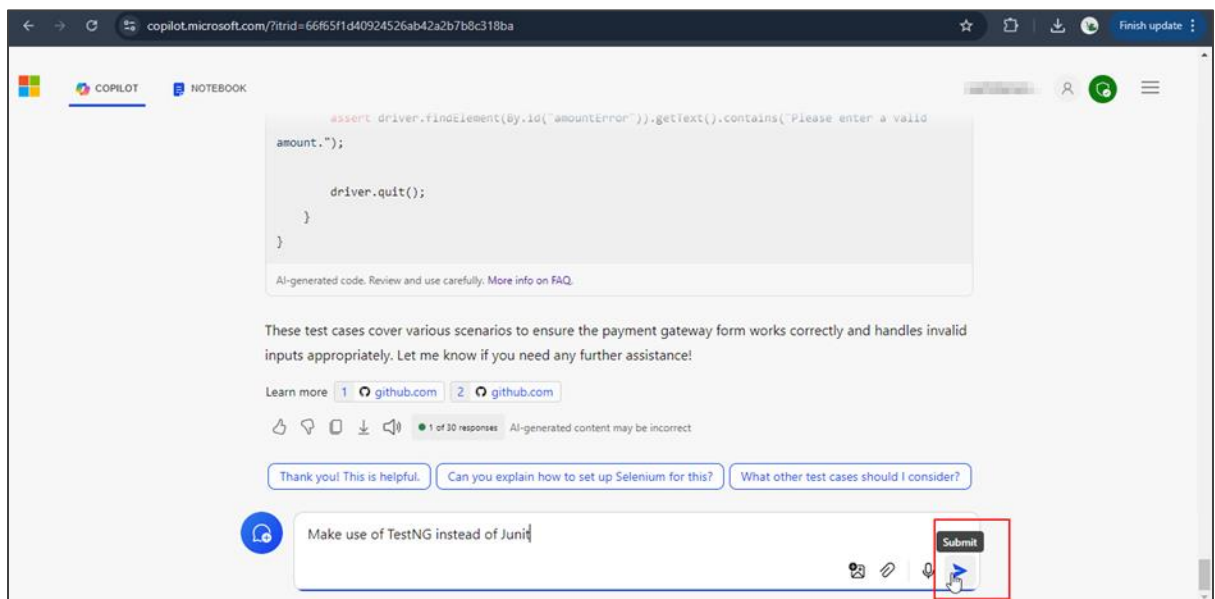
Microsoft Copilot starts generating the test cases as shown below:



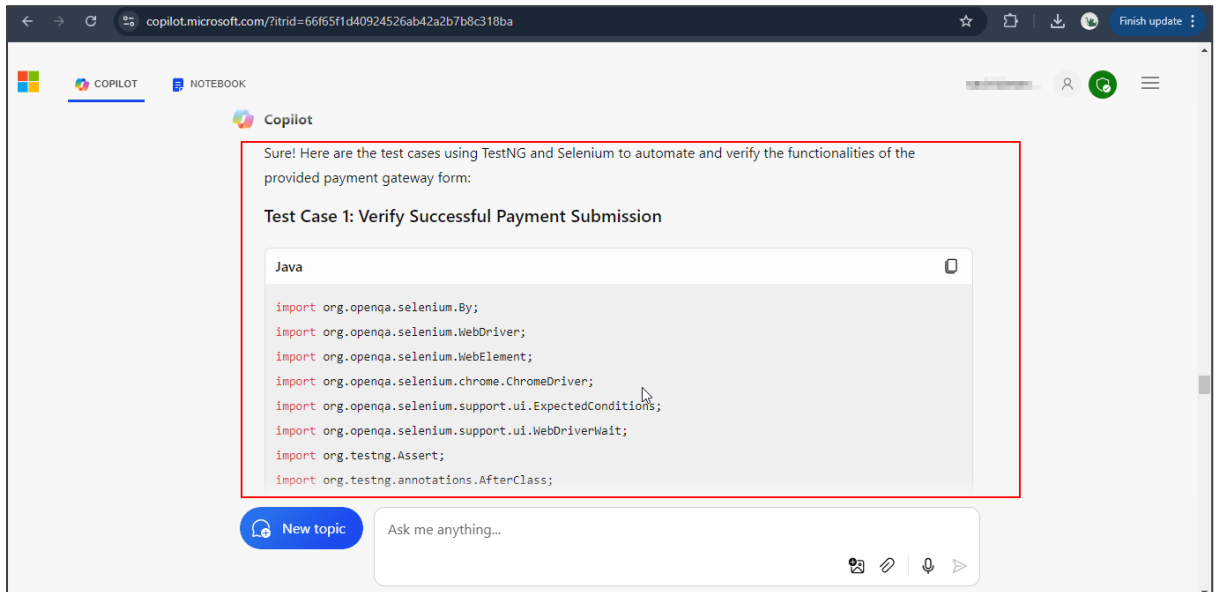
1.8 Add the following prompt to make the changes in the test script:
Make use of TestNG instead of Junit



1.9 Click on the **Submit** button to generate the test cases



Microsoft Copilot starts generating the test cases as shown below:



← → ↻ copilot.microsoft.com/?itrid=66f65f1d40924526ab42a2b7b8c318ba ☆ 📁 ⬇️ 🗨️ Finish update ⋮

COPILOT NOTEBOOK

Test Case 3: Verify Invalid Card Number

Java

```
public class PaymentGatewayTest {
    WebDriver driver;

    @BeforeClass
    public void setUp() {
        System.setProperty("webdriver.chrome.driver", "path/to/chromedriver");
        driver = new ChromeDriver();
        driver.get("path/to/your/payment-gateway.html");
    }

    @Test
    public void testInvalidCardNumber() {
```

New topic Ask me anything... 🗨️ 📎 🗣️ ➡️

← → ↻ copilot.microsoft.com/?itrid=66f65f1d40924526ab42a2b7b8c318ba ☆ 📁 ⬇️ 🗨️ Finish update ⋮

COPILOT NOTEBOOK

Test Case 4: Verify Invalid Expiry Date

Java

```
public class PaymentGatewayTest {
    WebDriver driver;

    @BeforeClass
    public void setUp() {
        System.setProperty("webdriver.chrome.driver", "path/to/chromedriver");
        driver = new ChromeDriver();
        driver.get("path/to/your/payment-gateway.html");
    }

    @Test
    public void testInvalidExpiryDate() {
```

New topic Ask me anything... 🗨️ 📎 🗣️ ➡️

copilot.microsoft.com/?itrid=66f65f1d40924526ab42a2b7b8c318ba

COPILOT NOTEBOOK

Test Case 5: Verify Invalid CVV

Java

```
public class PaymentGatewayTest {
    WebDriver driver;

    @BeforeClass
    public void setUp() {
        System.setProperty("webdriver.chrome.driver", "path/to/chromedriver");
        driver = new ChromeDriver();
        driver.get("path/to/your/payment-gateway.html");
    }

    @Test
    public void testInvalidCVV() {
```

New topic Ask me anything...

copilot.microsoft.com/?itrid=66f65f1d40924526ab42a2b7b8c318ba

COPILOT NOTEBOOK

Test Case 6: Verify Invalid Amount

Java

```
public class PaymentGatewayTest {
    WebDriver driver;

    @BeforeClass
    public void setUp() {
        System.setProperty("webdriver.chrome.driver", "path/to/chromedriver");
        driver = new ChromeDriver();
        driver.get("path/to/your/payment-gateway.html");
    }

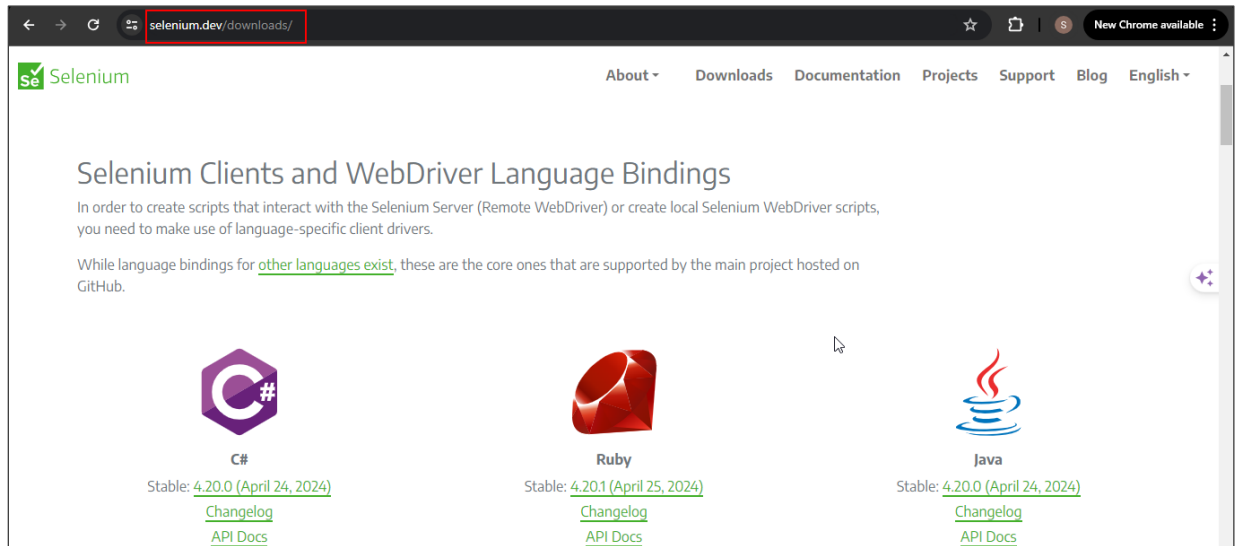
    @Test
    public void testInvalidAmount() {
```

New topic Ask me anything...

