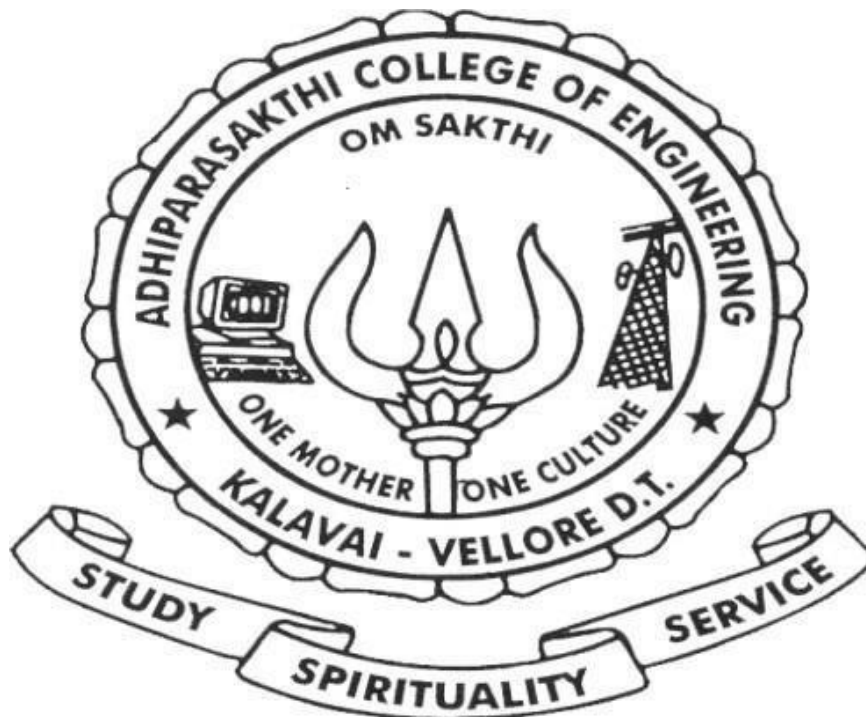


OM SAKTHI

**ADHIPARASAKTHI COLLEGE OF ENGINEERING
(NAAC ACCREDITED)**

G. B. Nagar, Kalavai - 632 506, Ranipet District, Tamil Nadu.



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CCS366-SOFTWARE TESTING AND AUTOMATION

Name:

Year/Semester :

Reg.No.:

OM SAKTHI

**ADHIPARASAKTHI COLLEGE OF ENGINEERING
(NAAC ACCREDITED)**

G. B. Nagar, Kalavai - 632 506, Ranipet District, Tamil Nadu.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING



Name :
Register Number :
Subject Code : CCS366
Subject Name : SOFTWARE TESTING AND AUTOMATION
Semester : VI Year: III

CERTIFICATE

Certified that this is the bonafide record of work done by the above student in
the _____ laboratory during the year 2024-2025.

SIGNATURE OF FACULTY-IN-CHARGE

SIGNATURE OF HOD

Submitted for the Practical Examination held on _____

Internal Examiner

External Examiner

CONTENTS

SL. NO	DATE	NAME OF THE EXPERIMENT	PAGE NO.	MARKS	SIGNATURE
1		Develop the tests plan for testing an e-commerce web/mobile application (www.amazon.in).			
2		design the test cases for Testing E-commerce application.			
3		test the e-commerce application and report the defects in it.			
4		develop a test plan and design the test case for the inventory control system.			
5		execute the test cases against a client server or desktop application and identify the defects.			
6		test the performance of e-commerce application			
7		Automate The Testing Of E-Commerce Application Using The Selenium			
8		integrate the testng with the above test automation.			
9		<p style="text-align: center;">DEVELOP A MINI PROJECT</p> <p>a)build the data driven frame work using selenium and testing.</p> <p>b)Build Page Object Model Using Selenium And Testng.</p> <p>c)build bdd framework with selenium,testng and cucumber</p>			

EXP NO:1
DATE:

**DEVELOP THE TESTS PLAN FOR TESTING AN
ECOMMERCE WEB/MOBILE APPLICATION
(WWW.AMAZON.IN).**

AIM

To Develop the tests plan for testing an e-commerce web/mobile application (www.amazon.in).

PROCEDURE

Step-(1)- Testing Homepage of an ecommerce Site.

Obviously homepage of any website is the very first page that many users see. Homepages in ecommerce websites actually go far beyond than just a landing page. Besides a clickable hero image or product slideshow with auto-scroll features, homepage of an online store can actually serve as a powerful marketing tool. Here are some essential test cases that your ecommerce testing team should focus when testing the homepage:

- Does the hero carousel autoscroll? If so, then at what interval (image refresh time)?
- Does the hero carousel continue to autoscroll when the user hovers mouse over the image? Does the scrolling pause or continue to auto scroll to the next image slide?
- When clicked on one of the slider images or call to action (CTA) buttons, is it taking the customer to the intended page?
- Is the hero slider scrolling too fast?
- Is the homepage loading speed acceptable?
- Does the homepage load and appear the same way across supported browsers and screen resolutions?
- Can the shopping cart be easily located from homepage?
- Can the login/signup button be easily located from homepage?
- Can the contact information be easily located from homepage?
- Is the rest of the homepage content such as the newsletters, banners, social media links in the site footer etc easily viewable?

Step-(2)- Testing the Master page of an Online Shopping Website.

Unlike most other pages, the master page of an ecommerce site is not a stand-alone page but consists of page components that are used in most of the other pages.

Typically, the master page consists of the header, the navigation menu and the footer components. Here are some important test cases that your ecommerce QA team should focus when testing the master page components:

- Do the master page components appear on all page types as expected?
- Does the site logo appear as per the design?
- Is the website logo clickable and takes user to the site homepage?
- Is the navigation menu appearing as per the design?
- Does the nav menu constitute all the sub-menus and mega-menu (if implemented)? – Does the nav menu turn to a burger menu when viewed on smaller screen breakpoints (e.g. mobile and tablet screens)?
- Is the footer appearing as per they design?
- Are all the footer components (e.g. footer menu, footer logos, legal links such as privacy policy, terms & conditions, the ' notice and social icons etc) appearing as expected?
- Does the footer menu turn into an accordion to save space on smaller screen sizes (e.g. mobile and tablet screens)?

Step-(3)- Testing the Login & Registration flow of an eCommerce Site

While all of the ecommerce sites have user registration feature, some of them do allow their customers to purchase products as a guest, i.e. without forcing them to register and create an account, and then with an optional step to create an account in the order confirmation page after the order has been placed. Once an account has been created, the user can choose to log in at any stage during a checkout process. Here are some good test cases for your ecommerce testing team to focus when testing the login and registration process

- Assuming that the site permits this, can the customer purchase items as a guest user?
- Can a guest user easily create an account from the order confirmation page after the order has been placed?
- Can a guest user register as a member from the registration page?
- Once logged in, are the products in the cart getting correctly associated with the logged in member's account?
- Once logged in, is the user still being prompted to sign in? This shouldn't happen.
- Are the login redirects working as expected and whether the user is being redirected to the correct page after completing a particular user journey?
- Are the user sessions being maintained for the intended time period?
- Is the user's session timing out and expiring after the stipulated time?

Step-(4)- Testing the My Account of an Online Web Store

My Account area can differ from site to site but typically they contain pages such as My Account, Account Information, Address Book, My Orders, Newsletter Subscriptions and a link to Log Out. Here are some good test cases for your ecommerce QA team to focus when testing the My Account areas:

- Is the logged out user prevented from accessing My Account areas?
- Is a logged in user able to access My Account areas?
- Is the user able to add, edit, modify and access their account information (e.g. Name, Email, Password etc) in the Account Information area?
- Is the user able to add, edit, modify and access all the addresses in the Address Book? – Are the addresses that the user is adding, editing, modifying in the checkout flow appearing as expected in the Address Book?

- Is the customer able to view, track, cancel and reorder his past orders in My Orders area?
- Is the user able to enable and disable newsletter subscriptions in Newsletter Subscriptions area? – Is the user able to log out of his session using the Log Out link?

Step-(5)– Testing the Search feature of an eCommerce Site .

Believe it or not, the search feature is one of the most-used features in an ecommerce store. Even with ecommerce sites with sophisticated product listing (category) pages and easy to use navigation, customers often find it easier to use the website search feature to find exactly what they are looking for. Here are some important test cases for your ecommerce testing team to focus when testing the Search features:

- Is the search feature robust enough to allow searching by product names, brand names or product categories etc? e.g. iPhone X, Apple iPhone, latest iPhone models, cheap iPhone, iPhones in 2023 or even just iPhones etc.
- Are the search results relevant for all variations of search keywords?
- Does the search result page have sorting options?
- If sorting option is available on search results page, then can customers sort the results based on price (high to low, low to high), date ranges, brand, review / ratings etc?
- Is pagination option available on search results page when a long list of products are displayed?
- How many results are displayed on each page?
- Does the next page load automatically when user scrolls? If not, can the user easily navigate to the next search result page using pagination?

Step-(6)- Testing the Categories or Product Listing Page (PLP) of an Online Shopping Store

While it is neither possible nor prudent to test each and every category page of an ecommerce site, these pages list products that the customers are looking for and hence you must test at least few of the product listing pages randomly to ensure everything is in order. Here are some critical test cases for your ecommerce QA team to focus when testing the Category or Product Listing Page:

- Does this page display the correct products based on the selected category?
- Does this page display proper sorting options based on price (high to low, low to high), date ranges, brand, review / ratings etc?
- Does this page display proper filter options?
- Is pagination option available on this page when a long list of products are displayed?
- How many results are displayed on each page?
- Does the next page load automatically when user scrolls? If not, can the user easily navigate to the next page using pagination?
- Does this page display the correct recommended products based on the selected category?
- Does this page display the correct related products based on the selected category?
- Does this page display the correct featured products based on the selected category?

Step-(7)- Testing the Product Details Page (PDP) of an eCommerce Site

Whether you realize this or not, but this is the page where every other page (home page, the navigation menu, category pages, search result pages etc) is trying to drive traffic in.

Thus, the product details page is the page that ultimately begins the checkout flow journey that results in revenue for every ecommerce sites. Here are some crucial test cases for your ecommerce testing team to focus when testing the Product Details Page:

- Does this page display all the product related information (e.g. product name, SKU, price, discount if any, product images, product specifications, user reviews, Add to Cart button, quantity changer etc) as expected?
- Does this precut display other similar products that were bought by customers who purchased this item?
- Can the customer customize the product if it is a customizable product and available with variations (size, colors etc)?
- Does this page show the stock availability accurately?
- Can the user select required number of items using the quantity selector and then add the product to cart?
- Does this product gets added to cart when added? – Does the quantity in cart get updated when the product is added to cart?

Step-(8)- Testing the Shopping Cart of an Online Shopping Platform

Shopping cart is one of the crucial components of an ecommerce website for obvious reasons. Just like physical shopping baskets, the shopping carts in an ecommerce website allows the customers to select and store multiple different items in their cart and then purchase them together when ready. Here are some key test cases, which should be part of testing a shopping cart.

- Does the cart get updated to show updated quantity when an item is added to it?
- Does the mini-cart (that slides when you click on the cart icon) correctly reflect the items, their quantity and price?
- Can you increase or decrease quantity of an item from the cart and mini-cart?

- Can you remove an item from the cart and mini-cart?
- Can you add the item several times to the cart?
- Can you add several variations (e.g. color, size etc) of the same item to the cart?
- Can you add several items of different types to the cart?
- Can you visit the product detail page by clicking on an item from the cart?
- Does the cart still remembers the items, their quantity and price correctly even if you accidentally close the browser and then reopen it?
- Does the cart still remembers the items, their quantity and price correctly even if you log out as the member and then logs back in?
- Can you apply valid coupon codes and discount vouchers in the cart?

Step-(9)- Testing the Checkout Flow of an ecommerce Site

The Checkout flow is the user journey that takes place when the user decides to move the added product items from the shopping cart and to finally purchase them. In the checkout process, usually the user is required to provide shipping details, billing information and payment details to complete the purchase. Timed out or unsuccessful transactions are one of the top reasons why most online shoppers abandon an ecommerce website without completing their purchase. Failed transactions are thus one of the main reasons of abandoned carts. Here are some of the most crucial test cases to test during the checkout flow.

- If the user is already logged in, does their shipping and billing information get autofilled in checkout?
- If the user is a guest user, can they enter their shipping and billing information?
- Do all the types of supported payment methods work?
- Can customers check out as guests and make payments?

- For returning customers, does the checkout page prompt them to sign in?
- Are sensitive information such as customer's credit card details, PayPal / Amazon Pay / Google Pay / Apple Pay credentials, etc handled securely and not stored on the site's server?
- Is the user taken to Order Confirmation page upon successful checkout?

Step-(10)-Testing the Order Confirmation Page of an Online Shopping Portal

This is the final step of the checkout flow user journey and is displayed only when each steps of the checkout including shipping, billing and payment are successful. Here are some of the most important test cases to test on the order confirmation page.

- Does this page display the Order ID correctly?
- Does this page display the product details correctly?
- Does this page offer an easy way to create account, in case of guest users?

RESULT

Thus the Develop the tests plan for testing an e-commerce web/mobile application (www.amazon.in).

**EXP NO:2 DESIGN THE TEST CASES FOR TESTING E-COMMERCE
DATE: APPLICATION**

AIM

To design the test cases for Testing E-commerce application.

TEST CASES:

Test cases for testing the functionality of the cart of e-commerce application.

Test Case ID	Module Name	Test Scenario	Test Case
TC001	Cart Page	Verify the details on Cart Page	Verify that when user clicks on add to cart, the product should be added to cart.
TC002	Cart Page	Verify the details on Cart Page	Verify that user should be able to continue shopping after adding items to cart.
TC003	Cart Page	Verify the details on Cart Page	Verify that item quantity should be increased if user adds same item in cart again.
TC004	Cart Page	Verify the details on Cart Page	Verify that total amount of all items should be displayed to user.
TC005	Cart Page	Verify the details on Cart Page	Verity that the amount displayed to user

PYTHON CODE FOR TEST CASE OF CART FUNCTIONALITY:

In this code:

- We have a `ShoppingCart` class with methods to `add_item`, `remove_item`, and `get_total_items`.

- We also have a `TestShoppingCart` class that inherits from `unittest.TestCase` to define test cases.
- In the `setUp` method of the test class, we create an instance of `ShoppingCart` and add some items to it.
- We then have three test methods:
 - `test_add_item`: Tests the `add_item` method.
 - `test_remove_item`: Tests the `remove_item` method.
 - `test_get_total_items`: Tests the `get_total_items` method.
- Finally, we run the tests using `unittest.main()`

```
import unittest

class ShoppingCart:
    def __init__(self):
        self.items = {}

    def add_item(self, item, quantity):
        if item in self.items:
            self.items[item] += quantity
        else:
            self.items[item] = quantity

    def remove_item(self, item, quantity):
        if item in self.items:
            if self.items[item] <= quantity:
                del self.items[item]
            else:
                self.items[item] -= quantity

    def get_total_items(self):
        return sum(self.items.values())

class TestShoppingCart(unittest.TestCase):
    def setUp(self):
        self.cart = ShoppingCart()
```

```
        self.cart.add_item("apple", 3)
        self.cart.add_item("banana", 2)
    def test_add_item(self):
        self.cart.add_item("apple", 2)
        self.assertEqual(self.cart.items["apple"], 5)
    def test_remove_item(self):
        self.cart.remove_item("apple", 2)
        self.assertEqual(self.cart.items["apple"], 1)
    def test_get_total_items(self):
        self.assertEqual(self.cart.get_total_items(), 5)
if __name__ == "__main__":
    unittest.main()
```

OUTPUT for positive case:

Ran 3 tests in 0.091s

OK

OUTPUT for negative case:

If we have some mistakes in estimated values then it shows the attribute not equal error. for example if we change the estimated value of test_get_total_items from 5 to 7 function then it shows the following error.

FAIL: test_get_total_items (__main__.TestShoppingCart.test_get_total_items)

Traceback (most recent call last):

File "C:/Users/welcome/Desktop/New folder/cart.py", line 38, in
test_get_total_items

```
    self.assertEqual(self.cart.get_total_items(), 7)
```

AssertionError: 5 != 7

Ran 3 tests in 0.063s

FAILED (failures=1)

Test cases for testing the functionality of the homepage of e-commerce application.

Test Case ID	Module Name	Test Scenario	Test Case
TC001	Home Page	Verify the details on Home page	Verify that home page is displayed after login or not.
TC002	Home Page	Verify the details on Home page	Verify that user name is displayed on home page or not
TC003	Home Page	Verify the details on Home page	Verify that featured products are displayed on home page
TC004	Home Page	Verify the details on Home page	Verify that Search Functionality is present on home page.
TC005	Home Page	Verify the details on Home page	Verify the home page of application on different browsers.
TC006	Home Page	Verify the details on Home page	Verify the alignment on the home page
TC007	Home Page	Verify the details on Home page	Verify that products displayed on home page are clickable.
TC008	Home Page	Verify the details on Home page	Verify that when user clicks on a product, user should be redirected to product specification page.