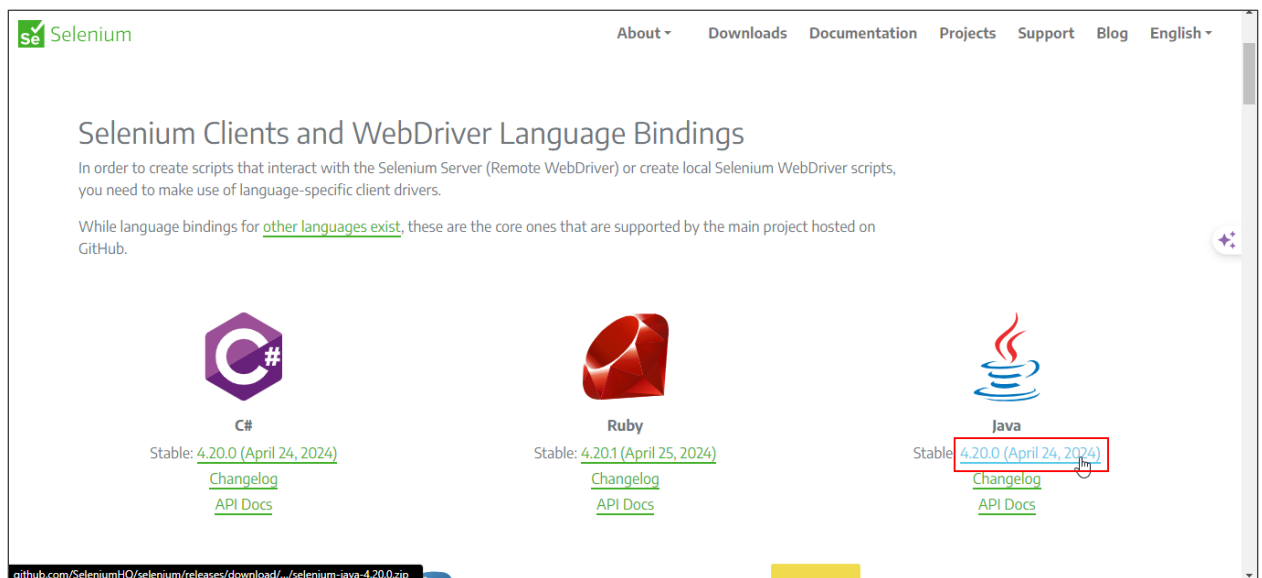
Note: Copy all the test scripts generated by **Microsoft Copilot** and save them in Notepad

Step 2: Execute the test script using Eclipse IDE and Selenium

2.1 Navigate to <https://www.selenium.dev/downloads/>

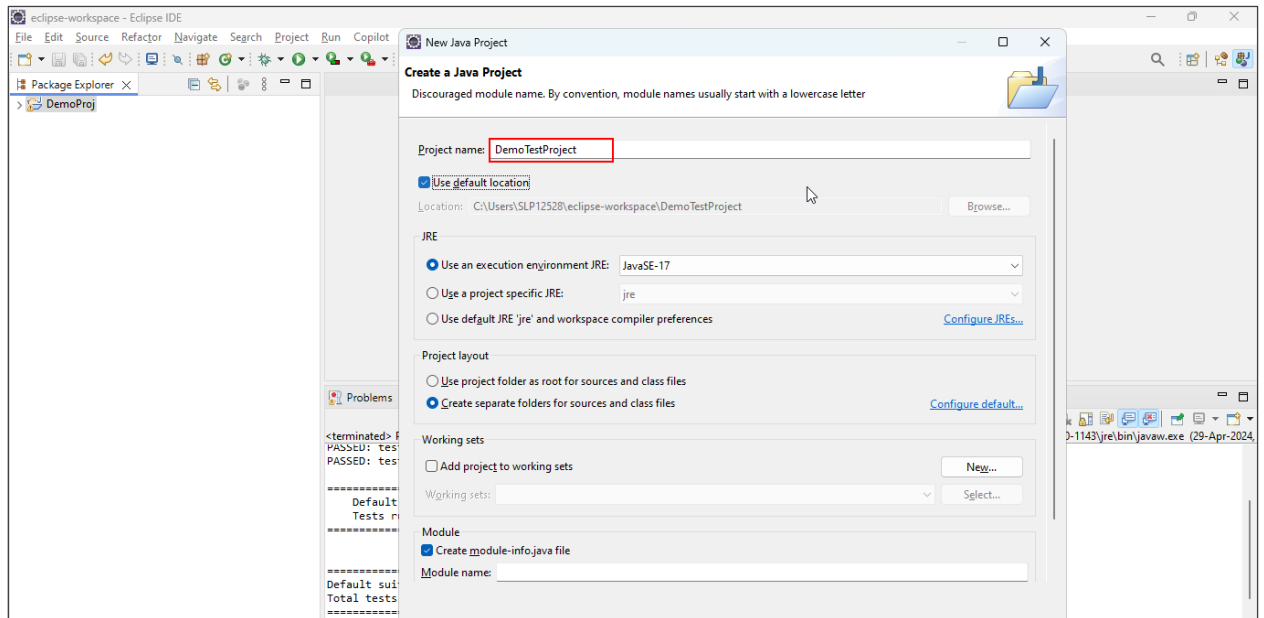


2.2 Click on the **4.20.0** link of the Java client driver as shown in the screenshot below:



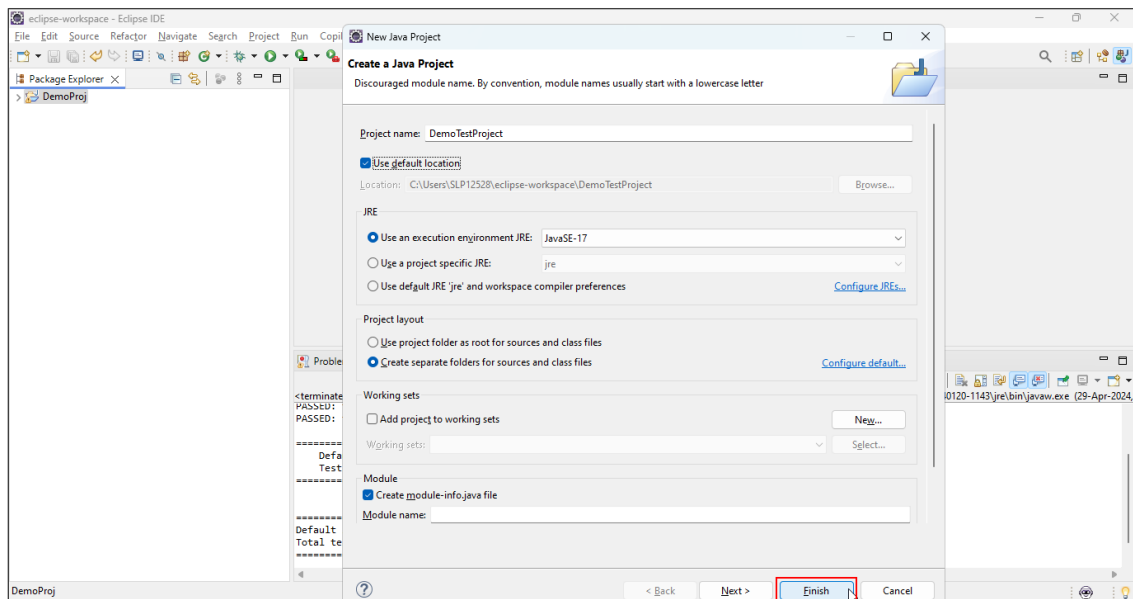
Note: Once the download is complete, extract the files and keep them in a folder to import later in the Java project

2.3 Open the Eclipse IDE and create a new Java project as shown in the screenshot below:

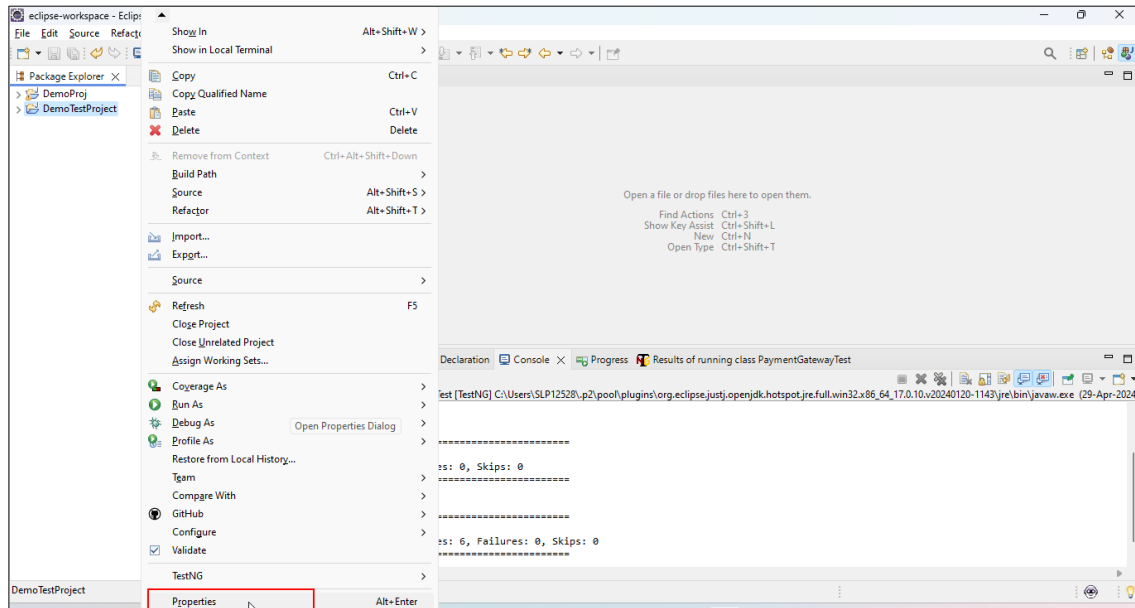


Note: Download and install Eclipse IDE from the official website link (<https://www.eclipse.org/downloads/>) if you do not have it installed

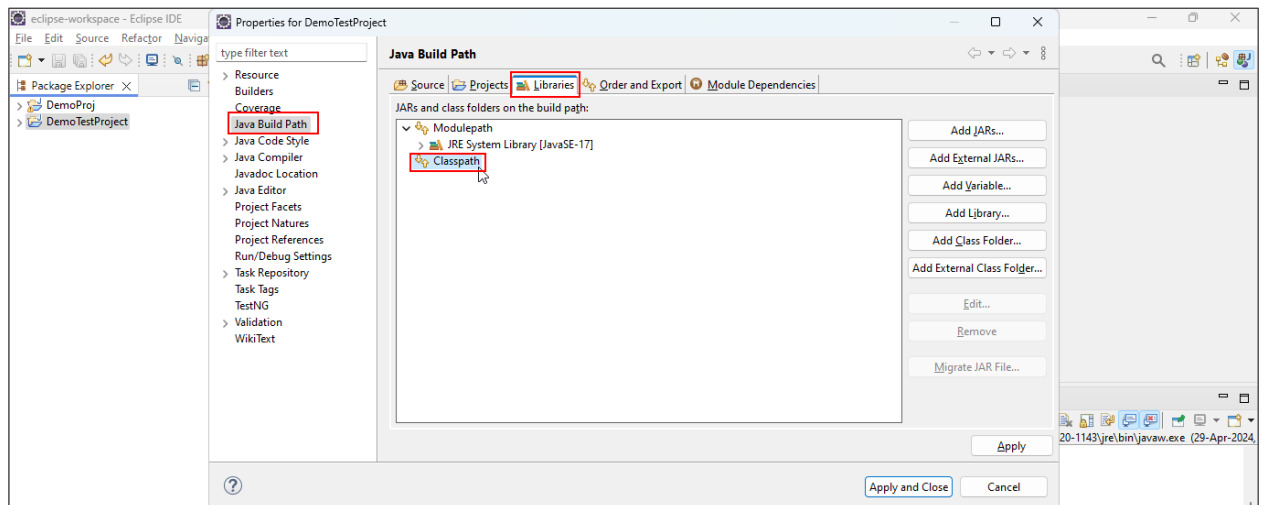
2.4 Click on the **Finish** button as shown in the screenshot below:



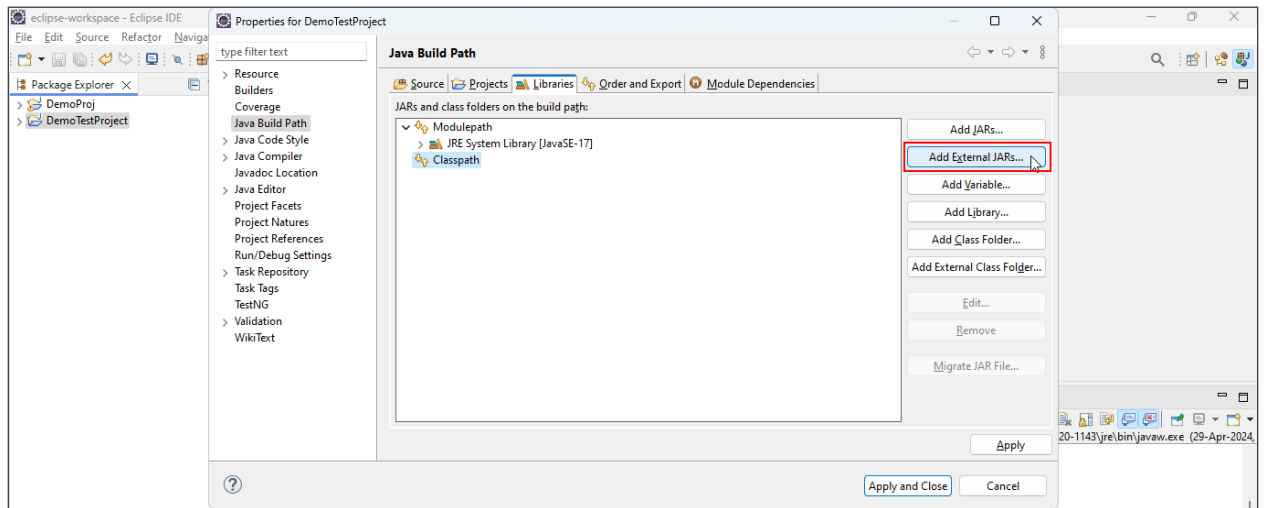
2.5 Right-click on the project name and select the **properties** option as shown in the screenshot below:



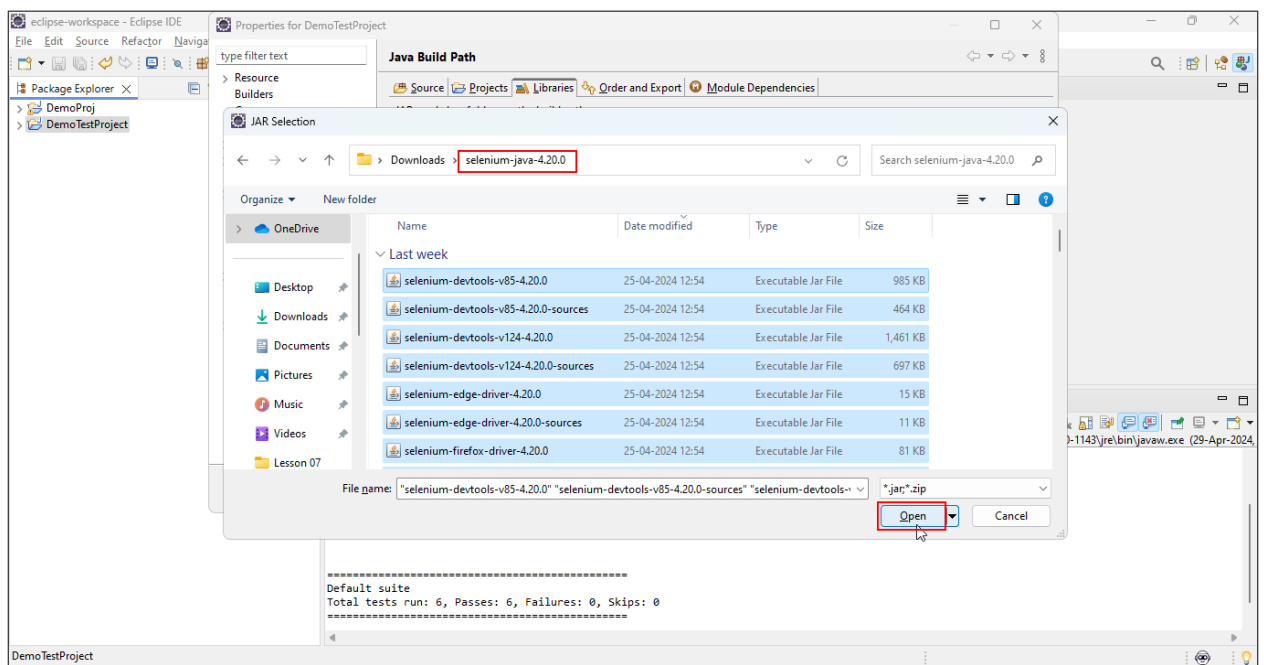
2.6 Click on **Java Build Path**, then click on **Libraries**, and select **Classpath** from the drop-down as shown in the screenshot below:



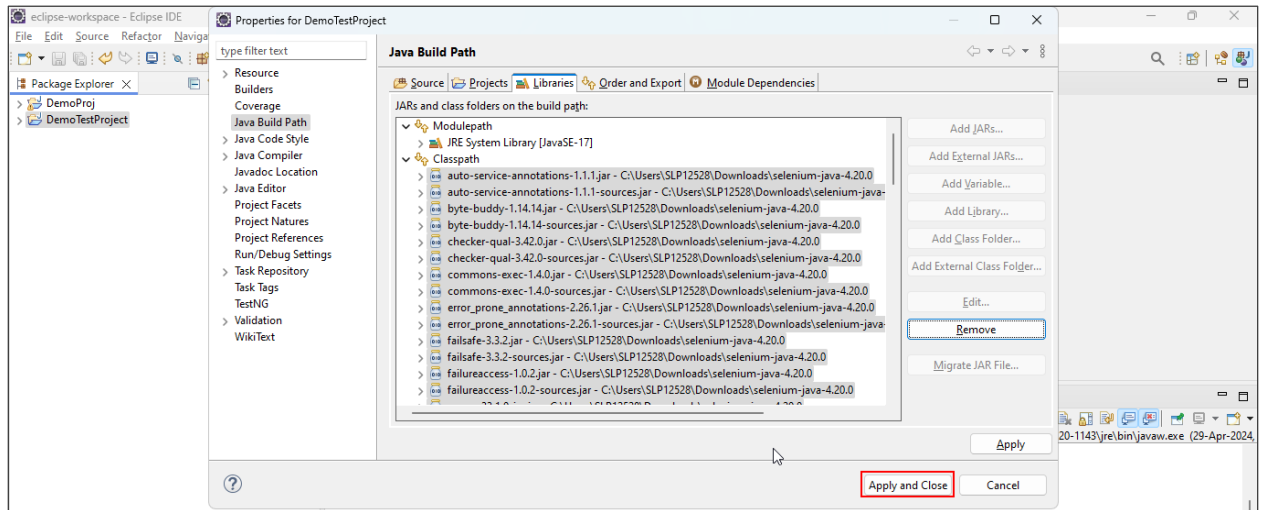
2.7 Click on the **Add External JARs...** button as shown in the screenshot below:



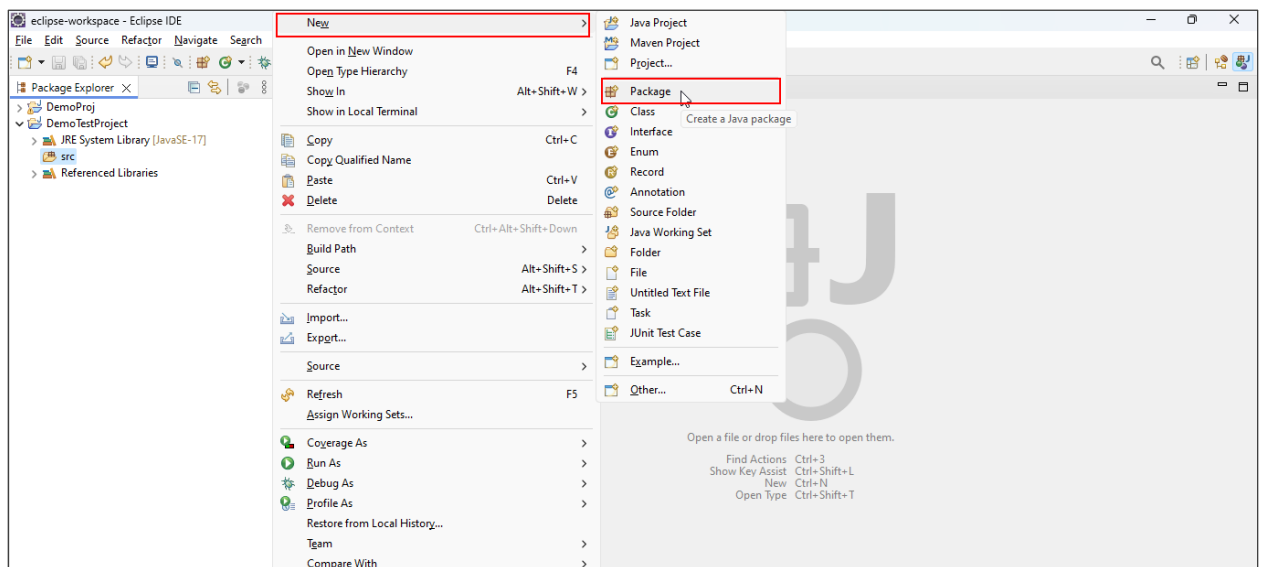
2.8 Select all the JAR files from the extracted **selenium-java-4.20.0** folder and click on the **Open** button as shown in the screenshot below:



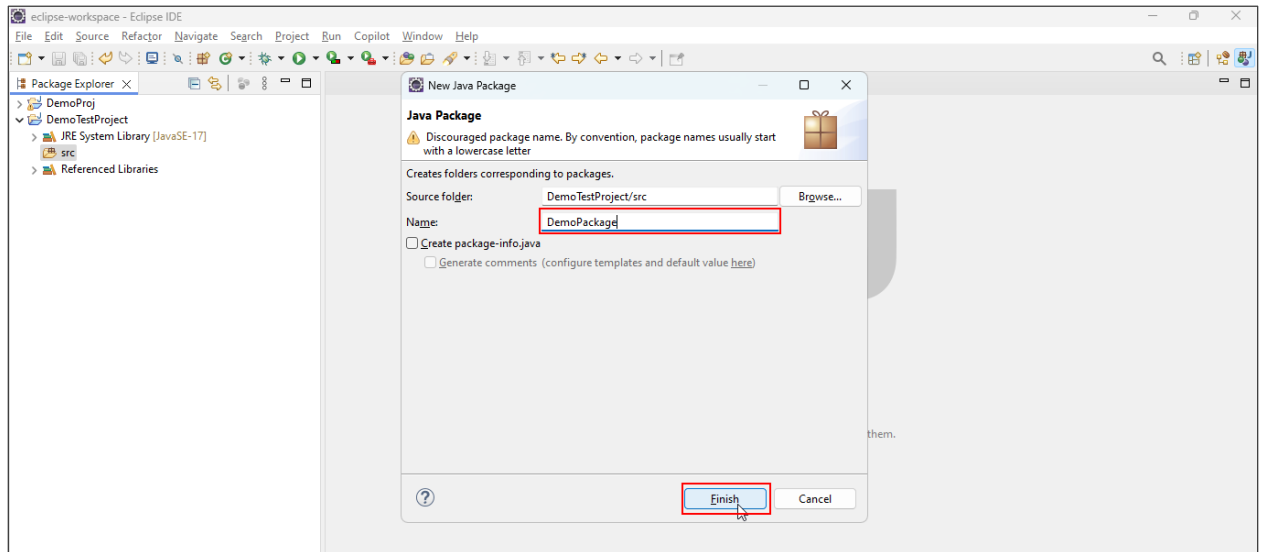
2.9 Click on the **Apply and Close** button once the JAR files are added to the **Classpath** as shown in the screenshot below:



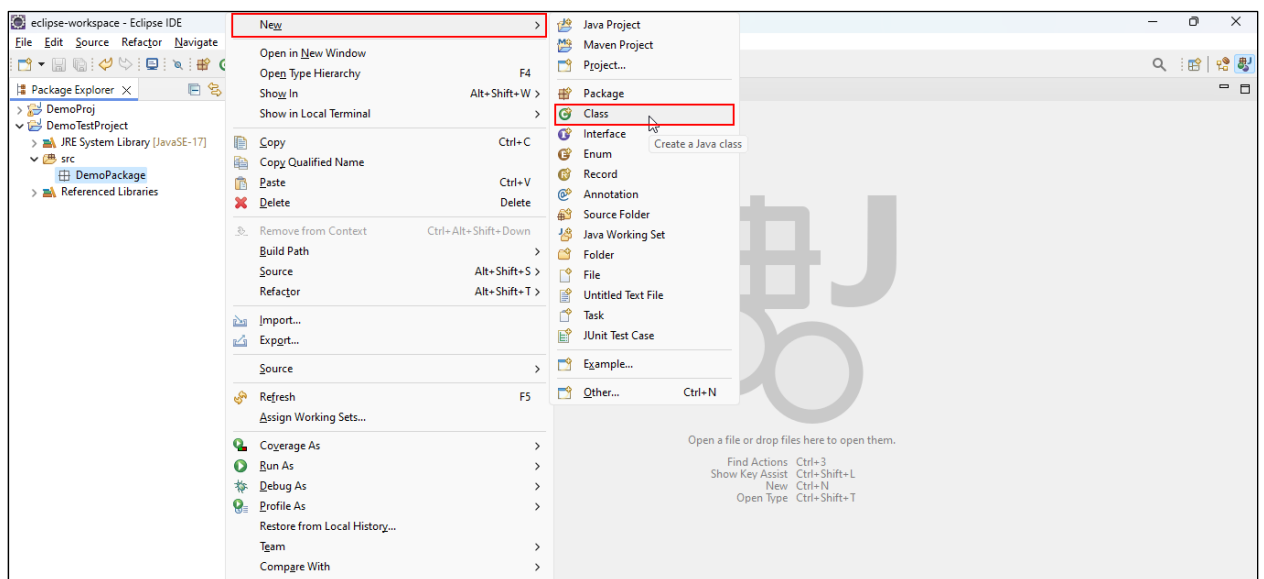
2.10 Right-click on the src folder, select **New**, and then select **Package** to create a new package as shown in the screenshot below:



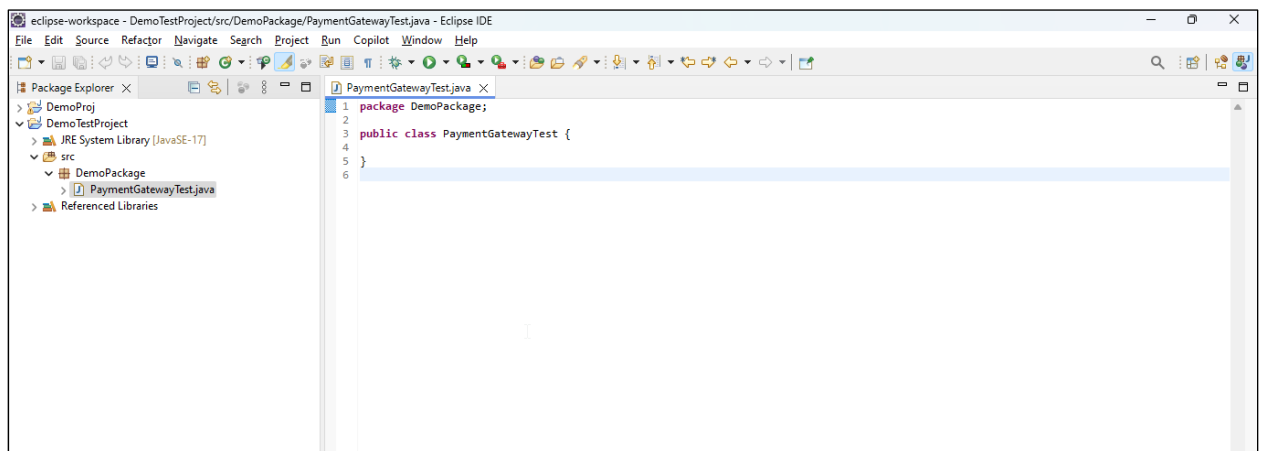
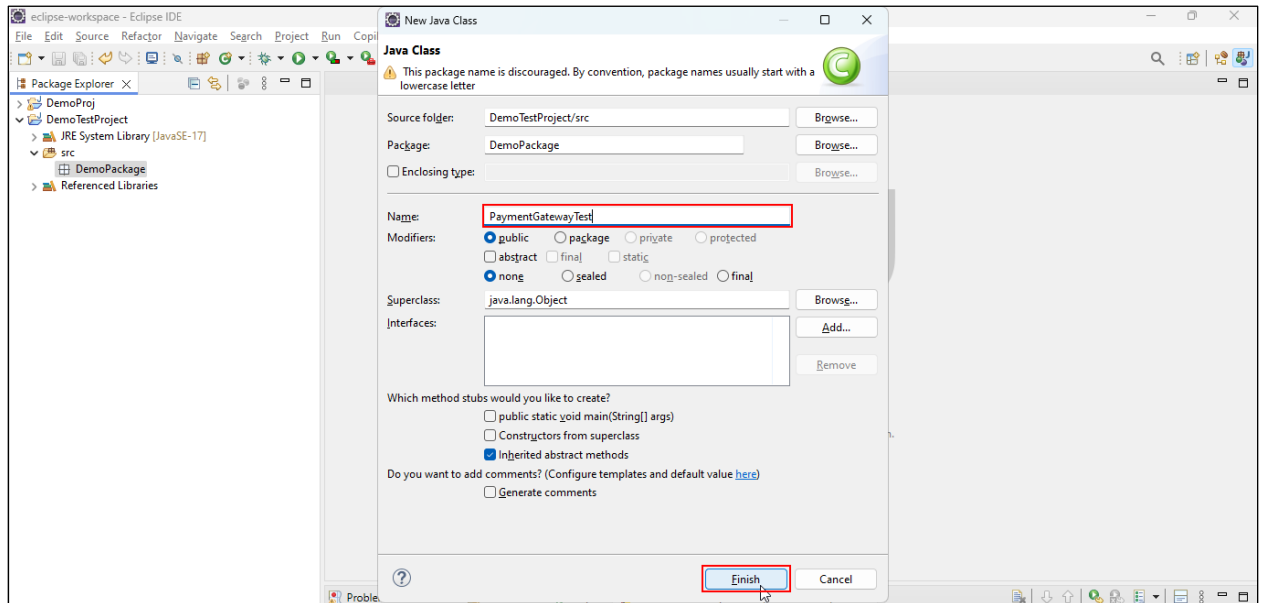
2.11 Enter the name of the package and click on the **Finish** button as shown in the screenshot below:



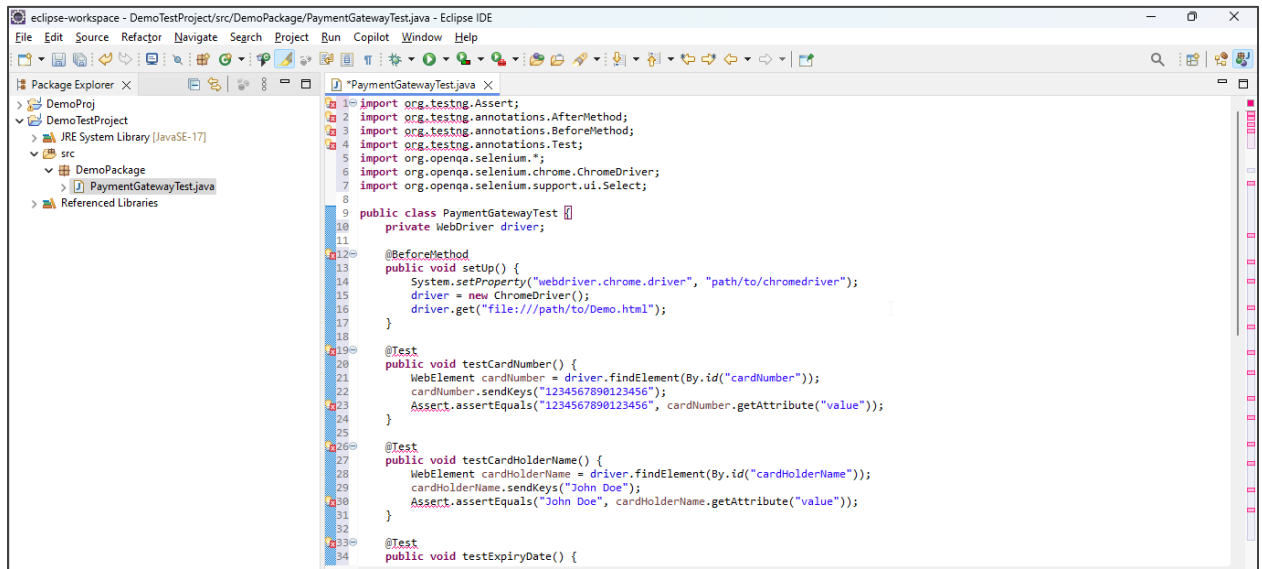
2.12 Right-click on the package name, scroll to **New**, and select **Class** to create a new Java class as shown in the screenshot below:



2.13 Enter the same class name that you got in step 1.7 and click on the **Finish** button as shown in the screenshot below:

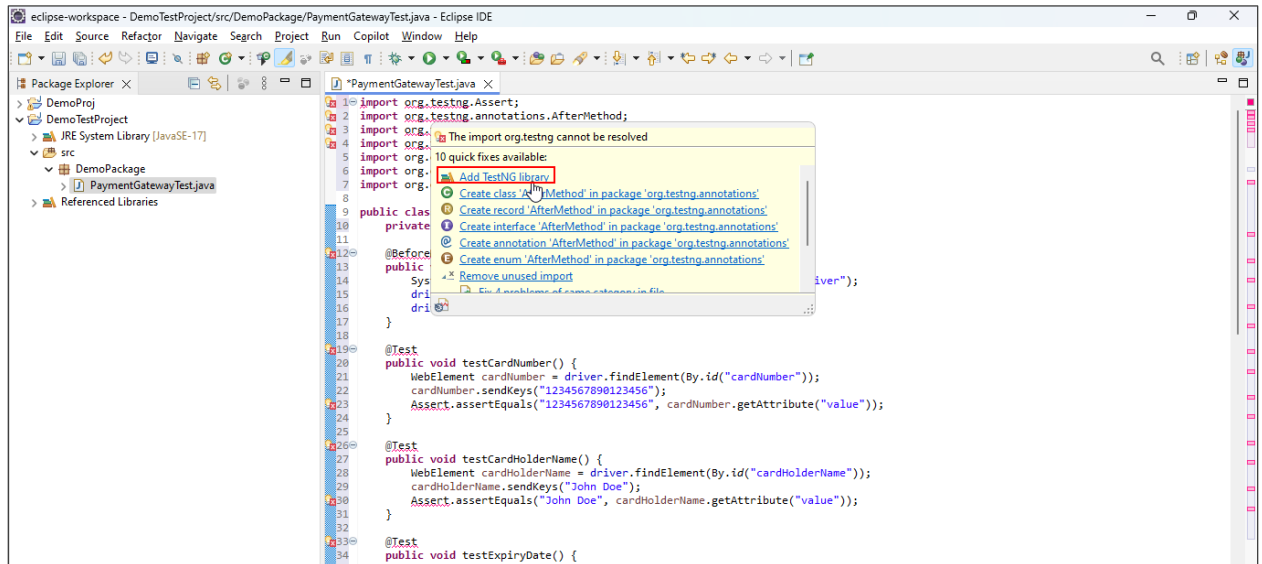


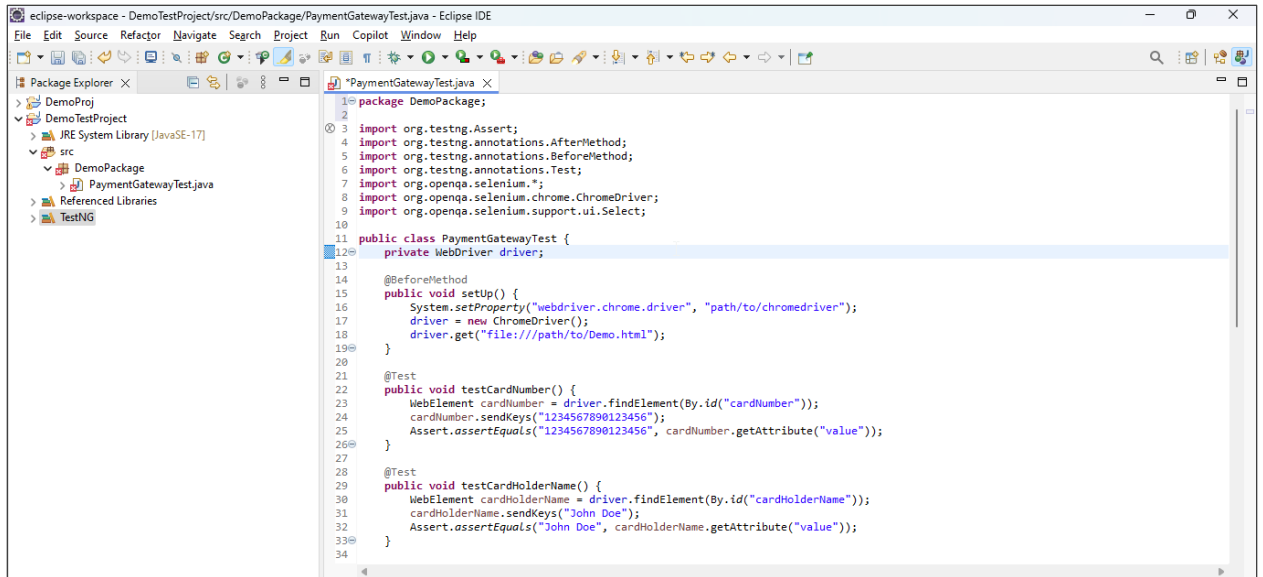
2.14 Replace the existing code with the code that you saved earlier in the Notepad



Note: Do not replace the package name, else it might cause an error

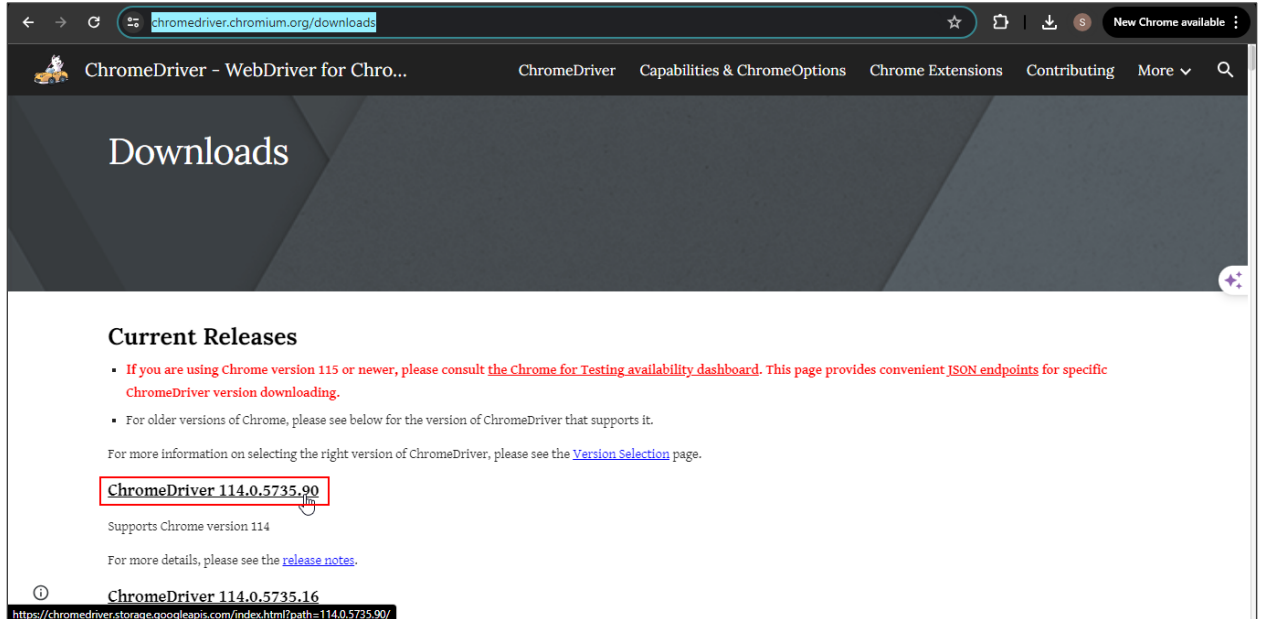
2.15 Scroll to the line indicating the error and add the TestNG library to resolve the errors as shown in the screenshots below:





Note: Navigate to the **Help** section, click on **Eclipse marketplace...**, and search for and install **TestNG** if you do not have it installed