PYTHON CODE FOR TEST CASE OF HOME PAGE:

Before executing the code install the selenim module by typing the following command. `pip install selenium==3.141.0` and `pip install --upgrade urllib3==1.26.1`.

```
from selenium import webdriver
import time
def test_homepage(driver, homepage_url, expected_title):
    driver.get(homepage_url)
    time.sleep(2) # Wait for the page to load completely
    actual_title = driver.title
    if actual_title == expected_title:
        print("Homepage test passed")
    else:
        print("Homepage test failed")
if __name__ == "__main__":
    # Replace 'path_to_webdriver' with the path to your webdriver executable
    driver = webdriver.Chrome('E:\chromedriver-win64\chromedriver.exe')
    # Replace 'your_ecommerce_homepage_url' with the URL of your e-commerce
    application's homepage
    homepage_url = 'http://www.flipkart.com'
    expected_title = 'Online Shopping Site for Mobiles, Electronics, Furniture,
    Grocery, Lifestyle, Books & More. Best Offers!'
    test_homepage(driver, homepage_url, expected_title)
    # Close the browser window
    driver.quit()
```

OUTPUT for positive case:

Warning (from warnings module):

File "C:/Users/welcome/Desktop/New folder/test.py", line 16

```
driver = webdriver.Chrome('E:\chromedriver-win64\chromedriver.exe')
```

SyntaxWarning: invalid escape sequence '\c'

>>>

```
=====
===== RESTART: C:/Users/welcome/Desktop/New folder/test.py
=====
=====
```

Homepage test passed.

OUTPUT for negative case:

if we change the name of the home page from expected name to the name “flipkart_name”.Then it shows the following error.

Warning (from warnings module):

File "C:/Users/welcome/Desktop/New folder/test.py", line 16

```
driver = webdriver.Chrome('E:\chromedriver-win64\chromedriver.exe')
```

SyntaxWarning: invalid escape sequence '\c'

>>>

```
=====
===== RESTART: C:/Users/welcome/Desktop/New folder/test.py
=====
=====
```

Homepage test failed.

RESULT

Thus the test cases were designed for the testing of the e-commerce application.

EXP NO:3 TEST THE E-COMMERCE APPLICATION AND REPORT THE DEFECTS IN IT.

AIM

To test the e-commerce application and report the defects in it.

PROCEDURE

Step-(1)-configuring the Junit in the eclipse.

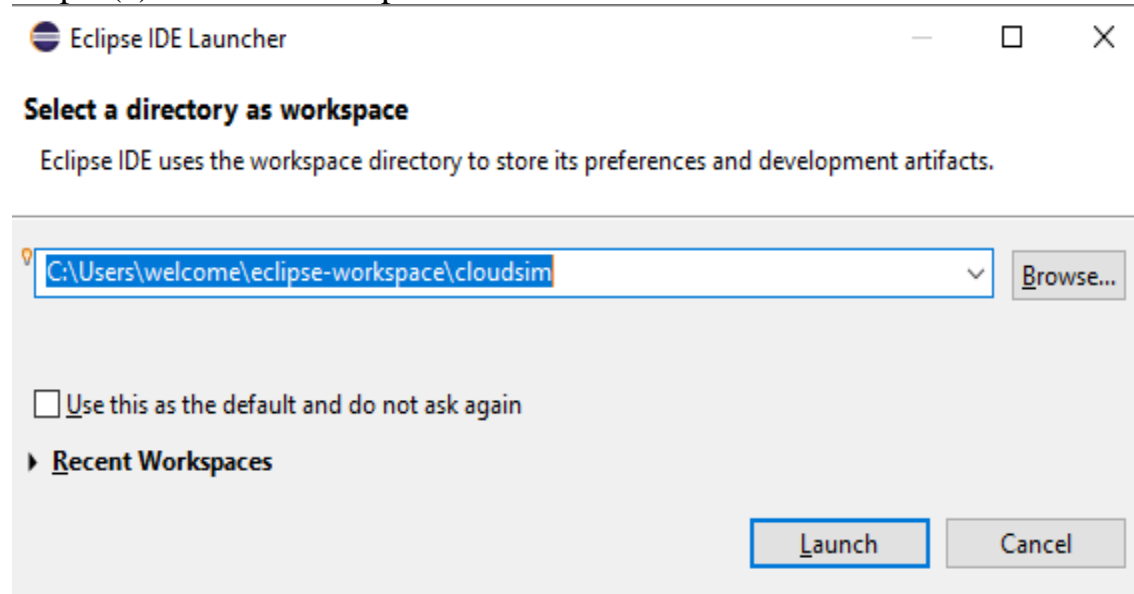
Step-(2)-developing the test case for functionality of cart.

Step-(3)-executing the test cases.

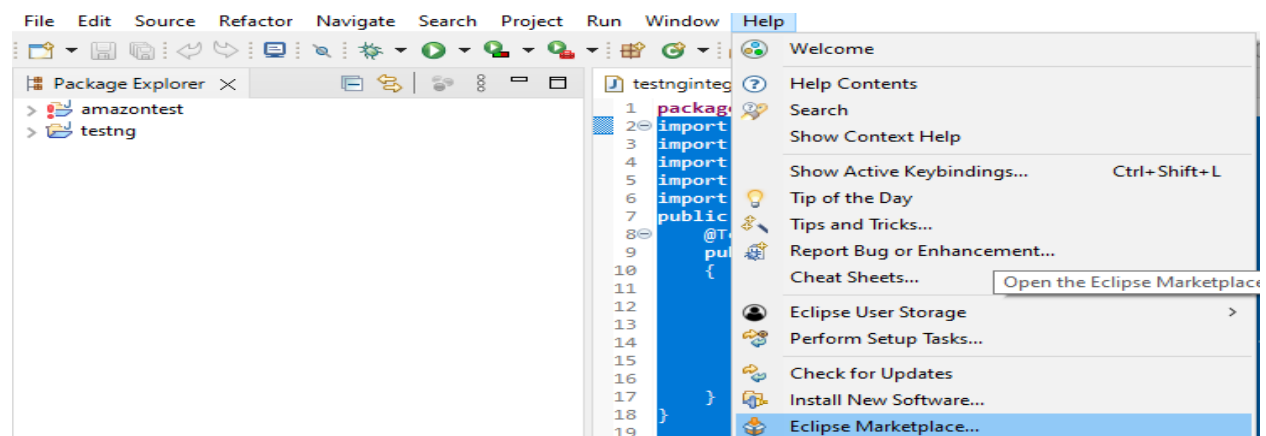
Step-(4)-verify the results and the defects.

CONFIGURING THE JUNIT IN THE ECLIPSE:

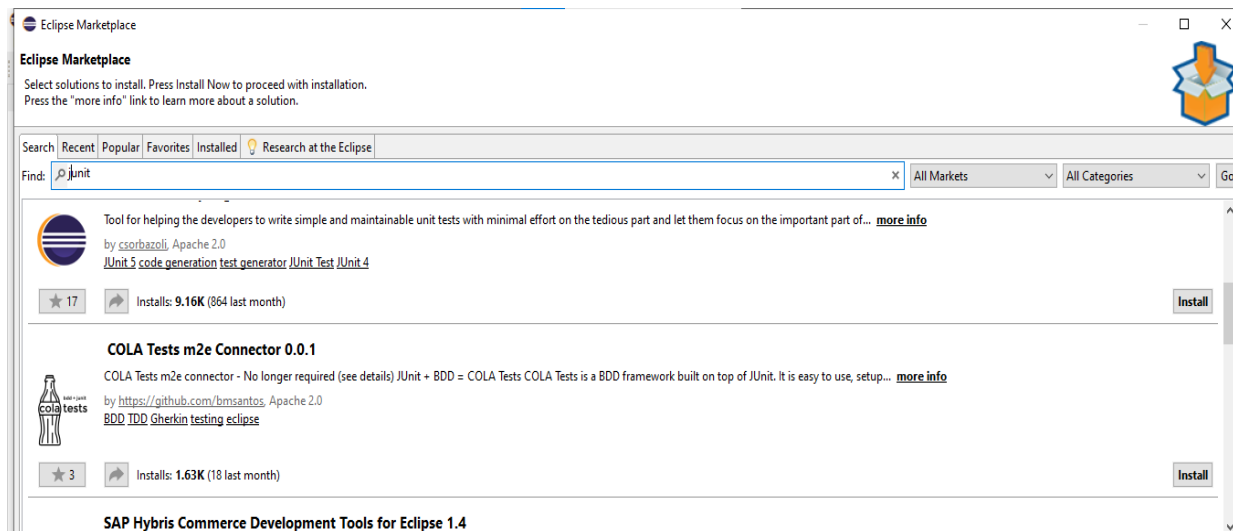
Step1-(a)-launch the eclipse.



Step1-(b)-click on the help under the help click on the eclipse market place
help→eclipse market place

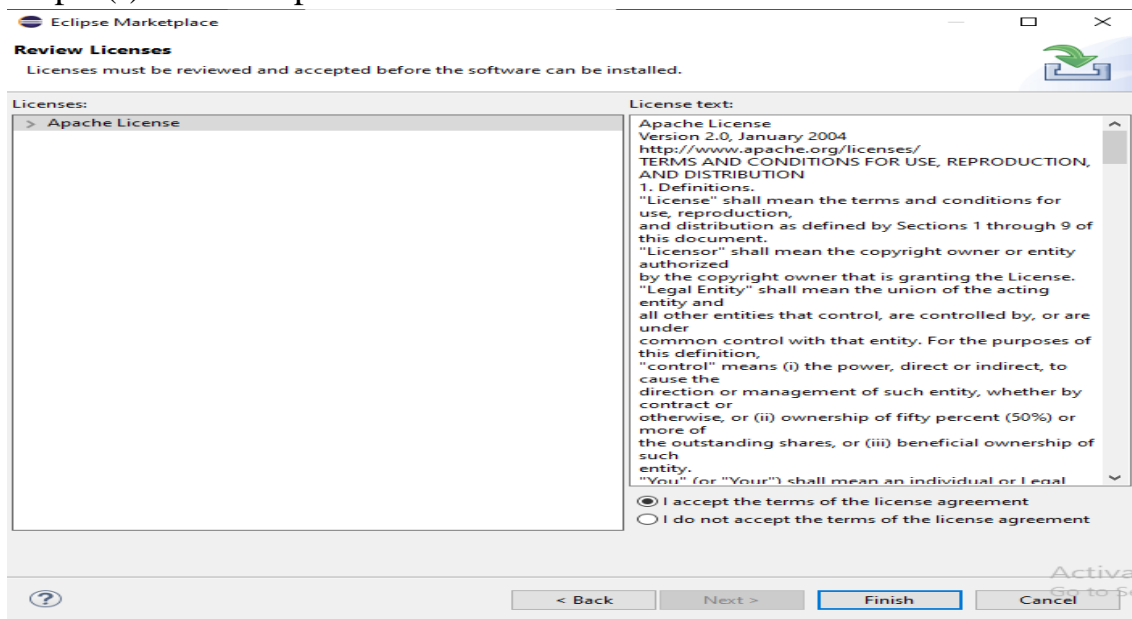


Step1-(c)-Then you can able to view the following window and type Junit in find.



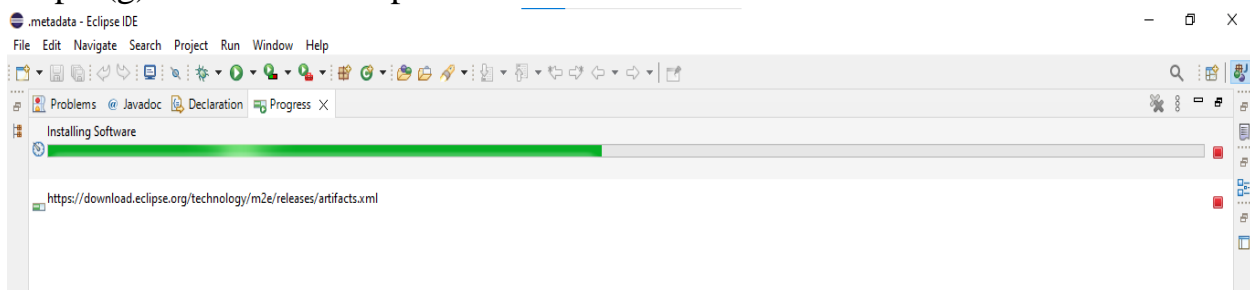
Click on the install button.

Step1-(f)-then accept the terms and condition.

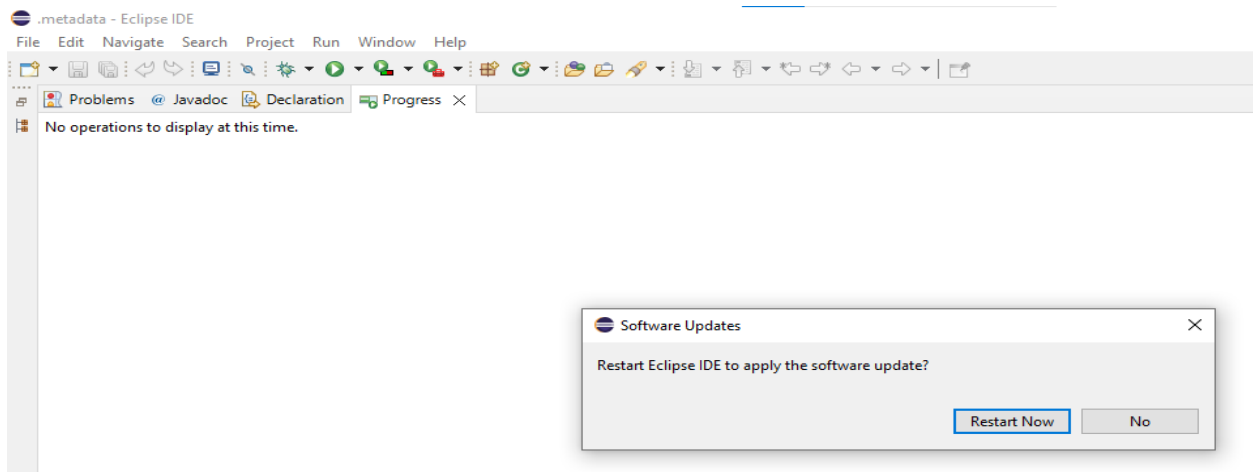


Click on finish to proceed further.

Step1-(g)-the installation process is carried out as shown below.



Step1-(h)-after the completion of installation it asks for the restart and then click restart.

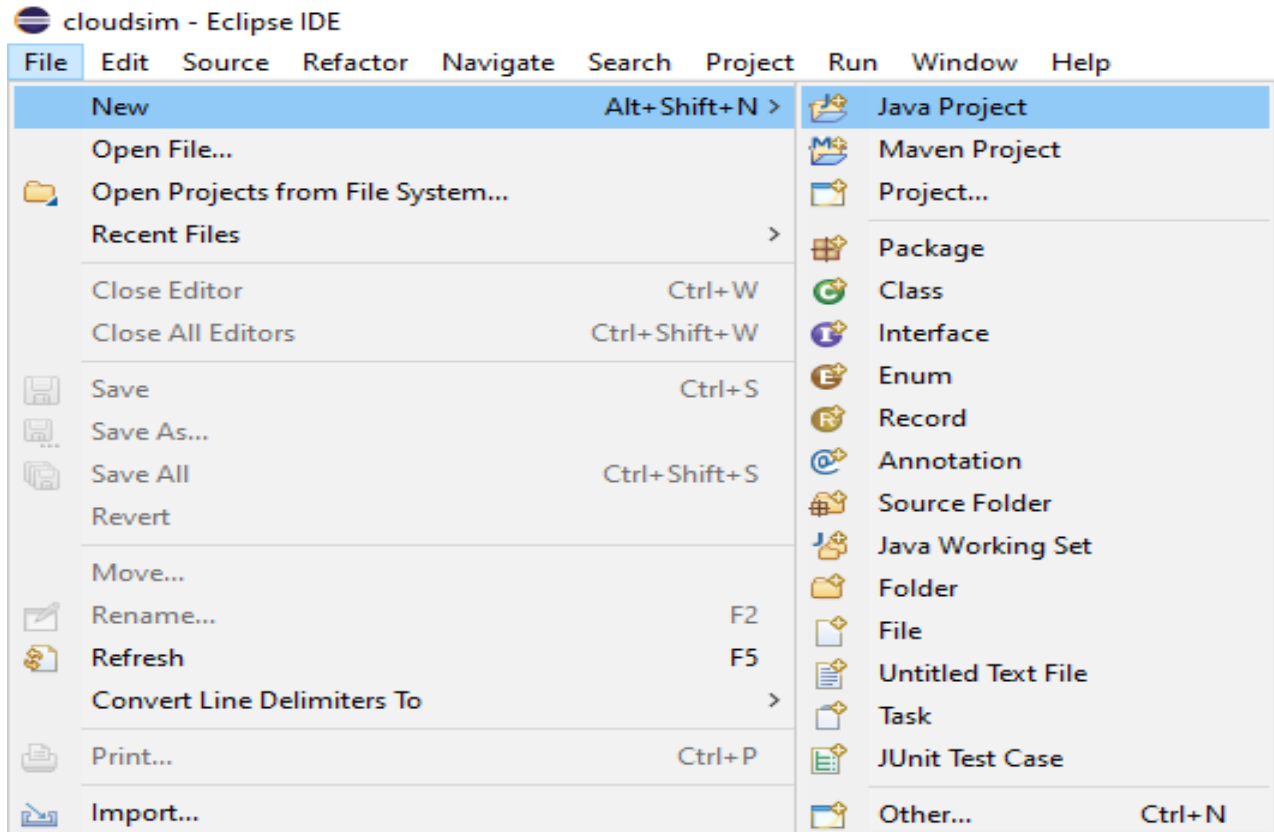


Now the junit was successfully configured with eclipse ide.

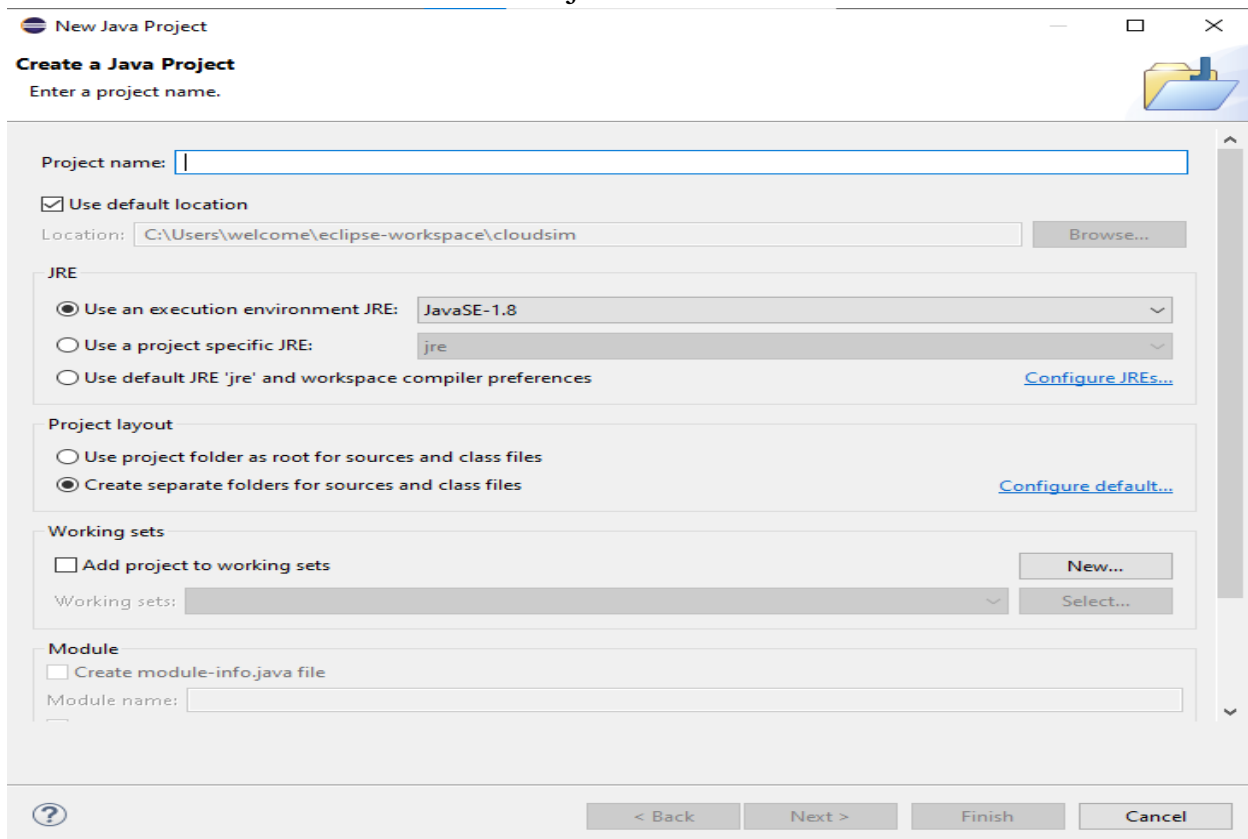
DEVELOPING THE TEST CASE FOR FUNCTIONALITY OF CART:

STEP2-(a)-then click on the new “file”,under this click on the “new”,under the new click on the “java project”.

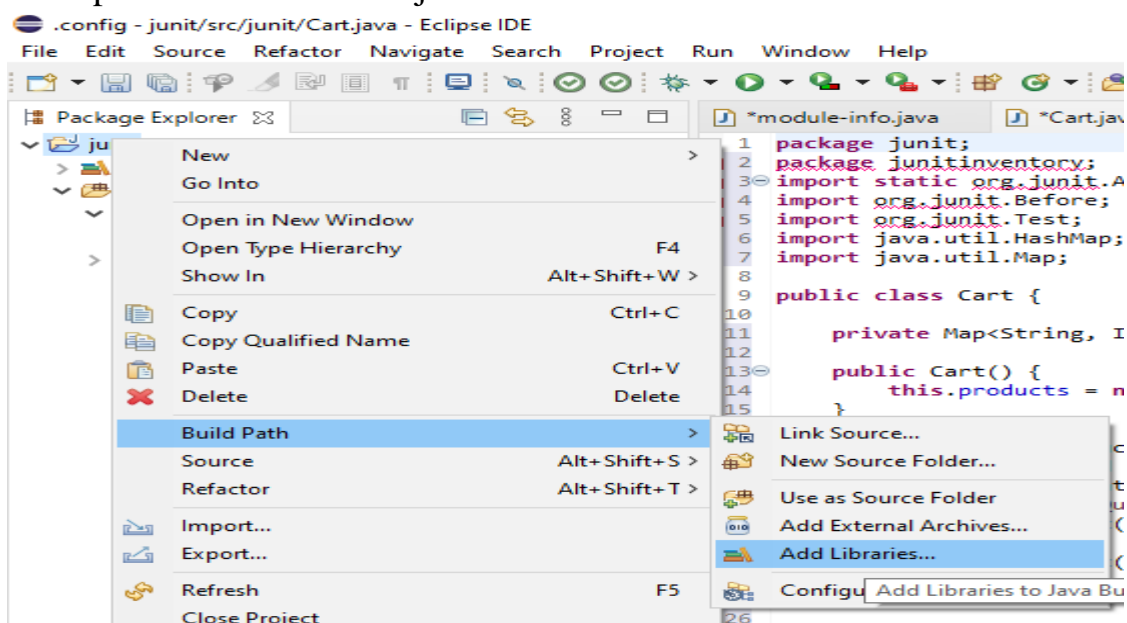
file→new→java project.

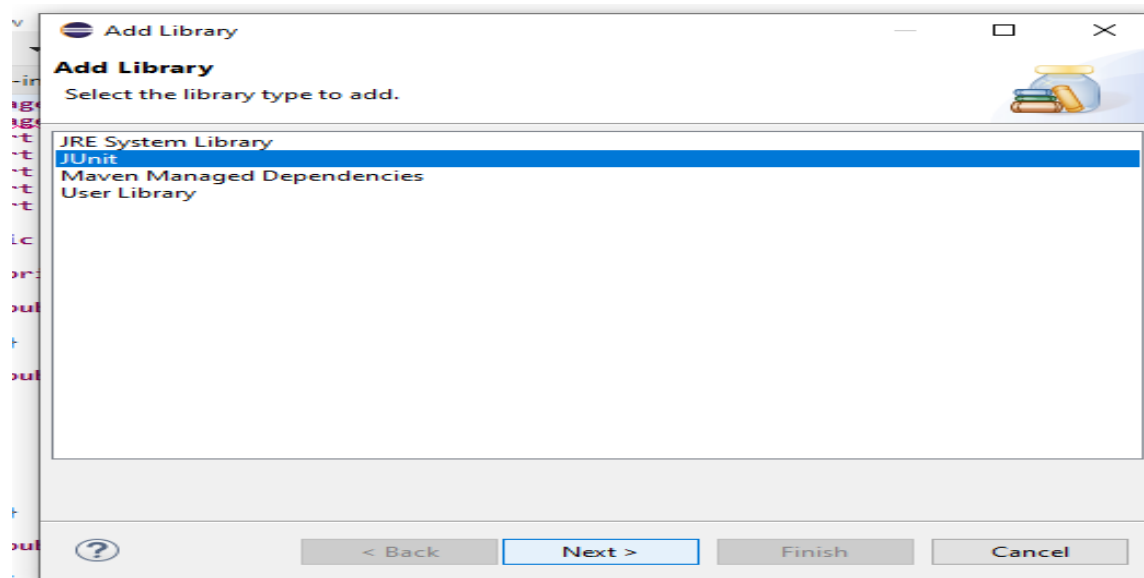


STEP2-(b)-after clicking on the “java project”.enter the name of the java project and set the execution environment as java S.E 1.8.



STEP2-(c)-to add the junit libraries right click on the project and click on the build path under build path click on the add library and click on the junit library.
build path→add libraries→junit.





Click on next to proceed further.

STEP2-(d)-create the new class by right clicking on the project created.
new → class.

name the class as Cart and type the following code.

```
package junit;
import static org.junit.Assert.*;
import org.junit.Before;
import org.junit.Test;
import java.util.HashMap;
import java.util.Map;

public class Cart {

    private Map<String, Integer> products;

    public Cart() {
        this.products = new HashMap<>();
    }

    public void addProduct(Product product, int quantity) {
        String productId = product.getProductId();
        if (products.containsKey(productId)) {
            int currentQuantity = products.get(productId);
            products.put(productId, currentQuantity + quantity);
        } else {
            products.put(productId, quantity);
        }
    }

    public void removeProduct(String productId) {
        products.remove(productId);
    }

    public boolean containsProduct(String productId) {
```

```

        return products.containsKey(productId);
    }

    public int getProductQuantity(String productId) {
        return products.getDefault(productId, 0);
    }

    public double calculateTotalPrice(Map<String, Double> productPrices) {
        double totalPrice = 0.0;
        for (Map.Entry<String, Integer> entry : products.entrySet()) {
            String productId = entry.getKey();
            int quantity = entry.getValue();
            double price = productPrices.getDefault(productId, 0.0);
            totalPrice += price * quantity;
        }
        return totalPrice;
    }
}

```

Again create the new class by right clicking on the project created.
new→class.

name the class as Product and type the following code.

```

package junit;
class Product {
    private String productId;
    private String name;
    private double price;

    public Product(String productId, String name, double price) {
        this.productId = productId;
        this.name = name;
        this.price = price;
    }

    public String getProductId() {
        return productId;
    }

    public String getName() {
        return name;
    }

    public double getPrice() {
        return price;
    }
}

```

Again create the new class by right clicking on the project created.
new→class.

name the class as ECommerceTest and type the following code.

```

package junit;
import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertFalse;
import static org.junit.Assert.assertTrue;
import java.util.HashMap;

```

```
import java.util.Map;
import org.junit.Before;
import org.junit.Test;
public class EcommerceTest {

    private Cart cart;

    @Before
    public void setUp() {
        // Initialize the cart
        cart = new Cart();
    }

    @Test
    public void testAddToCart() {
        // Create a sample product
        Product product = new Product("001", "Product 1", 10.99);

        // Add the product to the cart
        cart.addProduct(product, 2);

        // Check if the product was added successfully
        assertTrue(cart.containsProduct("001"));
        assertEquals(2, cart.getProductQuantity("001"));
    }

    @Test
    public void testRemoveFromCart() {
        // Create a sample product
        Product product = new Product("002", "Product 2", 20.99);

        // Add the product to the cart
        cart.addProduct(product, 1);

        // Remove the product from the cart
        cart.removeProduct("002");

        // Check if the product was removed successfully
        assertFalse(cart.containsProduct("002"));
    }

    @Test
    public void testCalculateTotalPrice() {
        // Create some sample products
        Product product1 = new Product("003", "Product 3", 15.49);
        Product product2 = new Product("004", "Product 4", 5.99);

        // Add the products to the cart
        cart.addProduct(product1, 2);
        cart.addProduct(product2, 3);

        // Calculate the total price
        Map<String, Double> productPrices = new HashMap<>();
        productPrices.put("003", 15.49);
        productPrices.put("004", 5.99);
    }
}
```



```

    double totalPrice = cart.calculateTotalPrice(productPrices);

    // Check if the total price is calculated correctly
    assertEquals(48.95, totalPrice, 0.01);
}

public static void main(String[] args) {
    // Run the JUnit test cases
    org.junit.runner.JUnitCore.main("ECommerceTest");
}
}

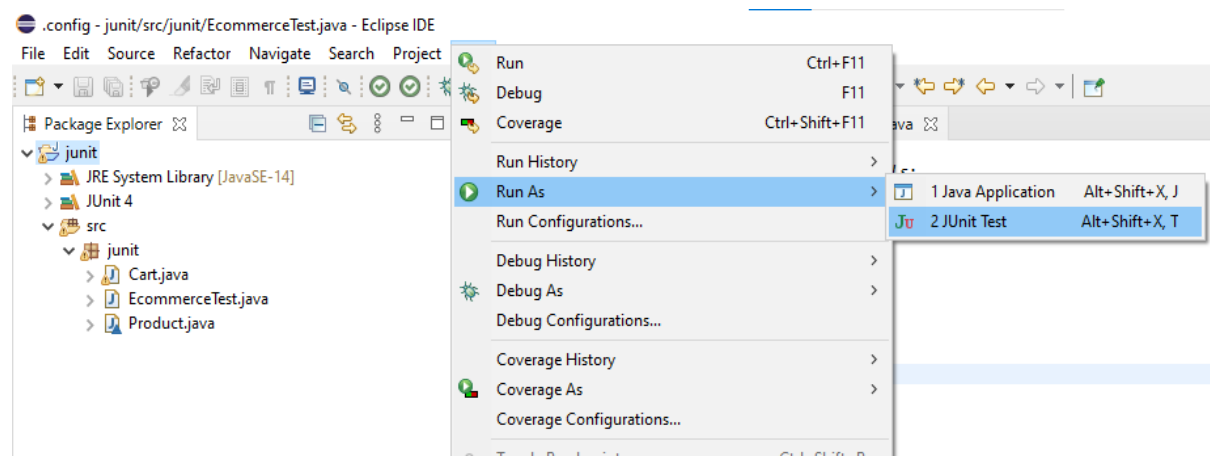
```

Save all three classes and test case was successfully designed.

EXECUTING THE TEST CASES:

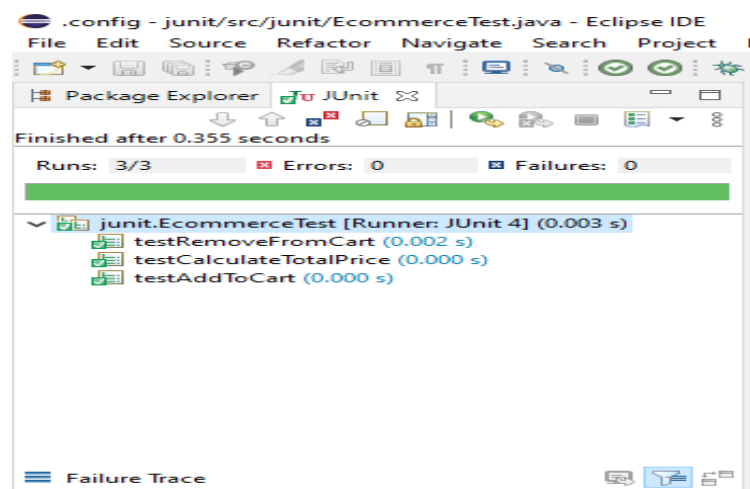
Step3-(a)-Run the test case by clicking on the “Run as” option.

Run as → JUnit Test



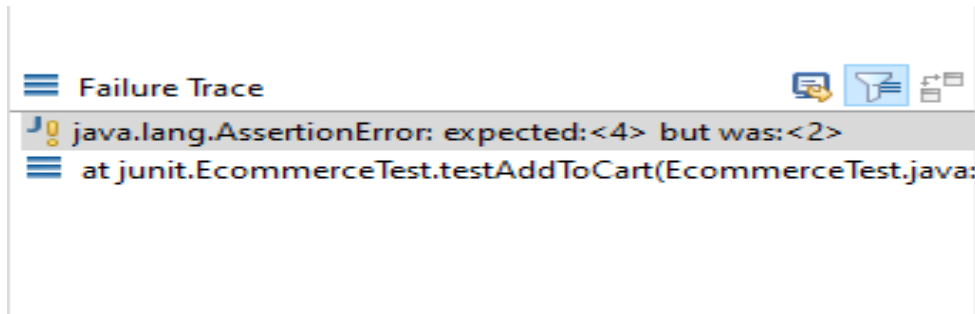
VERIFY THE RESULTS AND THE DEFECTS:

OUTPUT for positive case (for the positive case all the tests are passed).



OUTPUT for negative case(defects):

if we change the number of the products from value to the value to any arbitrary value in the function testAddToCart. Then it shows the following error in the failure trace.



RESULT

Thus the e-commerce application was tested and defects were reported successfully.

**EXP NO:4 DEVELOP A TEST PLAN AND DESIGN THE TEST CASES
DATE: FOR THE INVENTORY CONTROL SYSTEM**

AIM

To develop a test plan and design the test case for the inventory control system.

Test Plan for an Inventory Control System

1. Introduction

The Inventory Control System is designed to efficiently manage inventory, track stock levels, and ensure accurate stock information. This test plan outlines the testing approach, scope, objectives, and methodologies to verify the functionality, performance, and usability of the system.

2. Objectives

The primary objectives of this test plan are to:

- Validate that the Inventory Control System meets the functional requirements.
- Verify the accuracy and consistency of inventory tracking.
- Assess system performance under varying loads.
- Ensure the system's security mechanisms are effective.
- Evaluate the system's usability and user-friendliness.

3. Test Scope

The testing will cover the following aspects of the Inventory Control System:

- User authentication and authorization.
- Adding, updating, and deleting products in the inventory.
- Monitoring stock levels and receiving alerts for low stock.
- Generating various inventory reports.
- User interfaces for different user roles (admin, manager, employee).

4. Test Environment

- Hardware: Servers, workstations, mobile devices for testing different platforms.
- Software: Inventory Control System build, web browsers, databases.
- Network: LAN/WAN with various network conditions (latency, bandwidth).
- Test Data: Realistic inventory data and user scenarios for testing.