2.16 Navigate to <https://chromedriver.chromium.org/downloads> and download the latest version of Chrome driver that supports your Google Chrome and operating system



chrome://chromedriver.storage.googleapis.com/index.html?path=/114.0.5735.90/

Index of /114.0.5735.90/

Name	Last modified	Size	ETag
Parent Directory		-	
chromedriver_linux64.zip	2023-05-31 08:57:22	7.06MB	cd6613edf6628041684393706b62d3a6
chromedriver_mac64.zip	2023-05-31 08:57:25	8.29MB	b44390afbdddad8748a1d151483b2472
chromedriver_mac_arm64.zip	2023-05-31 08:57:29	7.40MB	0d515e46bea141705e49edaba1d49819
chromedriver_win32.zip	2023-05-31 08:57:32	6.30MB	7d455bed57ef682d41108e13d45545ca
notes.txt	2023-05-31 08:57:38	0.00MB	1670f6dde7877ca84ecd4c56b9cc759c

Note: Extract the zip folder on your system

2.17 Remove the **System.setProperty** line from the code as shown in the screenshot below:

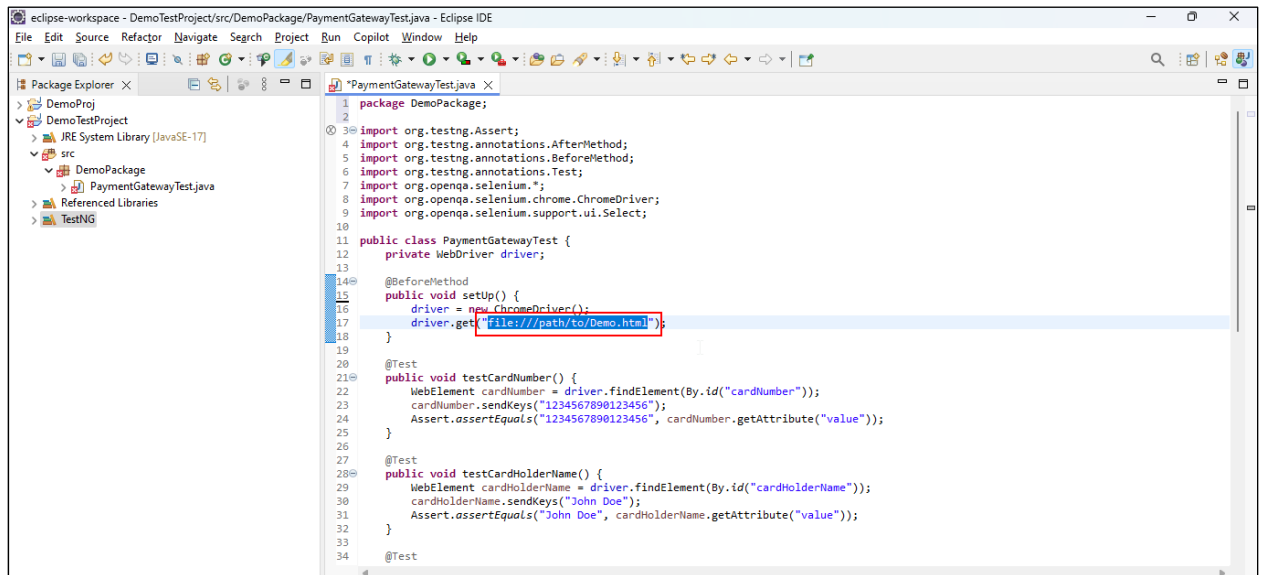
eclipse-workspace - DemoTestProject/src/DemoPackage/PaymentGateway/Test.java - Eclipse IDE

```

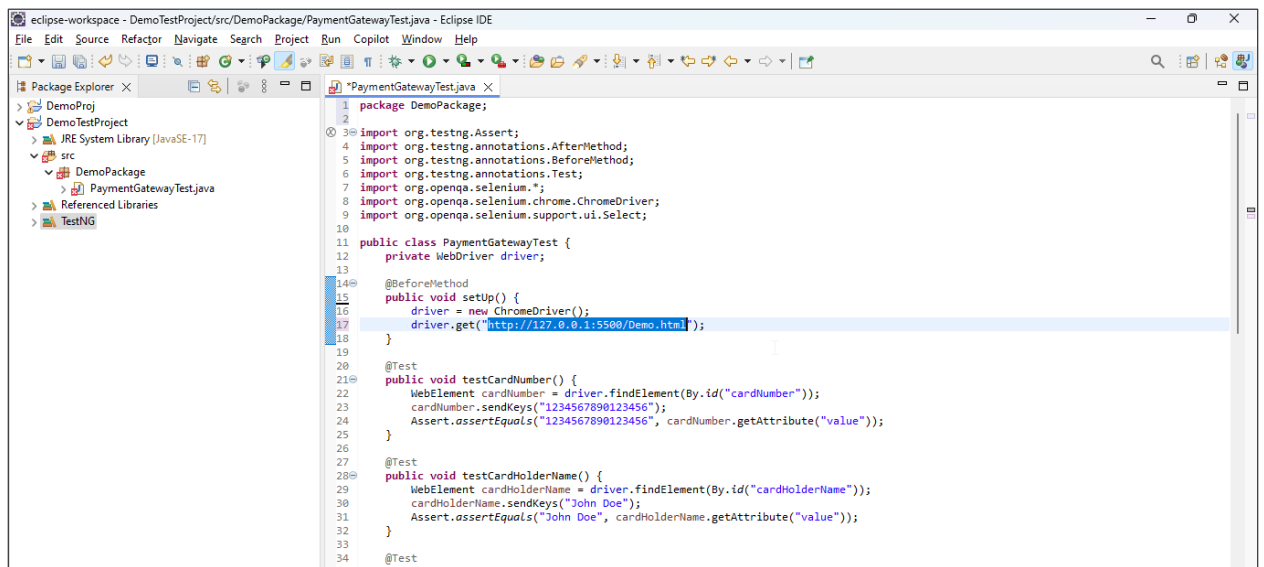
1 package DemoPackage;
2
3 import org.testng.Assert;
4 import org.testng.annotations.AfterMethod;
5 import org.testng.annotations.BeforeMethod;
6 import org.testng.annotations.Test;
7 import org.openqa.selenium.*;
8 import org.openqa.selenium.chrome.ChromeDriver;
9 import org.openqa.selenium.support.ui.Select;
10
11 public class PaymentGatewayTest {
12     private WebDriver driver;
13
14     @BeforeMethod
15     public void setUp() {
16         System.setProperty("webdriver.chrome.driver", "path/to/chromedriver");
17         driver = new ChromeDriver();
18         driver.get("file:///path/to/Demo.html");
19     }
20
21     @Test
22     public void testCardNumber() {
23         WebElement cardNumber = driver.findElement(By.id("cardNumber"));
24         cardNumber.sendKeys("1234567890123456");
25         Assert.assertEquals("1234567890123456", cardNumber.getAttribute("value"));
26     }
27
28     @Test
29     public void testCardHolderName() {
30         WebElement cardHolderName = driver.findElement(By.id("cardHolderName"));
31         cardHolderName.sendKeys("John Doe");
32         Assert.assertEquals("John Doe", cardHolderName.getAttribute("value"));
33     }
34 }