5. Test Data

- Test data will include a range of products, quantities, prices, and suppliers.
- Both normal and boundary test cases will be created to cover various scenarios.
- Data will be structured to validate different calculations and reports.

6. Test Cases

- A comprehensive list of test cases will be created for functional, performance, security, and usability testing.
- Each test case will include:
 - Test case ID
 - Description of the scenario
 - Preconditions
 - Steps to execute the test
 - Expected results
 - Actual results

▢ Pass/fail status

7. Types of Testing

- Unit Testing: Testing individual modules and functions.
- Integration Testing: Testing interactions between different system components.
- Functional Testing: Validating functional requirements.
- Performance Testing: Evaluating system response times and resource usage.
- Security Testing: Checking for vulnerabilities and unauthorized access.
- Usability Testing: Assessing user-friendliness and navigation.

8. Test Schedule

- Testing Phases and Estimated Timeline:
 - Unit Testing: 1 week
 - Integration Testing: 2 weeks
 - Functional Testing: 2 weeks
 - Performance Testing: 1 week

- Security Testing: 1 week
- Usability Testing: 1 week

9. Defect Management

- Defects will be logged using a defect tracking tool.
- Defects will be classified by severity (critical, major, minor) and priority.
- Defects will be retested after resolution.

10. Risk Assessment

- Identified risks include data loss, system downtime, security breaches.
- Mitigation strategies include regular data backups, redundancy, security protocols.

11. Test Deliverables Test cases document

- ☐ Test execution results and defect reports
- Performance testing results and analysis
- Usability testing observations and feedback

DESIGNING THE TEST CASES IN ECLIPSE:

Step-1-launch the eclipse.

Step-2-create the new project and also add the Library JUnit.

Step-3-create the three new classes as shown below.

create the new class by right clicking on the project created.

new→class.

name the class as InventoryManager and type the following code.

```
package inventory;
import java.util.HashMap;
import java.util.Map;

public class InventoryManager {

    private Map<String, InventoryItem> inventory;

    public InventoryManager() {
        this.inventory = new HashMap<>();
    }

    public void addItem(InventoryItem item) {
        inventory.put(item.getItemId(), item);
    }

    public void removeItem(String itemId) {
        inventory.remove(itemId);
    }
}
```

```

public boolean containsItem(String itemId) {
    return inventory.containsKey(itemId);
}

public void sellItem(String itemId, int quantitySold) {
    if (inventory.containsKey(itemId)) {
        InventoryItem item = inventory.get(itemId);
        int currentQuantity = item.getQuantity();
        if (currentQuantity >= quantitySold) {
            item.setQuantity(currentQuantity - quantitySold);
        } else {
            throw new IllegalArgumentException("Not enough quantity to sell.");
        }
    } else {
        throw new IllegalArgumentException("Item not found in inventory.");
    }
}

public void restockItem(String itemId, int quantityToAdd) {
    if (inventory.containsKey(itemId)) {
        InventoryItem item = inventory.get(itemId);
        item.setQuantity(item.getQuantity() + quantityToAdd);
    } else {
        throw new IllegalArgumentException("Item not found in inventory.");
    }
}

public int getItemQuantity(String itemId) {
    if (inventory.containsKey(itemId)) {
        return inventory.get(itemId).getQuantity();
    } else {
        throw new IllegalArgumentException("Item not found in inventory.");
    }
}
}

```

Again create the new class by right clicking on the project created.
new → class.

name the class as InventoryItem and type the following code.

```

package inventory;
public class InventoryItem {
    private String itemId;
    private String itemName;
    private int quantity;
    private double price;

    public InventoryItem(String itemId, String itemName, int quantity, double price)
    {
        this.itemId = itemId;
        this.itemName = itemName;
        this.quantity = quantity;
        this.price = price;
    }
}

```

```

    }

    public String getItemId() {
        return itemId;
    }

    public String getItemName() {
        return itemName;
    }

    public int getQuantity() {
        return quantity;
    }

    public void setQuantity(int quantity) {
        this.quantity = quantity;
    }

    public double getPrice() {
        return price;
    }

    public void setPrice(double price) {
        this.price = price;
    }
}

```

Again create the new class by right clicking on the project created.
new → class.

name the class as InventorySystemTest and type the following code.

```

package inventory;
import static org.junit.Assert.*;
import org.junit.Before;
import org.junit.Test;

public class InventorySystemTest {

    private InventoryManager inventoryManager;

    @Before
    public void setUp() {
        // Initialize the inventory manager
        inventoryManager = new InventoryManager();
    }

    @Test
    public void testAddItem() {
        // Create a sample item
        InventoryItem item = new InventoryItem("001", "Item 1", 11, 5.99);

        // Add the item to the inventory
        inventoryManager.addItem(item);

        // Check if the item was added successfully
        assertTrue(inventoryManager.containsItem("001"));
    }
}

```

```

    }

    @Test
    public void testRemoveItem() {
        // Create a sample item
        InventoryItem item = new InventoryItem("002", "Item 2", 20, 9.99);

        // Add the item to the inventory
        inventoryManager.addItem(item);

        // Remove the item from the inventory
        inventoryManager.removeItem("002");

        // Check if the item was removed successfully
        assertFalse(inventoryManager.containsItem("002"));
    }

    @Test
    public void testSellItem() {
        // Create a sample item
        InventoryItem item = new InventoryItem("003", "Item 3", 15, 7.49);

        // Add the item to the inventory
        inventoryManager.addItem(item);

        // Sell some quantity of the item
        inventoryManager.sellItem("003", 5);

        // Check if the quantity is updated after selling
        assertEquals(10, inventoryManager.getItemQuantity("003"));
    }

    @Test
    public void testRestockItem() {
        // Create a sample item
        InventoryItem item = new InventoryItem("004", "Item 4", 6, 90);

        // Add the item to the inventory
        inventoryManager.addItem(item);

        // Restock the item
        inventoryManager.restockItem("004", 10);

        // Check if the quantity is updated after restocking
        assertEquals(16, inventoryManager.getItemQuantity("004"));
    }

    public static void main(String[] args) {
        // Run the JUnit test cases
        org.junit.runner.JUnitCore.main("InventorySystemTest");
    }
}

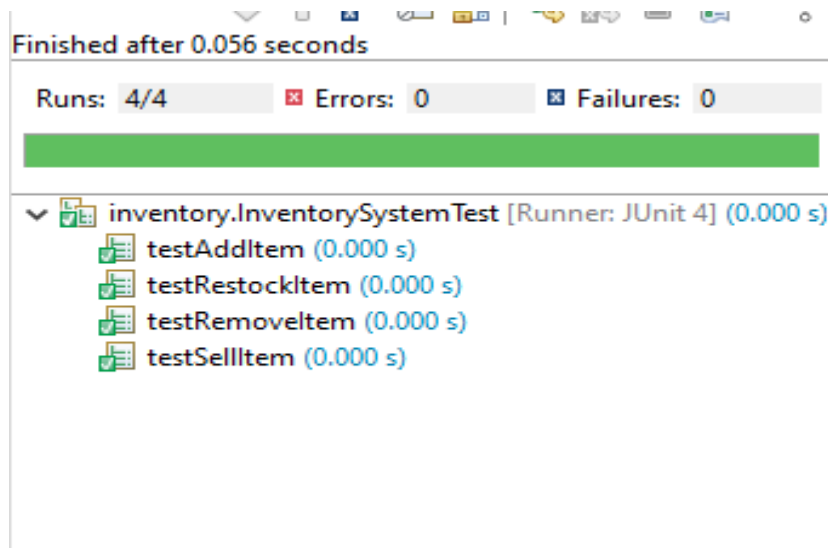
```

Thus the test cases were developed successfully and out is given below.

OUTPUT:

Run the test case by clicking on the “Run as” option.

Run as → JUnit Test



RESULT

Thus the test plan was developed and the test case for the inventory control system was designed.

**EXNO:5 EXECUTE THE TEST CASES AGAINST A CLIENT SERVER
DATE: OR DESKTOP APPLICATION AND IDENTIFY THE
DEFECTS.**

AIM

To execute the test cases against a client server or desktop application and identify the defects.

PROCEDURE

Step-1-launch the eclipse.

Step-2-create the new project and also add the Library JUnit.

Step-3-create the two new classes as shown below.

create the new class by right clicking on the project created.

new → class.

name the class as Client and type the following code.

```
import java.io.*;
import java.net.*;
import org.junit.Test;
import java.util.Scanner;

public class Client{

    @Test

    public void main()throws IOException{

        Socket socket=new Socket("localhost",12345);

        BufferedReader in=new BufferedReader(new
        InputStreamReader(socket.getInputStream()));

        PrintWriter out=new PrintWriter(socket.getOutputStream(),true);

        System.out.println("Enter the message to server: ");

        Scanner inp=new Scanner(System.in);

        String message = inp.nextLine();

        out.println(message);

        String response = in.readLine();
```

```
System.out.println("Server response:"+response);
socket.close();
}
}
```

create the new class by right clicking on the project created.
new → class.
name the class as Server and type the following code.

```
import java.io.*;
import org.junit.Test;
import java.net.*;

public class Server{
    @Test
    public void main()throws IOException{
        ServerSocket serverSocket=new ServerSocket(12345);
        System.out.println("Server is running and waiting for a connection...");
        Socket clientSocket = serverSocket.accept();
        System.out.println("Connection established with a client.");
        BufferedReader in=new BufferedReader(new
        InputStreamReader(clientSocket.getInputStream()));
        PrintWriter out=new PrintWriter(clientSocket.getOutputStream(),true);
        String message = in.readLine();
        System.out.println("Received from client:"+message);
        out.println("Message received: " + message);
        clientSocket.close();
        serverSocket.close();
    }
}
```

OUTPUT:

The output of the server program.

Server is running and waiting for a connection...

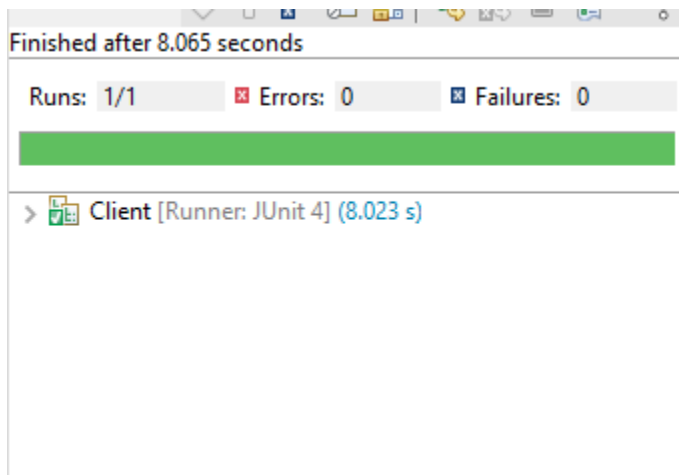
The output of the client program.

Enter the message to server:

hello

Server response:Message received: hello

And the corresponding test results.



DEFECTS:

- First defect that occur is the connection issues between the client and the server
- Some authentication and authorization issues occur.
- Data transferred between the client and server may become corrupted during transmission
- When multiple clients access and modify the same data simultaneously, conflicts can arise.
- Some security issues like SQLInjection and CrossSite Scripting may occur.
- Differences in client and server versions can result in incompatibility issues.
- The application might not release resources properly, leading to resource exhaustion.

RESULT

Thus the test cases against a client server was executed and defects were identified.

EXP NO:6

TEST THE PERFORMANCE OF THE E-COMMERCE APPLICATION.

DATE:

AIM

To test the performance of e-commerce application using the apache jmeter.

APACHE-JMETER

JMeter also known as 'Apache JMeter' is an open source, 100% java based application with a graphical interface. It is designed to analyse and measure the performance and load functional behaviour of web application and variety of services.

JMeter is mainly used for testing Web application or FTP application but currently, it is applicable in functional testing, JDBC database connections, Web services, generic TCP connections and OS native processes. You can perform various testing activities like Performance, Load, Stress, Regression and Functional testing, in order to get accurate performance metrics against your web server.

JMeter was originally written and developed by **Stefano Mazzocchi** of the Apache Software Foundation. It was primarily written to test the performance of Apache JServ(currently known as Apache Tomcat project).Apache redesigned JMeter to enhance the GUI, to add more features and functional testing capabilities.

JMeter is not a browser and it doesn't render html pages like any browser does, rather it works on protocol level.

PROCEDURE(for performing the load test(performance test))

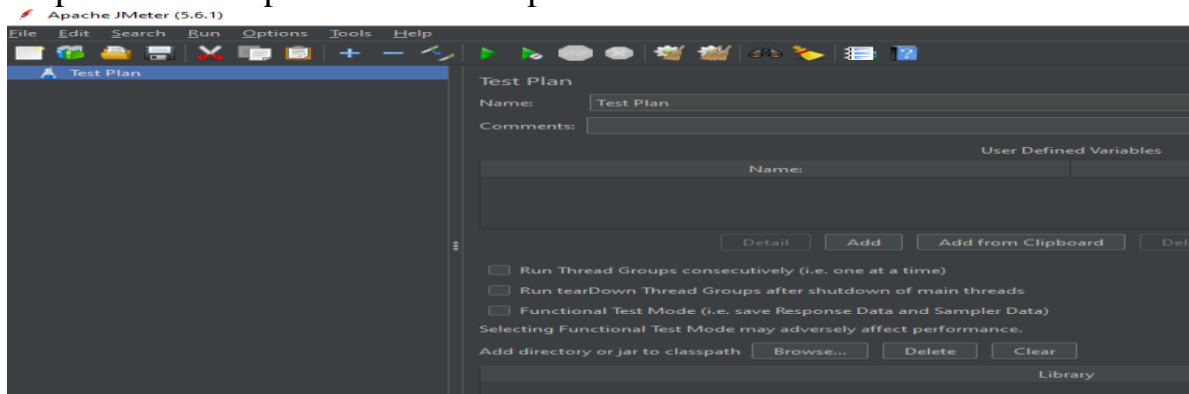
Step-1- download the apache jmeter from the official website.

Step-2-Extract the zip file to your parent directory.

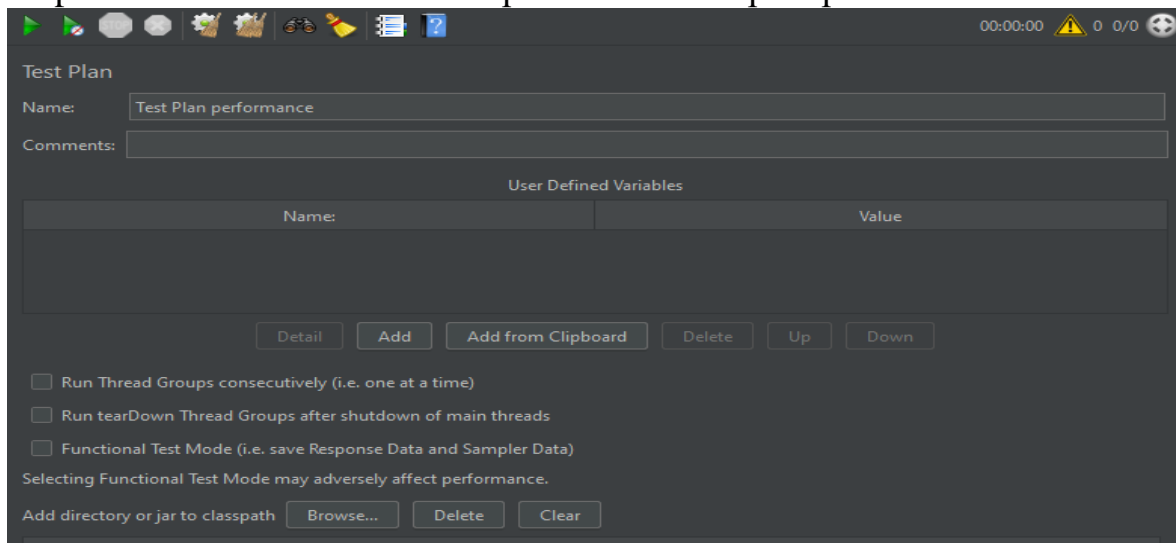
Step-3-double click on the Apachejmeter.jar executable jar file.

Name	Date modified	Type	Size
examples	5/5/2023 1:21 PM	File folder	
report-template	5/5/2023 1:21 PM	File folder	
templates	5/5/2023 1:21 PM	File folder	
ApacheJMeter	7/4/2023 11:03 AM	Executable Jar File	14 KB
BeanShellAssertion.bshrc	5/5/2023 1:21 PM	BSHRC File	2 KB
BeanShellFunction.bshrc	5/5/2023 1:21 PM	BSHRC File	3 KB
BeanShellListeners.bshrc	5/5/2023 1:21 PM	BSHRC File	2 KB
BeanShellSampler.bshrc	5/5/2023 1:21 PM	BSHRC File	3 KB
create-rmi-keystore	5/5/2023 1:21 PM	Windows Batch File	2 KB
create-rmi-keystore.sh	5/5/2023 1:21 PM	SH File	2 KB
hc.parameters	5/5/2023 1:21 PM	PARAMETERS File	2 KB

Step-4-then the apache JMeter is opened as follows

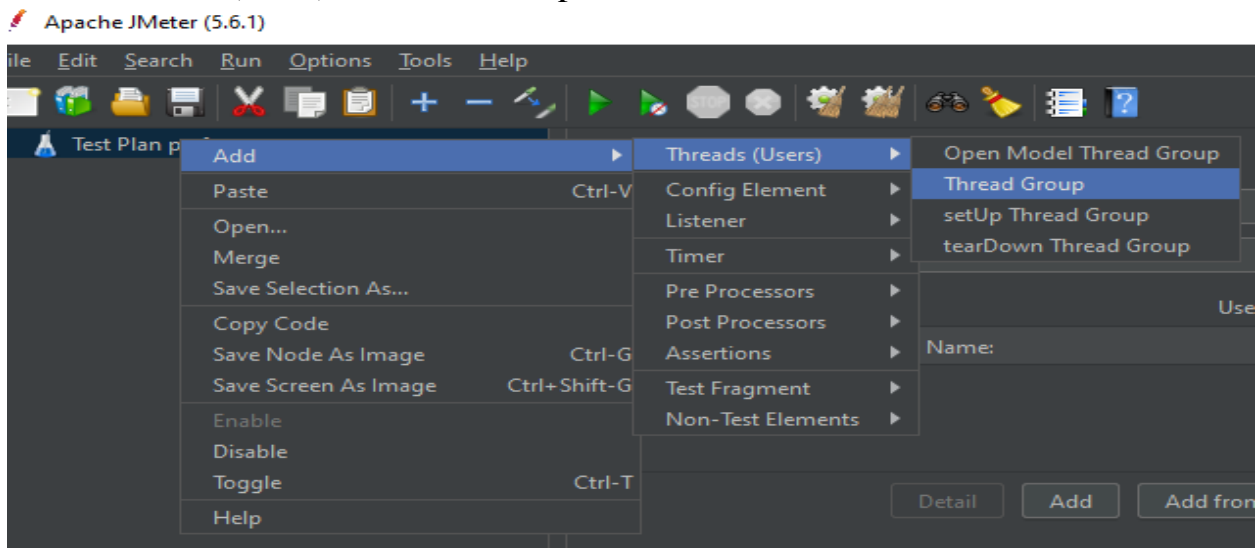


Step-5-enter the name of the test plan as the “test plan performance”

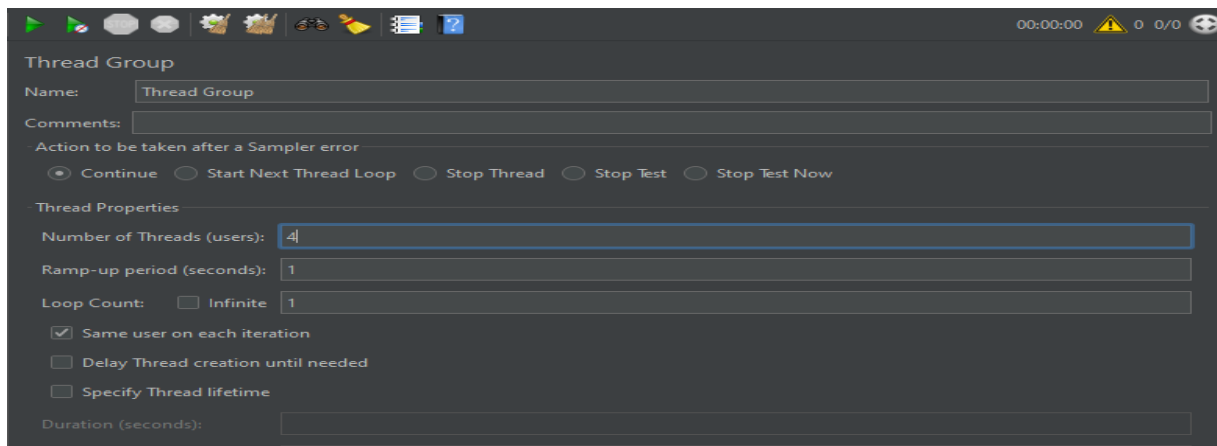


Step-6-then right click on the test plan and click on add to add users.

Add→Threads(users)→Thread Group

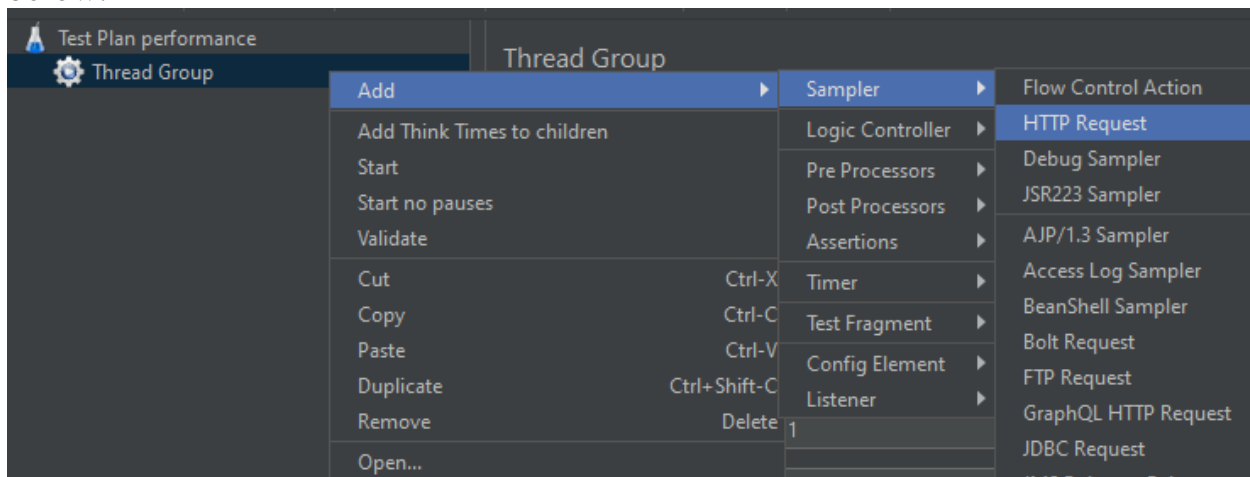


Step-7-then specify the number of threads and ramp-time as shown below.

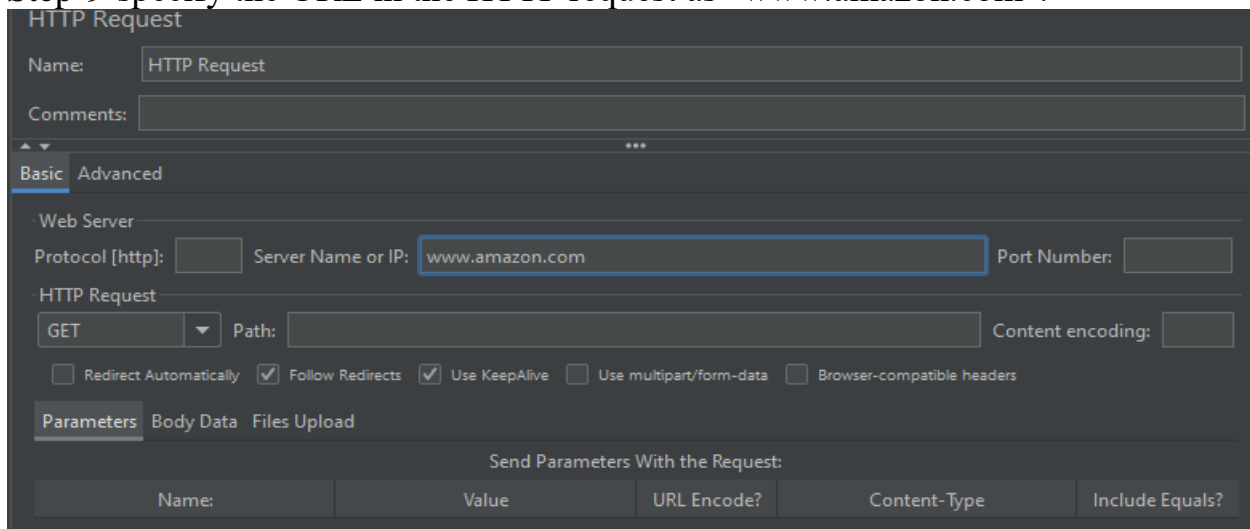


The screenshot shows the 'Thread Group' configuration window. The 'Name' field is set to 'Thread Group'. The 'Comments' field is empty. Under 'Action to be taken after a Sampler error', the 'Continue' radio button is selected. In the 'Thread Properties' section, 'Number of Threads (users)' is set to 4, 'Ramp-up period (seconds)' is set to 1, and 'Loop Count' is set to 1. The 'Same user on each iteration' checkbox is checked. The 'Duration (seconds)' field is empty.

Step-8-now right click on the thread and navigate to the HTTP request as shown below.

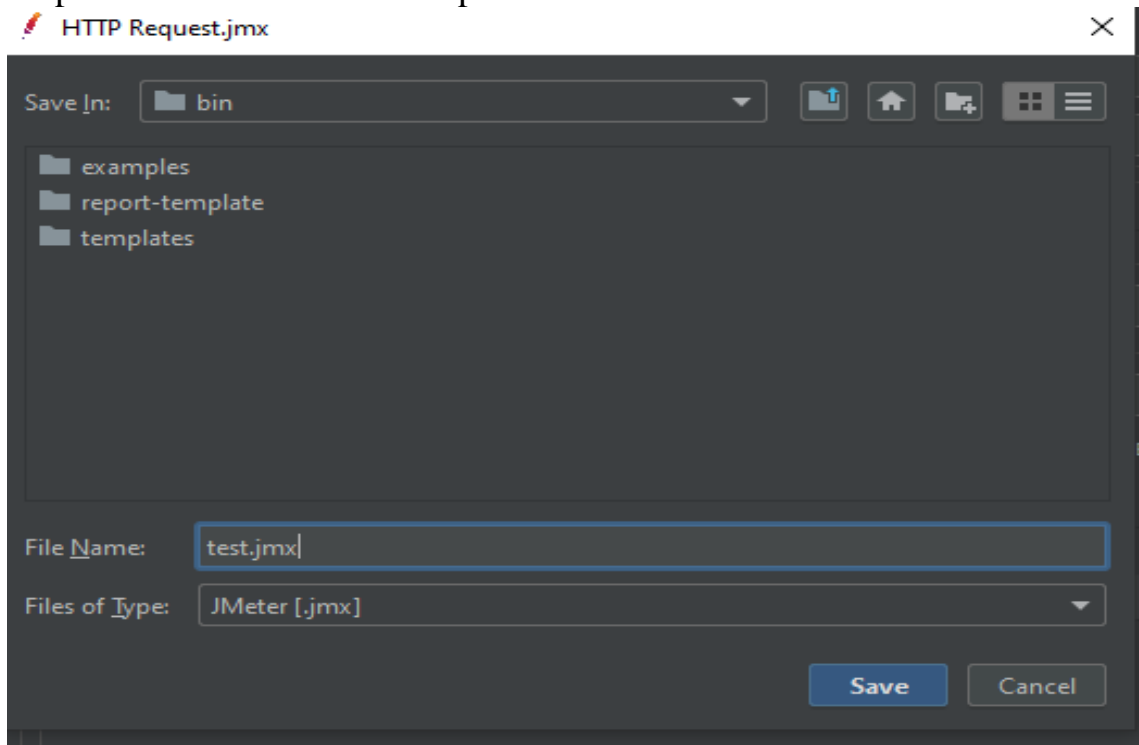


Step-9-specify the URL in the HTTP request as “www.amazon.com”.



The screenshot shows the 'HTTP Request' configuration window. The 'Name' field is set to 'HTTP Request'. The 'Comments' field is empty. The 'Basic' tab is selected. Under 'Web Server', 'Protocol [http:]' is set to 'http', 'Server Name or IP' is set to 'www.amazon.com', and 'Port Number' is empty. Under 'HTTP Request', the method is set to 'GET' and 'Path' is empty. The 'Content encoding' field is empty. The 'Redirect Automatically' checkbox is unchecked, and the 'Follow Redirects', 'Use KeepAlive', 'Use multipart/form-data', and 'Browser-compatible headers' checkboxes are checked. The 'Parameters' tab is selected. The 'Send Parameters With the Request' section shows a table with columns: Name, Value, URL Encode?, Content-Type, and Include Equals?.

Step-10-now save the test script as shown below.



Step-11-inoreder to execute the test script open the cmd in the bin directory of the jmeter.

```
C:\Windows\System32\cmd.exe

Microsoft Windows [Version 10.0.19045.4046]
(c) Microsoft Corporation. All rights reserved.

C:\apache-jmeter-5.6.1\bin>
```

Step-12-type the following command to execute the test.

```
jmeter -n -t C:\apache-jmeter-5.6.1\bin\test.jmx -l C:\apache-jmeter-5.6.1\bin\summarydata1.csv
```


The above runs the script and stores the summary of the execution in the file csv file summarydata.csv.

The command is runs as follows

```
C:\apache-jmeter-5.6.1\bin>jmeter -n -t C:\apache-jmeter-5.6.1\bin\test.jmx -l C:\apache-jmeter-5.6.1\bin\summarydata1.csv
WARN StatusConsoleListener The use of package scanning to locate plugins is deprecated and will be removed in a future release
WARN StatusConsoleListener The use of package scanning to locate plugins is deprecated and will be removed in a future release
WARN StatusConsoleListener The use of package scanning to locate plugins is deprecated and will be removed in a future release
WARN StatusConsoleListener The use of package scanning to locate plugins is deprecated and will be removed in a future release
Creating summariser <summary>
Created the tree successfully using C:\apache-jmeter-5.6.1\bin\test.jmx
Starting standalone test @ April 22, 2024 12:05:57 AM IST (1713724557243)
Waiting for possible Shutdown/StopTestNow/HeapDump/ThreadDump message on port 4445
summary + 1 in 00:00:06 = 0.2/s Avg: 3349 Min: 3349 Max: 3349 Err: 1 (100.00%) Active: 10 Started: 10 Finished: 0
summary + 9 in 00:00:01 = 13.0/s Avg: 4858 Min: 3273 Max: 6415 Err: 9 (100.00%) Active: 0 Started: 10 Finished: 10
summary = 10 in 00:00:07 = 1.4/s Avg: 4707 Min: 3273 Max: 6415 Err: 10 (100.00%)
Tidying up ... @ April 22, 2024 12:06:05 AM IST (1713724565818)
... end of run
```

The contents of the summarydata1.csv is shown below.

timeStamp	elapsed	label	response	response	threadName	dataType	success	failureMe	bytes	sentBytes	grpThread	allThread
1.71E+12	3349	HTTP Req	503	Service Ur	Thread Gr	text	FALSE		7875	349	10	10
1.71E+12	584	HTTP Req	301	Moved Pe	Thread Gr	text	TRUE		351	115	10	10
1.71E+12	2253	HTTP Req	301	Moved Pe	Thread Gr	text	TRUE		373	115	10	10
1.71E+12	509	HTTP Req	503	Service Ur	Thread Gr	text	FALSE		7151	119	10	10
1.71E+12	3793	HTTP Req	503	Service Ur	Thread Gr	text	FALSE		7875	349	10	10
1.71E+12	632	HTTP Req	301	Moved Pe	Thread Gr	text	TRUE		351	115	10	10
1.71E+12	2631	HTTP Req	301	Moved Pe	Thread Gr	text	TRUE		373	115	10	10
1.71E+12	525	HTTP Req	503	Service Ur	Thread Gr	text	FALSE		7151	119	10	10
1.71E+12	4472	HTTP Req	503	Service Ur	Thread Gr	text	FALSE		7875	349	8	8
1.71E+12	664	HTTP Req	301	Moved Pe	Thread Gr	text	TRUE		351	115	8	8
1.71E+12	3262	HTTP Req	301	Moved Pe	Thread Gr	text	TRUE		373	115	8	8
1.71E+12	543	HTTP Req	503	Service Ur	Thread Gr	text	FALSE		7151	119	8	8
1.71E+12	5761	HTTP Req	503	Service Ur	Thread Gr	text	FALSE		7875	349	7	7
1.71E+12	778	HTTP Req	301	Moved Pe	Thread Gr	text	TRUE		351	115	7	7
1.71E+12	4461	HTTP Req	301	Moved Pe	Thread Gr	text	TRUE		373	115	7	7
1.71E+12	490	HTTP Req	503	Service Ur	Thread Gr	text	FALSE		7151	119	7	7
1.71E+12	3588	HTTP Req	503	Service Ur	Thread Gr	text	FALSE		7875	349	6	6
1.71E+12	667	HTTP Req	301	Moved Pe	Thread Gr	text	TRUE		351	115	6	6
1.71E+12	1835	HTTP Req	301	Moved Pe	Thread Gr	text	TRUE		373	115	6	6
1.71E+12	1082	HTTP Req	503	Service Ur	Thread Gr	text	FALSE		7151	119	6	6
1.71E+12	5561	HTTP Req	503	Service Ur	Thread Gr	text	FALSE		7875	349	6	6
1.71E+12	665	HTTP Req	301	Moved Pe	Thread Gr	text	TRUE		351	115	6	6

RESULT

Thus the performance of the e-commerce application was tested successfully.

**EXNO:7 AUTOMATE THE TESTING OF E-COMMERCE APPLICATION
DATE: USING THE SELENIUM.**

AIM:

To Automate The Testing Of E-Commerce Application Using The Selenium.

SELENIUM TOOL:

Selenium is an open-source, automated testing tool used to test web applications across various browsers. Selenium can only test web applications, unfortunately, so desktop and mobile apps can't be tested. However, other tools like Appium and HP's QTP can be used to test software and mobile applications.



ECLIPSE IDE:

Package Description:

The essential tools for any Java developer, including a Java IDE, a CVS client, Git client, XML Editor, Mylyn, Maven integration and WindowBuilder.