```

2.18 Replace the URL path with the URL of the live server as shown in the screenshot below:



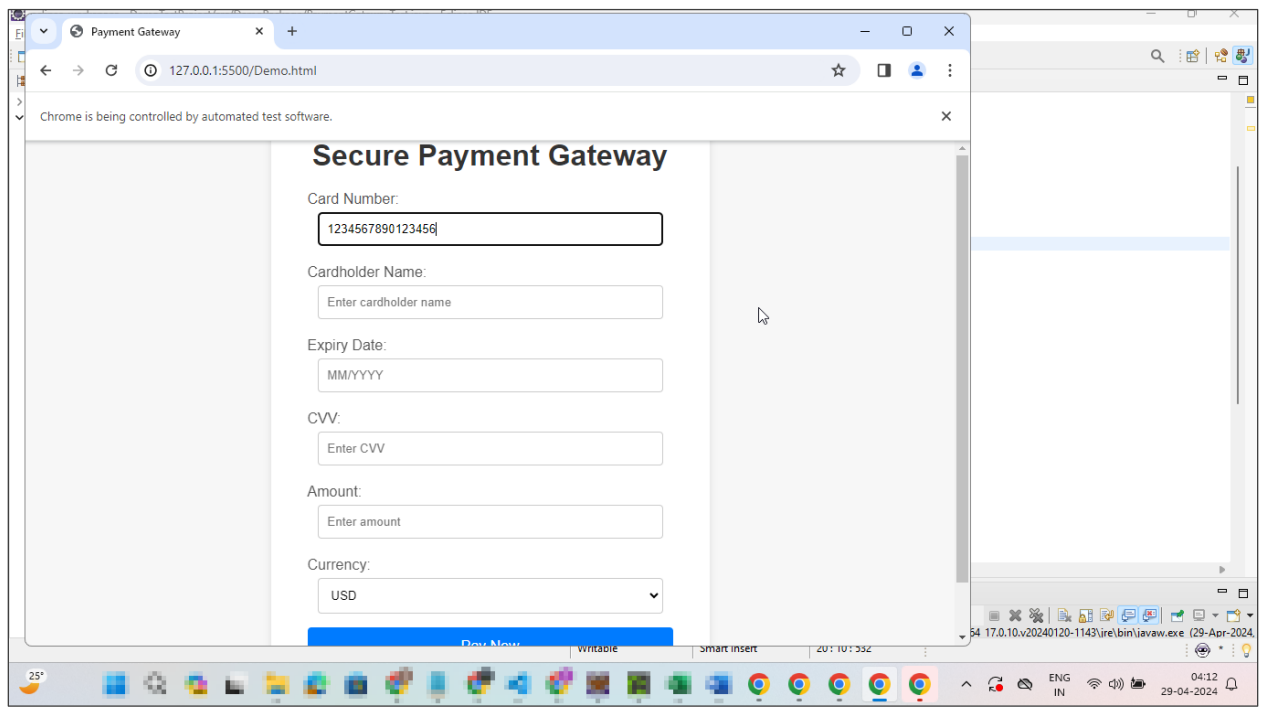
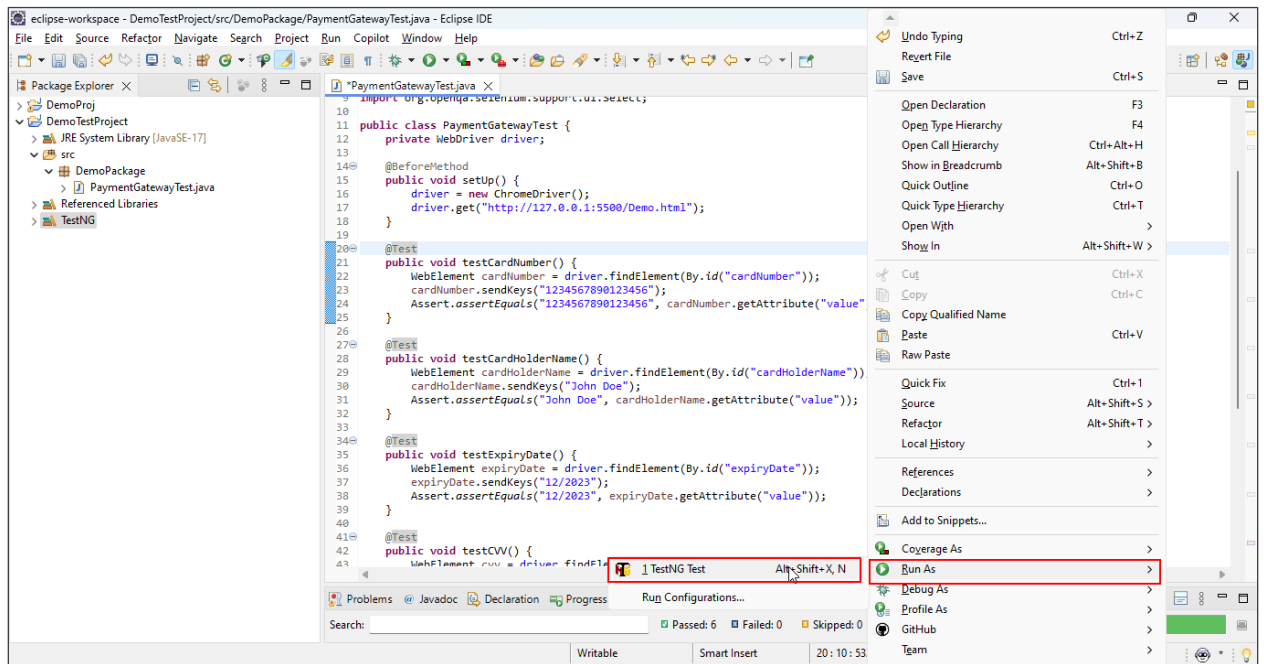
```
1 package DemoPackage;
2
3 import org.testng.Assert;
4 import org.testng.annotations.AfterMethod;
5 import org.testng.annotations.BeforeMethod;
6 import org.testng.annotations.Test;
7 import org.openqa.selenium.*;
8 import org.openqa.selenium.chrome.ChromeDriver;
9 import org.openqa.selenium.support.ui.Select;
10
11 public class PaymentGatewayTest {
12     private WebDriver driver;
13
14     @BeforeMethod
15     public void setUp() {
16         driver = new ChromeDriver();
17         driver.get("file:///path/to/Demo.html");
18     }
19
20     @Test
21     public void testCardNumber() {
22         WebElement cardNumber = driver.findElement(By.id("cardNumber"));
23         cardNumber.sendKeys("1234567890123456");
24         Assert.assertEquals("1234567890123456", cardNumber.getAttribute("value"));
25     }
26
27     @Test
28     public void testCardHolderName() {
29         WebElement cardHolderName = driver.findElement(By.id("cardHolderName"));
30         cardHolderName.sendKeys("John Doe");
31         Assert.assertEquals("John Doe", cardHolderName.getAttribute("value"));
32     }
33
34     @Test
```



```
1 package DemoPackage;
2
3 import org.testng.Assert;
4 import org.testng.annotations.AfterMethod;
5 import org.testng.annotations.BeforeMethod;
6 import org.testng.annotations.Test;
7 import org.openqa.selenium.*;
8 import org.openqa.selenium.chrome.ChromeDriver;
9 import org.openqa.selenium.support.ui.Select;
10
11 public class PaymentGatewayTest {
12     private WebDriver driver;
13
14     @BeforeMethod
15     public void setUp() {
16         driver = new ChromeDriver();
17         driver.get("http://127.0.0.1:5500/Demo.html");
18     }
19
20     @Test
21     public void testCardNumber() {
22         WebElement cardNumber = driver.findElement(By.id("cardNumber"));
23         cardNumber.sendKeys("1234567890123456");
24         Assert.assertEquals("1234567890123456", cardNumber.getAttribute("value"));
25     }
26
27     @Test
28     public void testCardHolderName() {
29         WebElement cardHolderName = driver.findElement(By.id("cardHolderName"));
30         cardHolderName.sendKeys("John Doe");
31         Assert.assertEquals("John Doe", cardHolderName.getAttribute("value"));
32     }
33
34     @Test
```

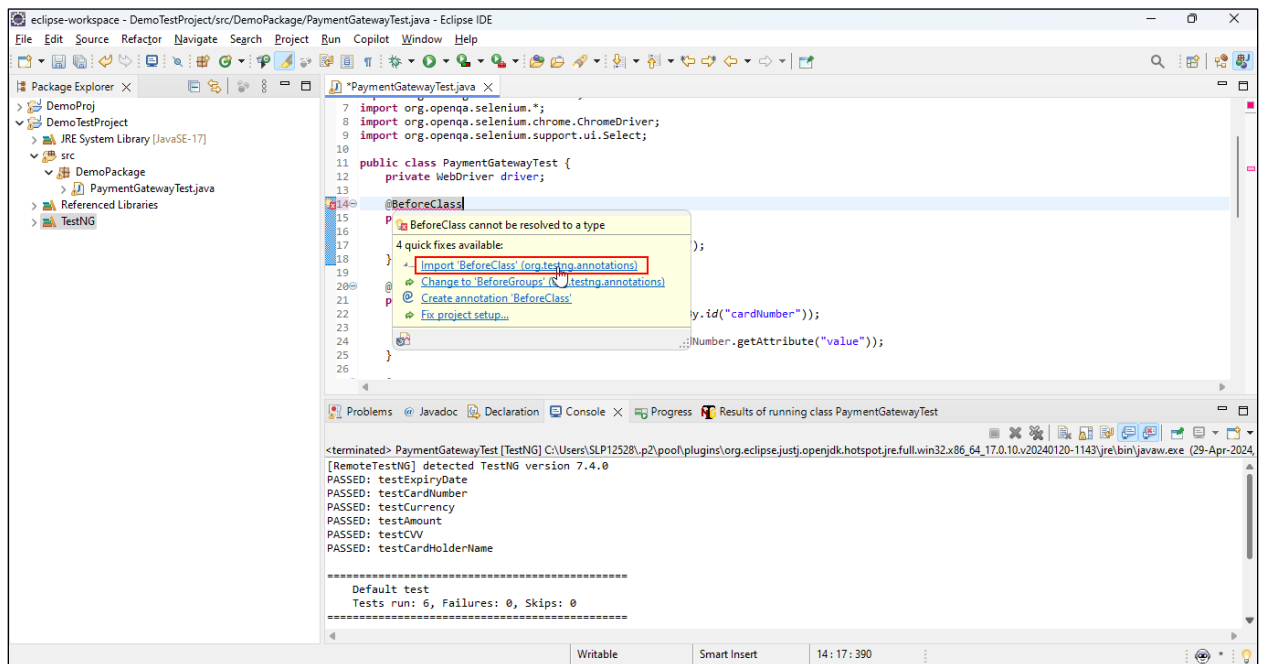
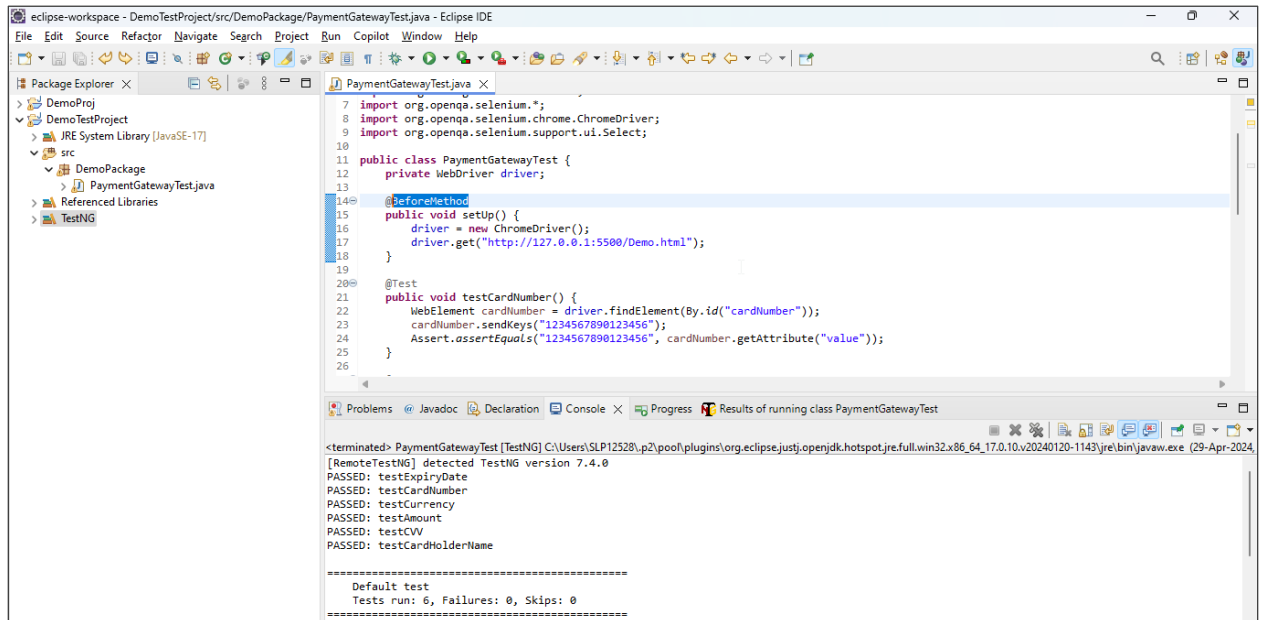
Note: Save the changes made

2.19 Right-click in the code window, scroll to **Run as**, and click on **TestNG Test** as shown in the screenshot below:

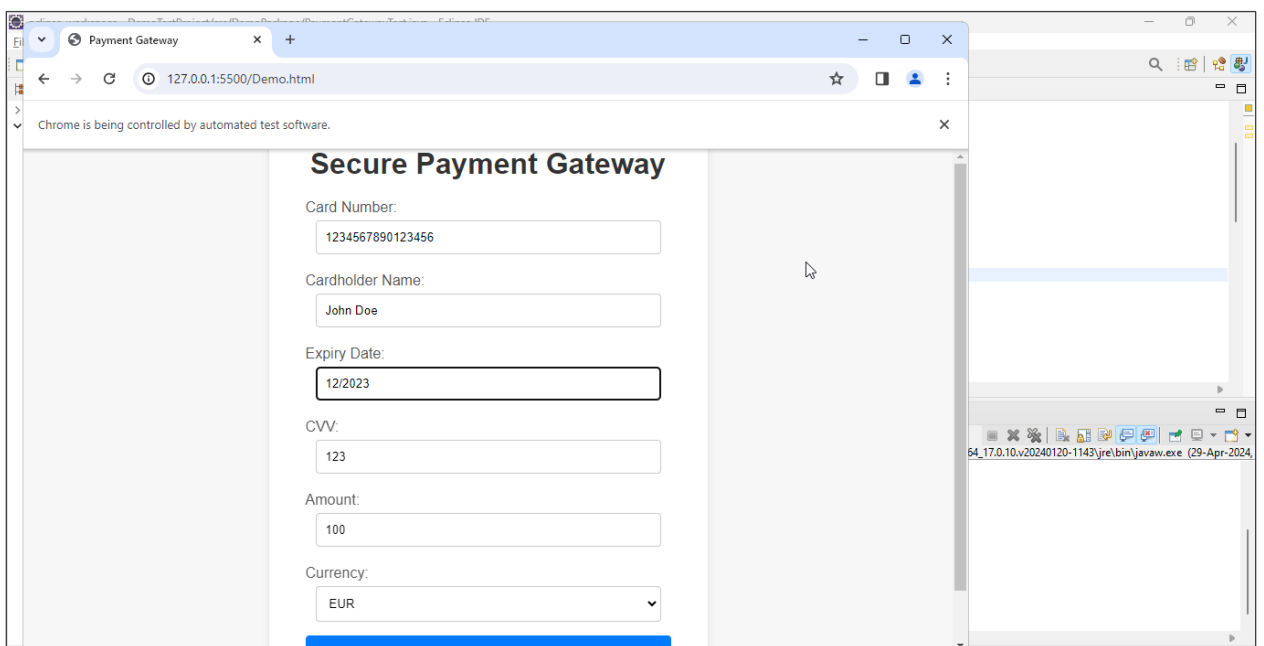
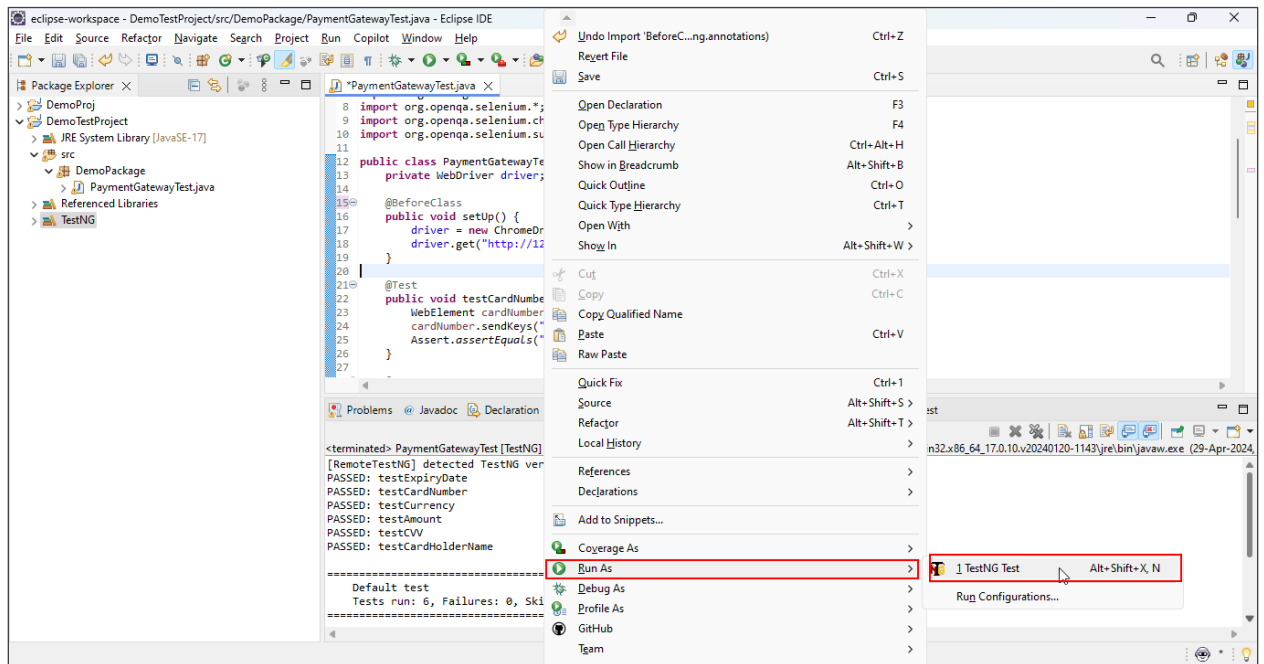


Note: As you are using **BeforeMethod**, a new window pops up every time.

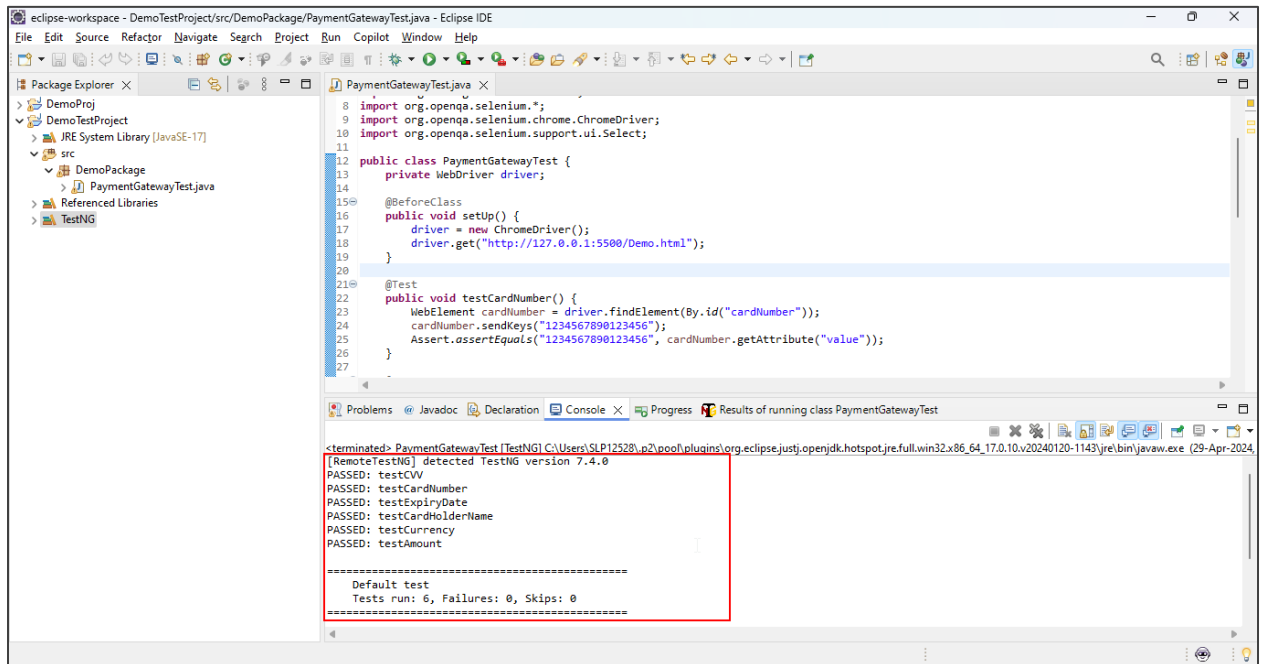
2.20 Navigate to the Java program, replace **BeforeMethod** with **BeforeClass**, and import the same as shown in the screenshot below:



2.21 Right-click in the code window, scroll to **Run as**, and click on **TestNG** Test as shown in the screenshot below:



2.22 Verify the test result in the console as shown in the screenshot below:



The screenshot displays the Eclipse IDE interface. The Package Explorer on the left shows a project named 'DemoTestProject' with a package 'DemoPackage' containing a class 'PaymentGatewayTest.java'. The main editor shows the source code of 'PaymentGatewayTest.java', which includes imports for Selenium, a WebDriver instance, and several test methods. The console at the bottom shows the output of running the class, indicating that all tests passed successfully.

```
8 import org.openqa.selenium.*;
9 import org.openqa.selenium.chrome.ChromeDriver;
10 import org.openqa.selenium.support.ui.Select;
11
12 public class PaymentGatewayTest {
13     private WebDriver driver;
14
15     @BeforeClass
16     public void setUp() {
17         driver = new ChromeDriver();
18         driver.get("http://127.0.0.1:5500/Demo.html");
19     }
20
21     @Test
22     public void testCardNumber() {
23         WebElement cardNumber = driver.findElement(By.id("cardNumber"));
24         cardNumber.sendKeys("1234567890123456");
25         Assert.assertEquals("1234567890123456", cardNumber.getAttribute("value"));
26     }
27 }
```

```
<terminated> PaymentGatewayTest [TestNG] C:\Users\SLP12528\p2\pools\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64_17.0.10.v20240120-1143\jre\bin\javaw.exe (29-Apr-2024)
[RemoteTestNG] detected TestNG version 7.4.0
PASSED: testCVV
PASSED: testCardNumber
PASSED: testExpiryDate
PASSED: testCardHolderName
PASSED: testCurrency
PASSED: testAmount
=====
Default test
Tests run: 6, Failures: 0, Skips: 0
=====
```

By following these steps, you have successfully utilized generative AI to create and execute detailed, scenario-based test scripts. This enhances the efficiency and accuracy of validating the functionality and reliability of a payment gateway system and ensures a robust user experience in software development.