This package includes:

- Code Recommenders Developer Tools
- Eclipse Git Team Provider
- Eclipse Java Development Tools
- Maven Integration for Eclipse
- Mylyn Task List
- WindowBuilder Core
- Eclipse XML Editors and Tools

PROCEDURE:

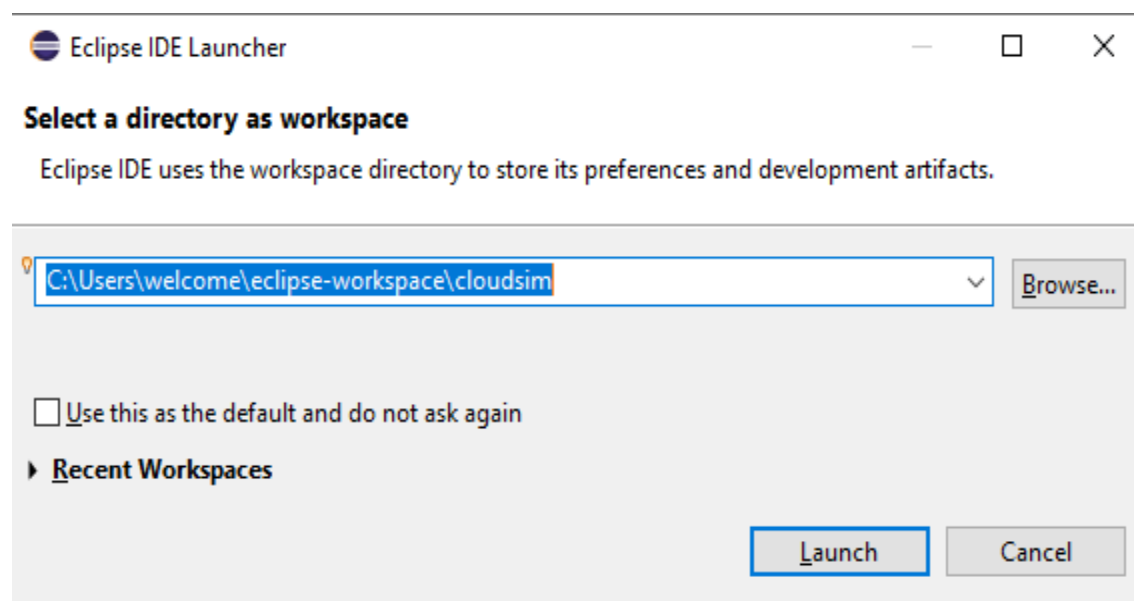
- 1.configuring the selenium external libraries in the eclipse project.
- 2.Developing the test case.
- 3.Executing the test case using the web drivers.

1.CONFIGURING THE SELENIUM EXTERNAL LIBRARIES IN THE ECLIPSE PROJECT

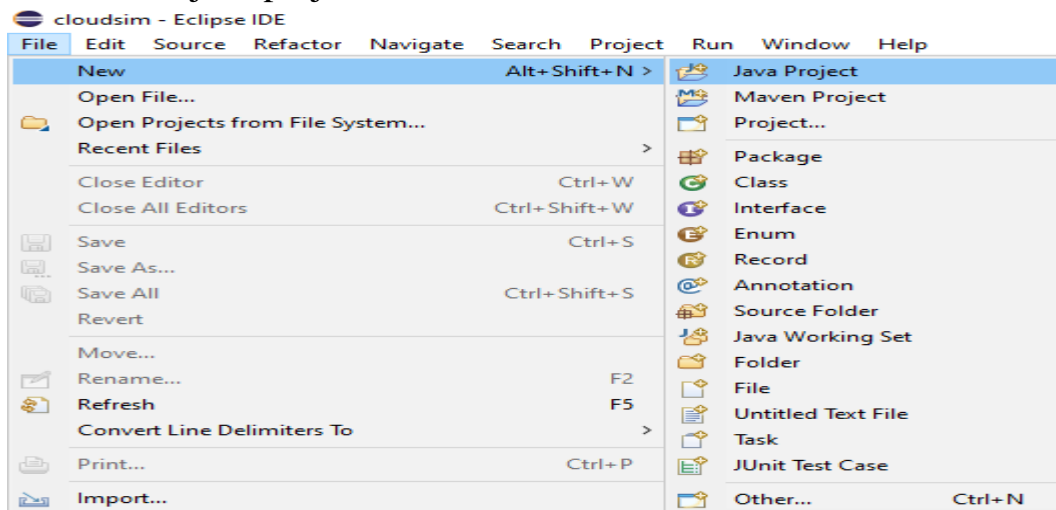
STEP1-(a)-download the selenium latest version from the official website.

STEP1-(b)-extract the selenium Winrar into your directory.

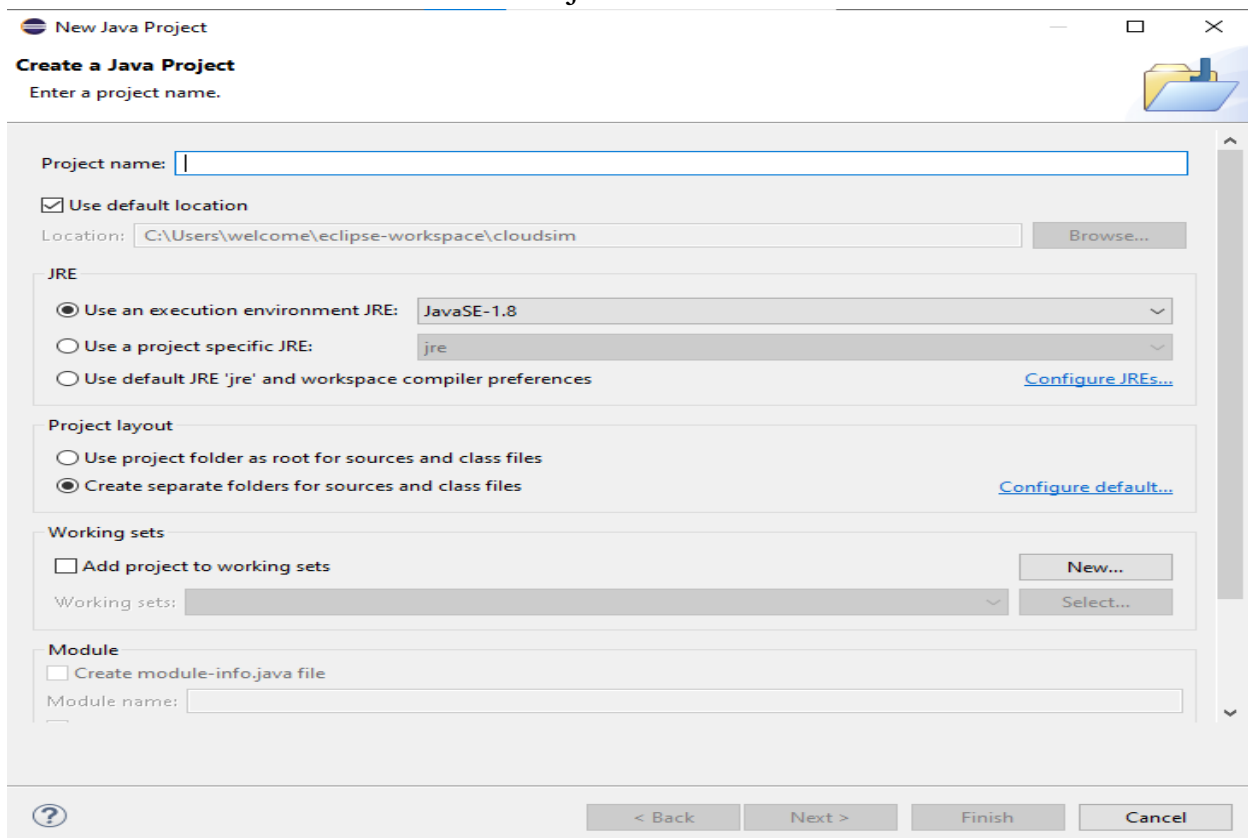
STEP1-(c)-launch the eclipse.



STEP1-(d)-then click on the new “file”,under this click on the “new”,under the new click on the “java project”.
file→new→java project.



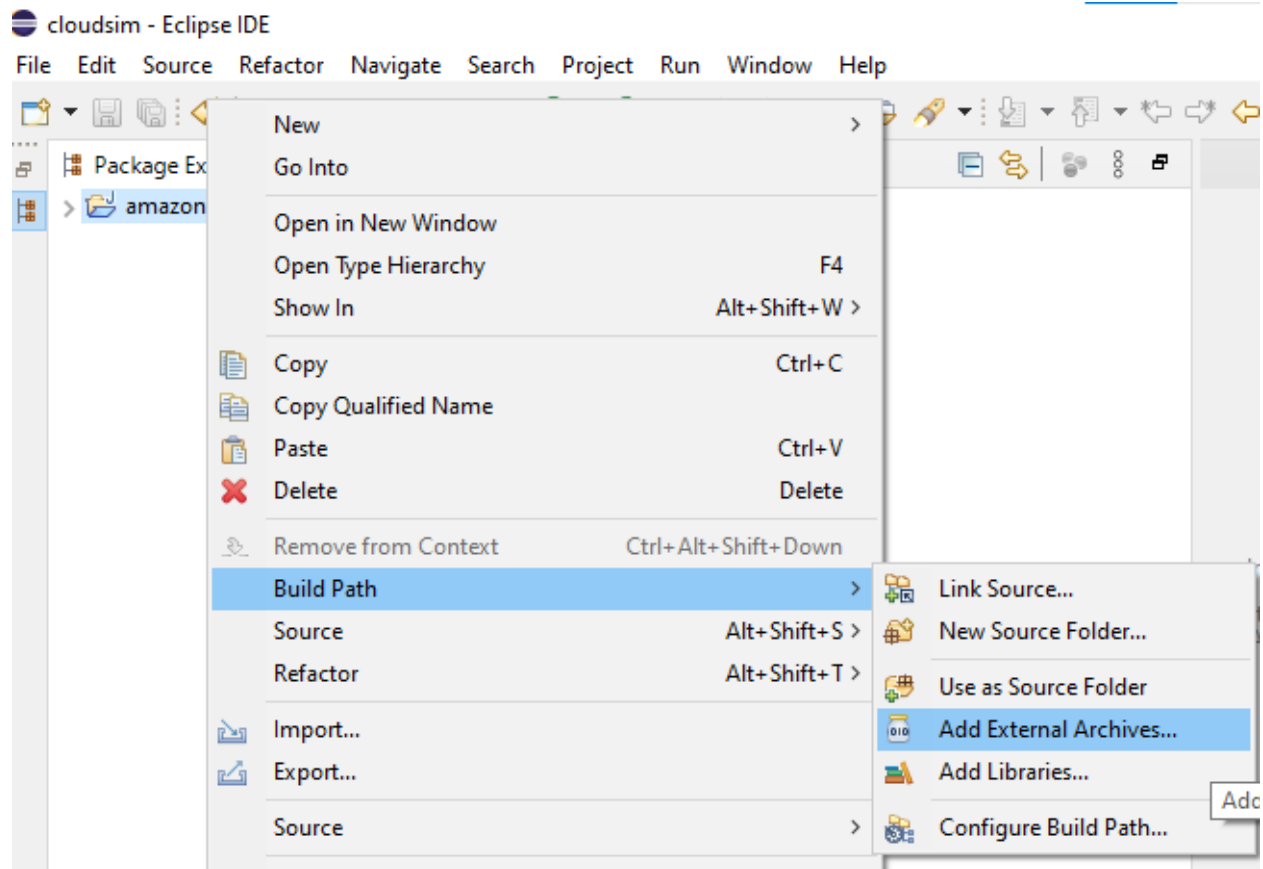
STEP1-(e)-after clicking on the “java project”.enter the name of the java project and set the execution environment as java S.E 1.8.



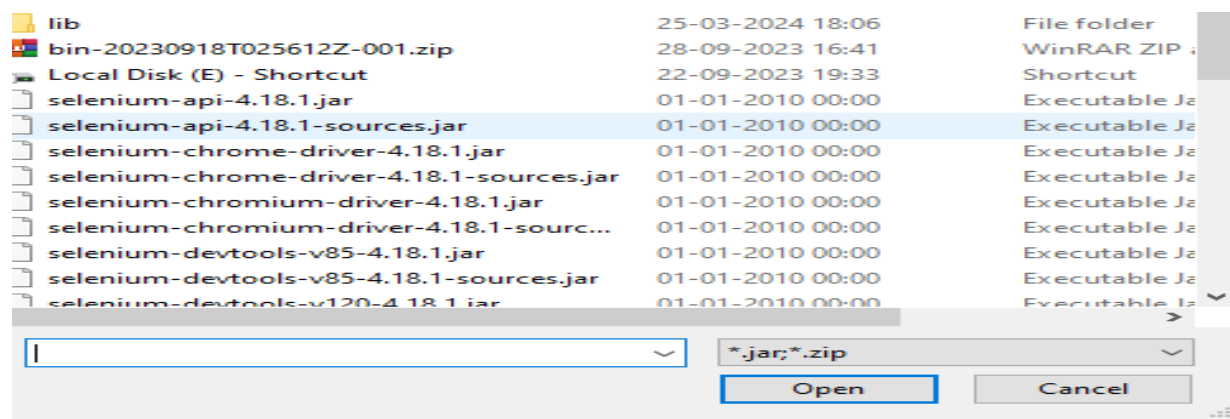
Then click on the finish button.

STEP1-(f) then right click on the project which is created on the project explorer. now click on the “build path”.under this click on the “add external archives”.

Build path→Add External Archives



STEP1-(g)- In the “add external archives“ add all the necessary jars of the selenium that you previously extracted in your directory as shown below.

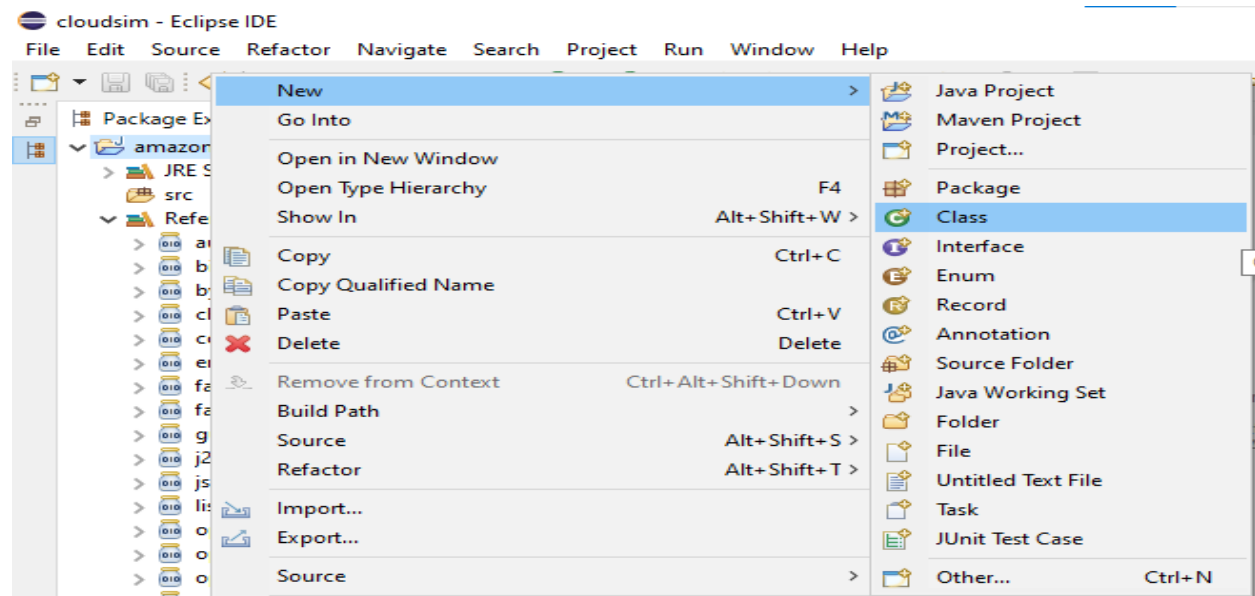


Select all the jars and then click on open to add selenium jars.now the selenium was successfully configured in the eclipse.

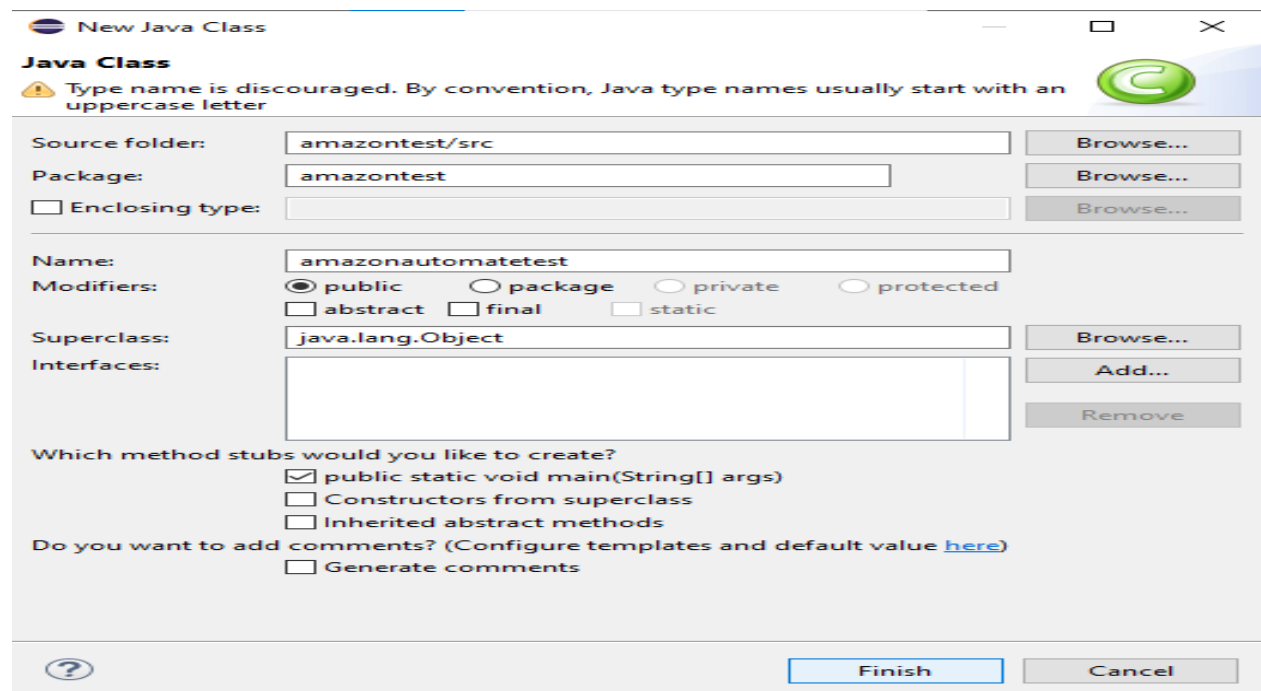
2.DEVELOPING THE TEST CASE.

STEP2-(a)-right click on the project in which the selenium was configured.under this click on the new class and create the new class.

New→Class



STEP2-(b)-name the class as the amazonautomatetest and make it as the main class.



Then click on the finish button.

The java code for testing the searchbox of the e-commerce application(amazon.in) is given below.

```
package amazontest;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
class amazonautomatetest{
    public static void main(String args[])
    {
        System.setProperty("webdriver.chrome.driver","E:\\chromedriver-
win64\\chromedriver.exe");
        WebDriver driver=new ChromeDriver();
        driver.get("http://www.amazon.in/");
        WebElement searchbox=driver.findElement(By.id("twotabsearchtextbox"));
        searchbox.sendKeys("watch");
        searchbox.submit();
    }
}
```

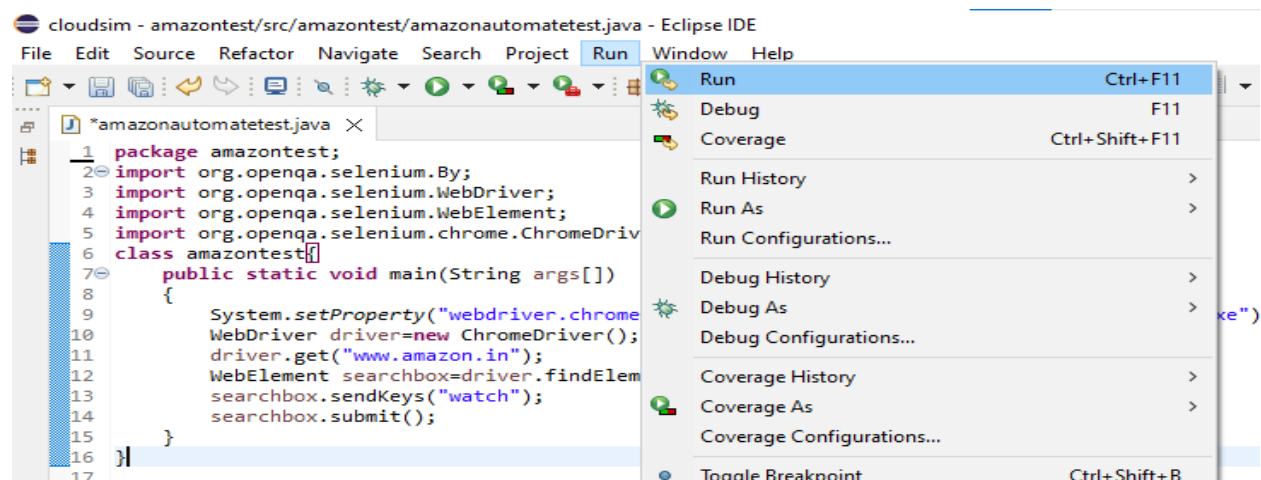
The above code is used to check whether the search box is working correctly or not by determining whether it shows the products which are related to which entered in the searchbox.

Now the test case was successfully developed using the java.

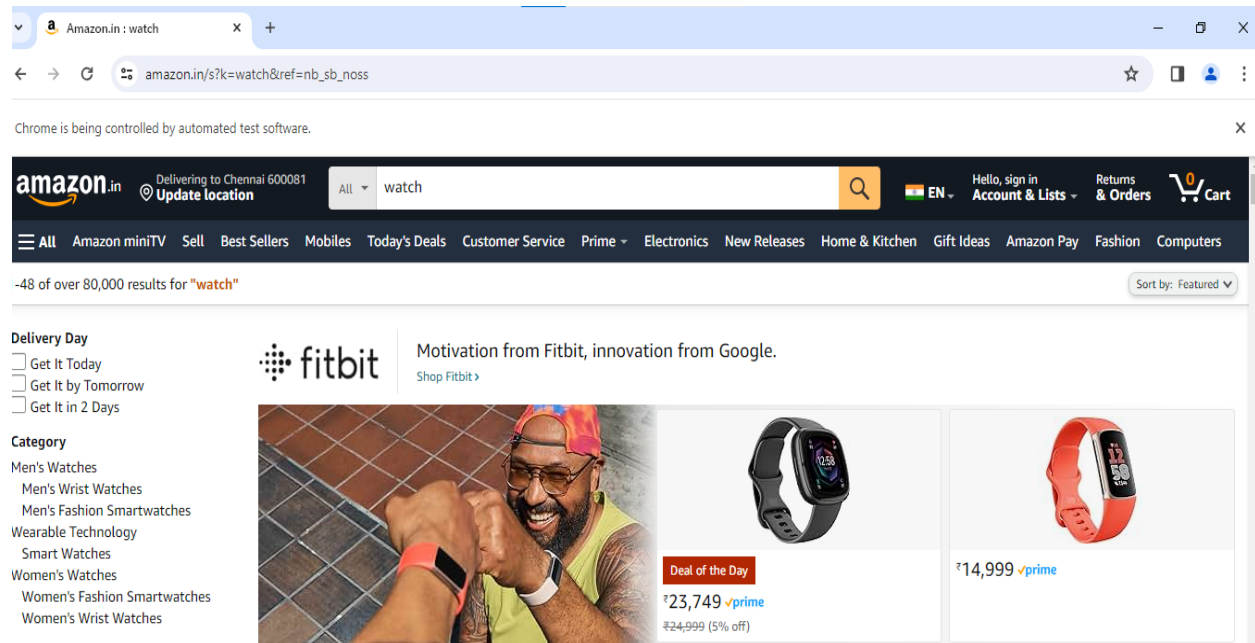
3.EXECUTING THE TEST CASE USING THE WEB DRIVERS.

STEP3-(a)-before executing the test case.check your chrome version if it is not updated then update it first and then download the chrome driver which corresponds to the updated version of the chrome browser.then pass the path of the chromedriver as the second parameter to the method setProperty("par1","par2").

STEP3-(b)-now execute the test by clicking on Run in eclipse.



OUTPUT OF THE DEVELOPED TEST CASE:



Thus the test was successfully executed and the search box of the amazon is working correctly.

RESULT:

Thus, The Testing Of E-Commerce Application Using The Selenium was automated successfully and output was verified.

EXPNO:8
DATE:

INTEGRATE THE TESTNG WITH THE ABOVE TEST AUTOMATION.

AIM

To integrate the testng with the above test automation.

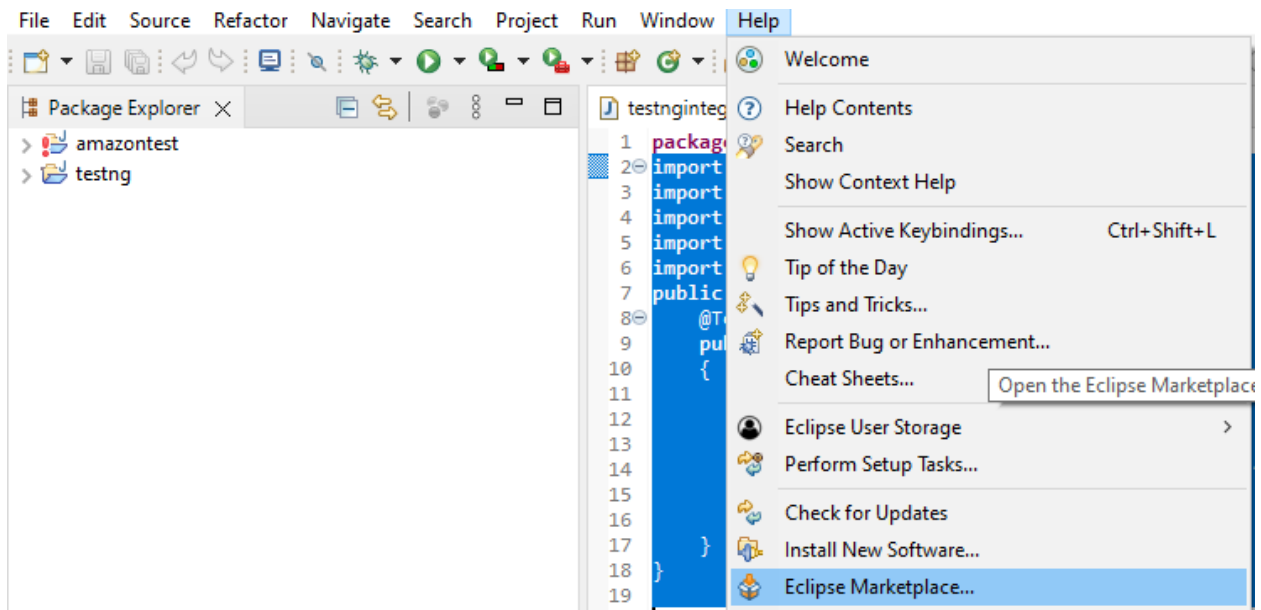
PROCEDURE

- step-(1)-install the TestNg in the eclipse IDE.**
- step-(2)-configuring the selenium external libraries in the eclipse project.**
- step-(3)-Developing the test case by integrating TestNg with selenium.**
- step-(4)-Executing the test case using the web drivers.**

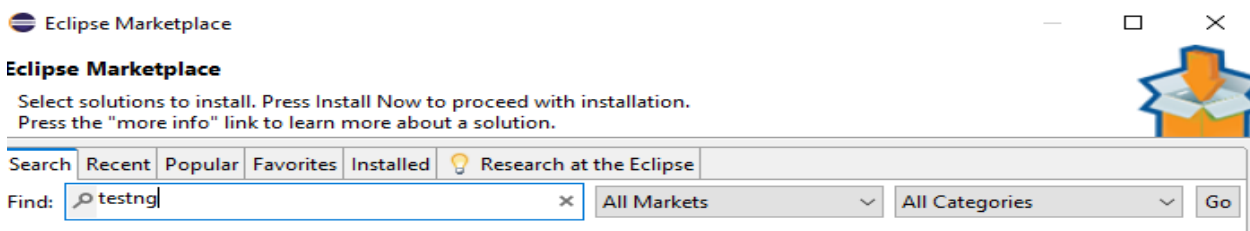
INSTALL THE TESTNG IN THE ECLIPSE IDE:

Step1-(a)-launch the eclipse.

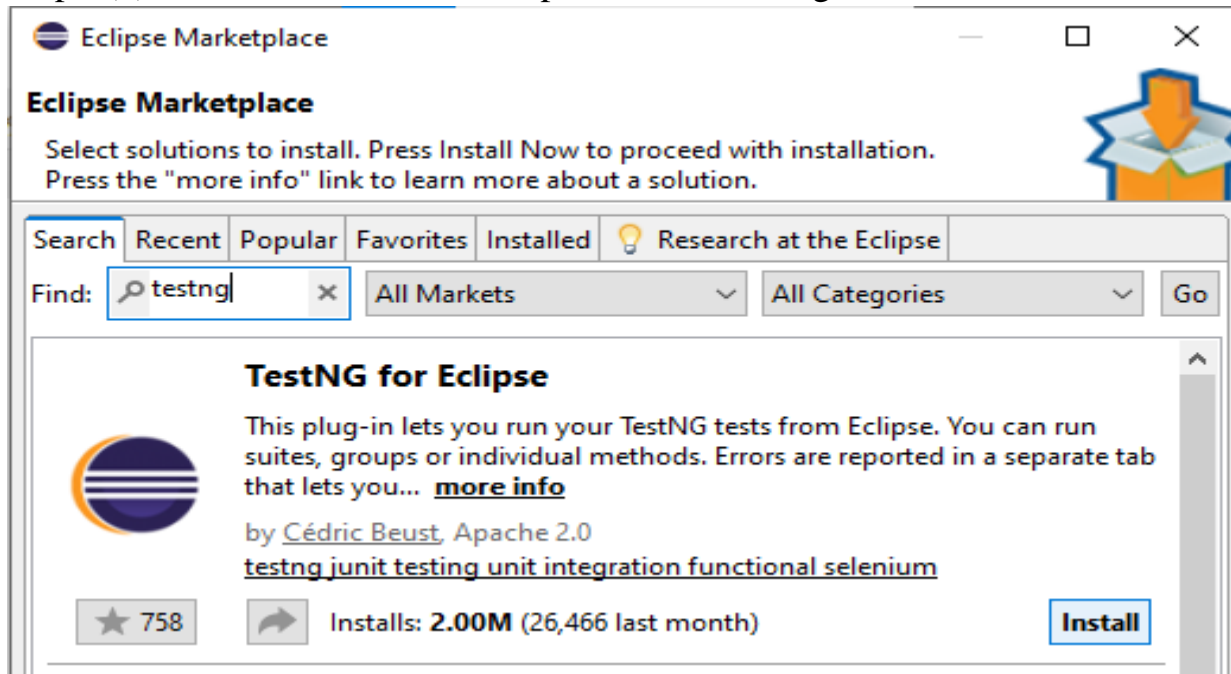
Step1-(b)-click on the help under the help click on the eclipse market place
help→eclipse market place



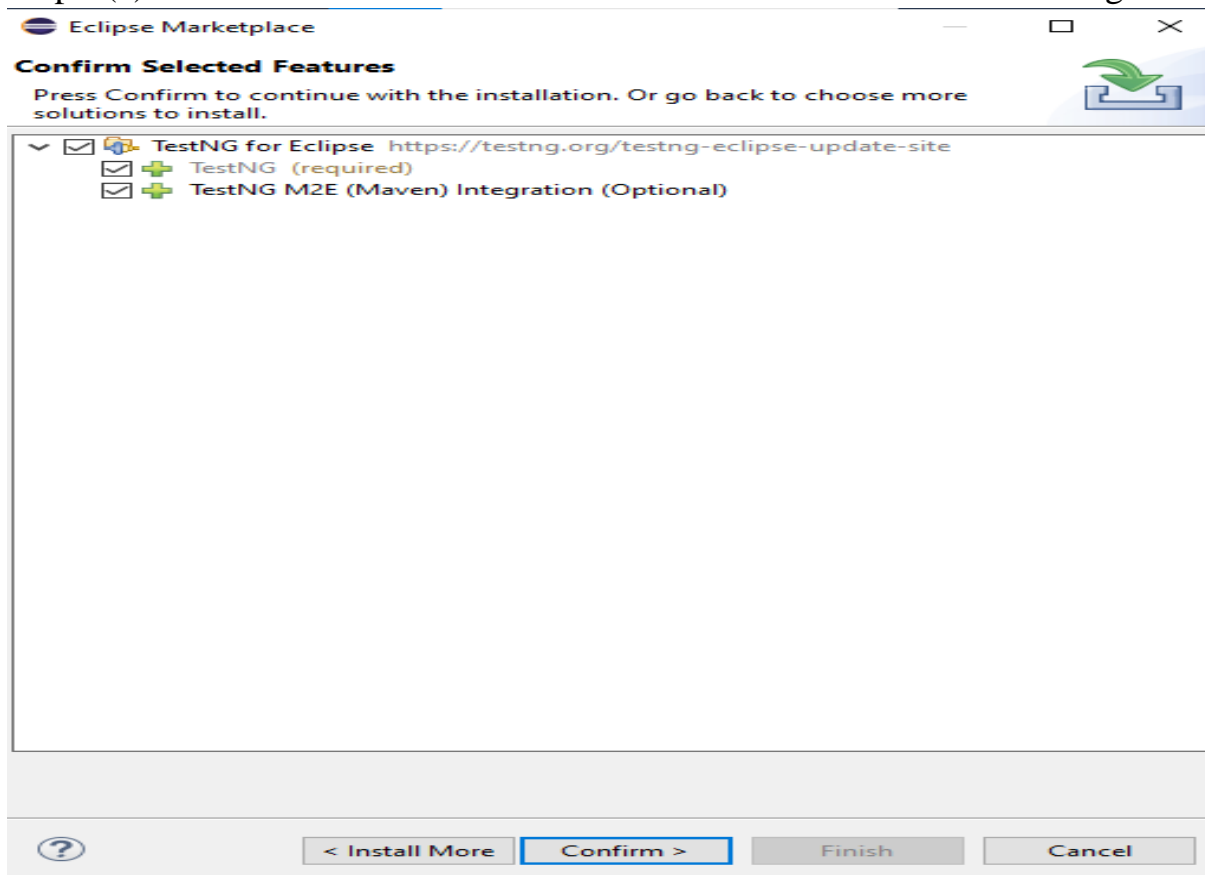
Step1-(c)-Then you can able to view the following window.type testng in find.



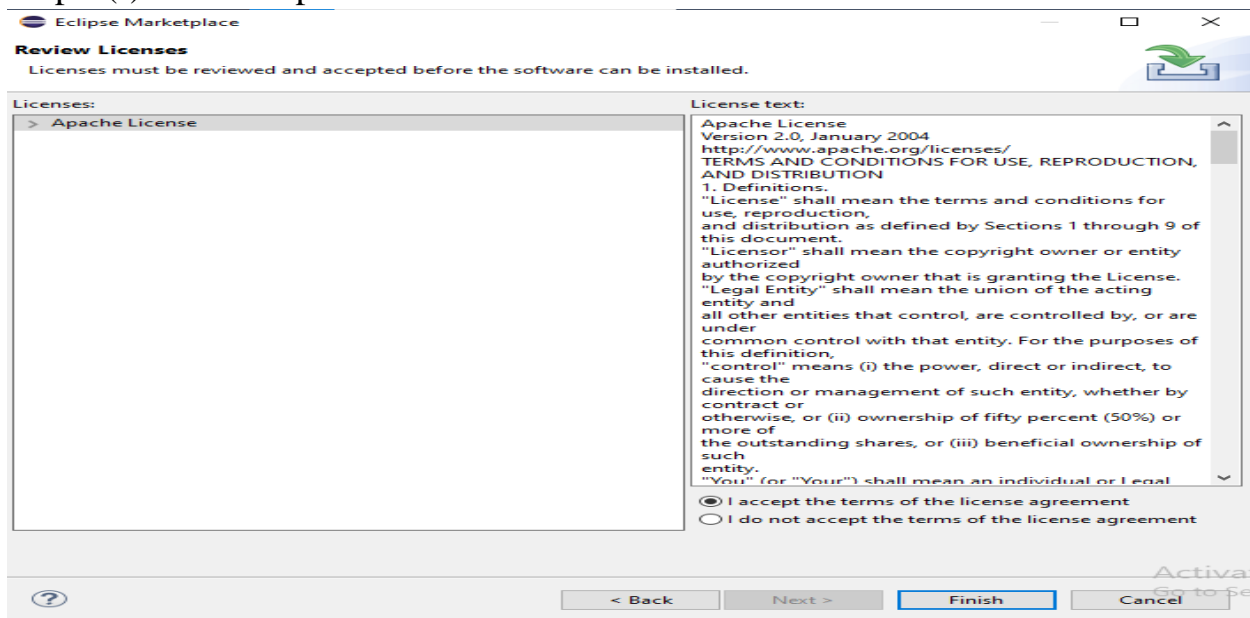
Step1-(d)-then click on the install option in the TestNg tool.



Step1-(e)-click on the conform button to conform the installation of testNg.

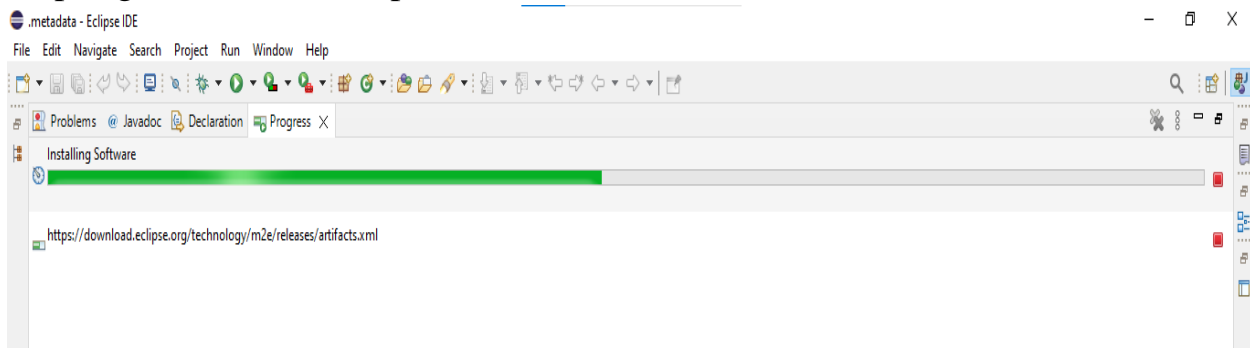


Step1-(f)-then accept the terms and condition.

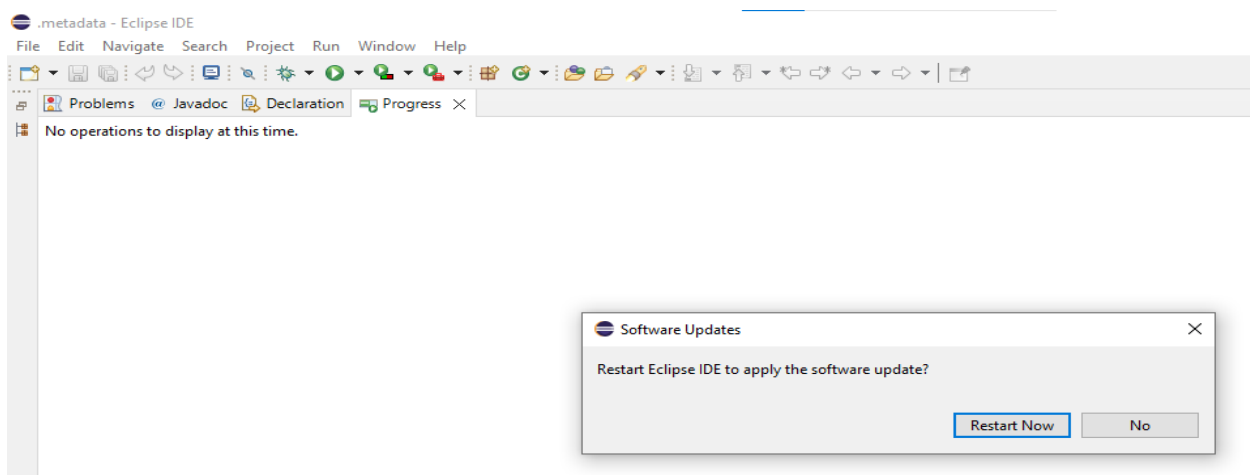


Click on finish to proceed further.

Step1-(g)-the installation process is carried out as shown below.



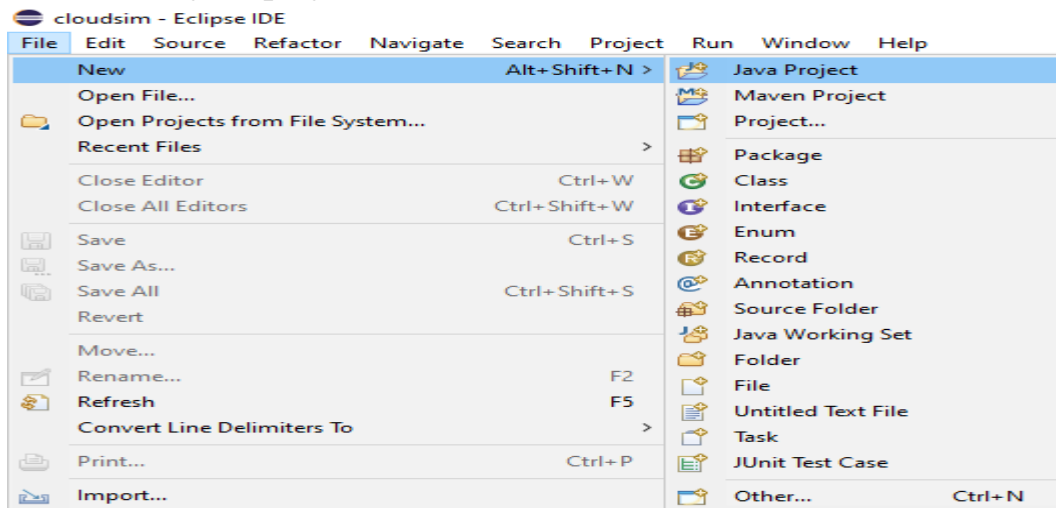
Step1-(h)-after the completion of installation it asks for the restart and then click restart.



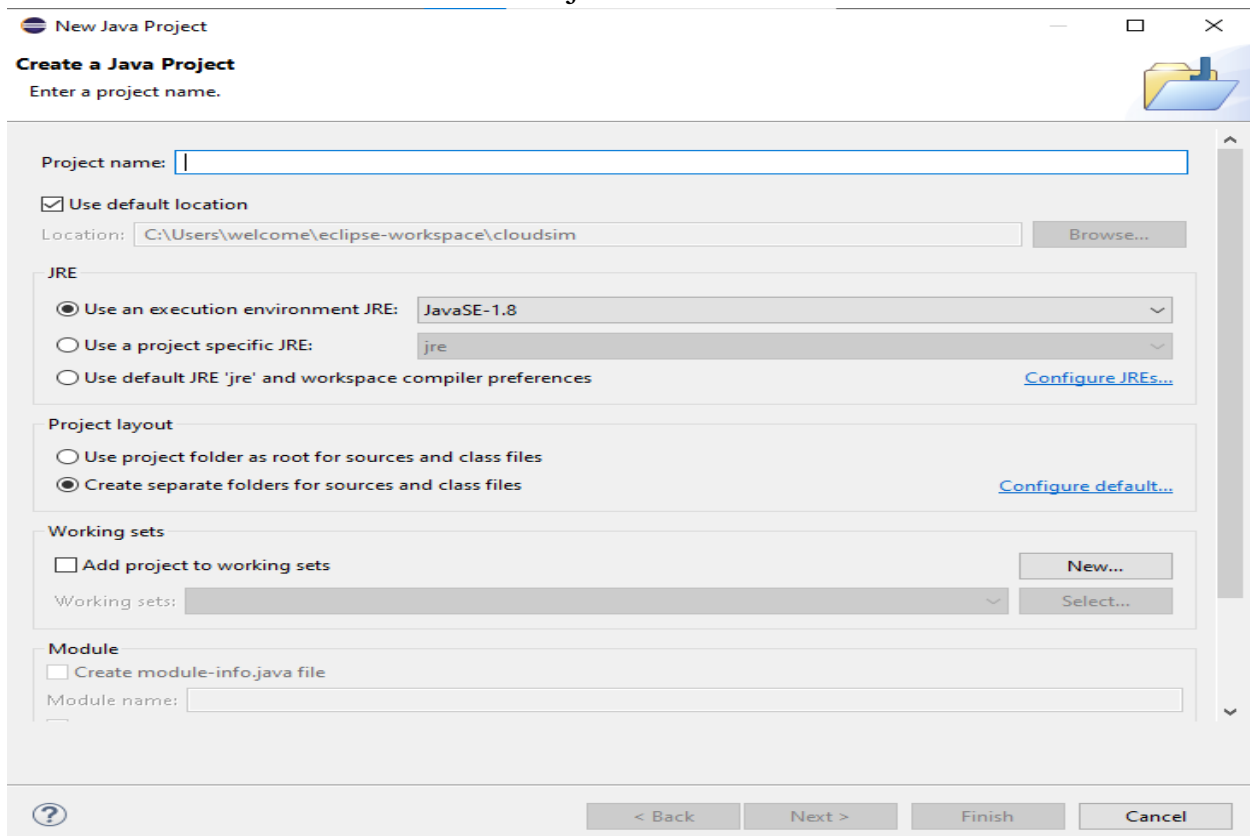
CONFIGURING THE SELENIUM EXTERNAL LIBRARIES IN THE ECLIPSE PROJECT.

STEP2-(a)-then click on the new “file”,under this click on the “new”,under the new click on the “java project”.

file→new→java project.



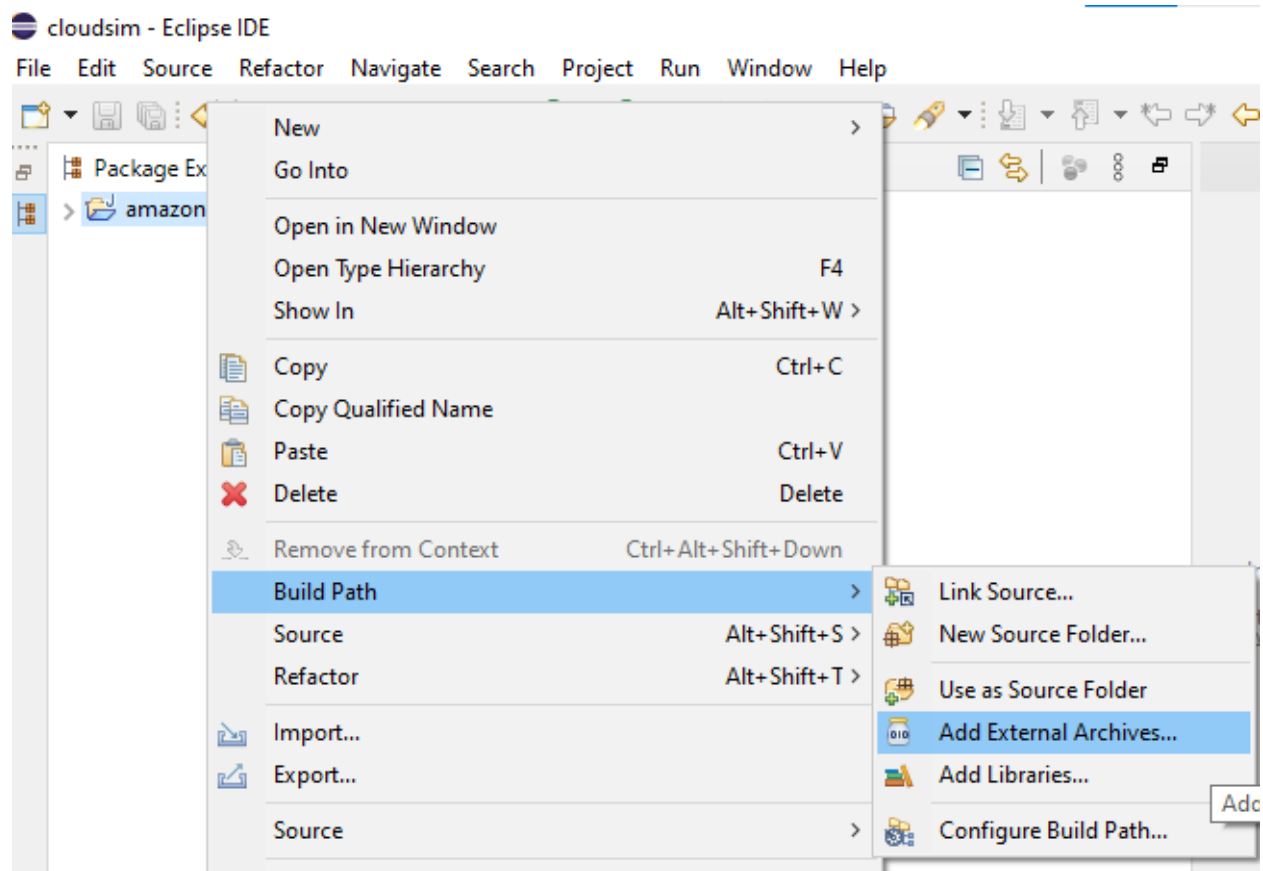
STEP2-(b)-after clicking on the “java project”.enter the name of the java project and set the execution environment as java S.E 1.8.



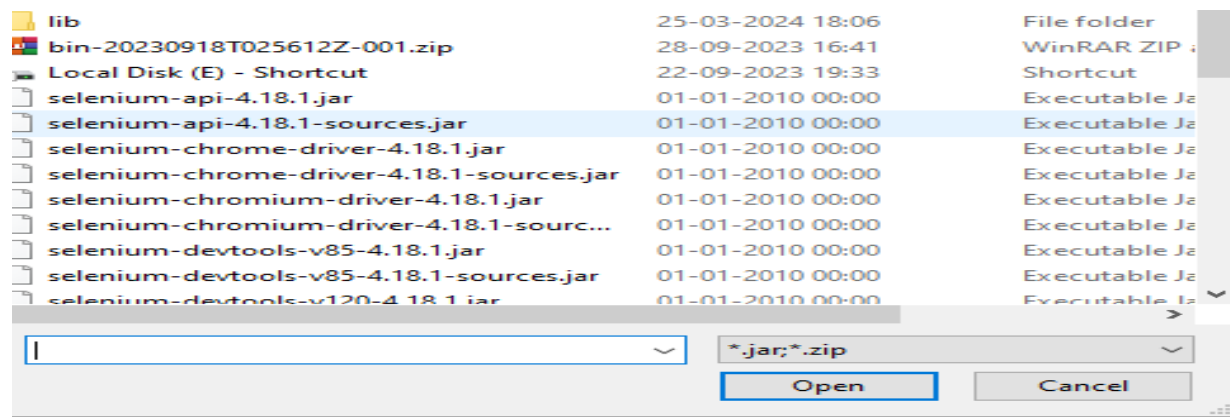
Then click on the finish button.

STEP2-(c) then right click on the project which is created on the project explorer. now click on the “build path”.under this click on the “add external archives”.

Build path→Add External Archives



STEP2-(d)- In the “add external archives“ add all the necessary jars of the selenium that you previously extracted in your directory as shown below.

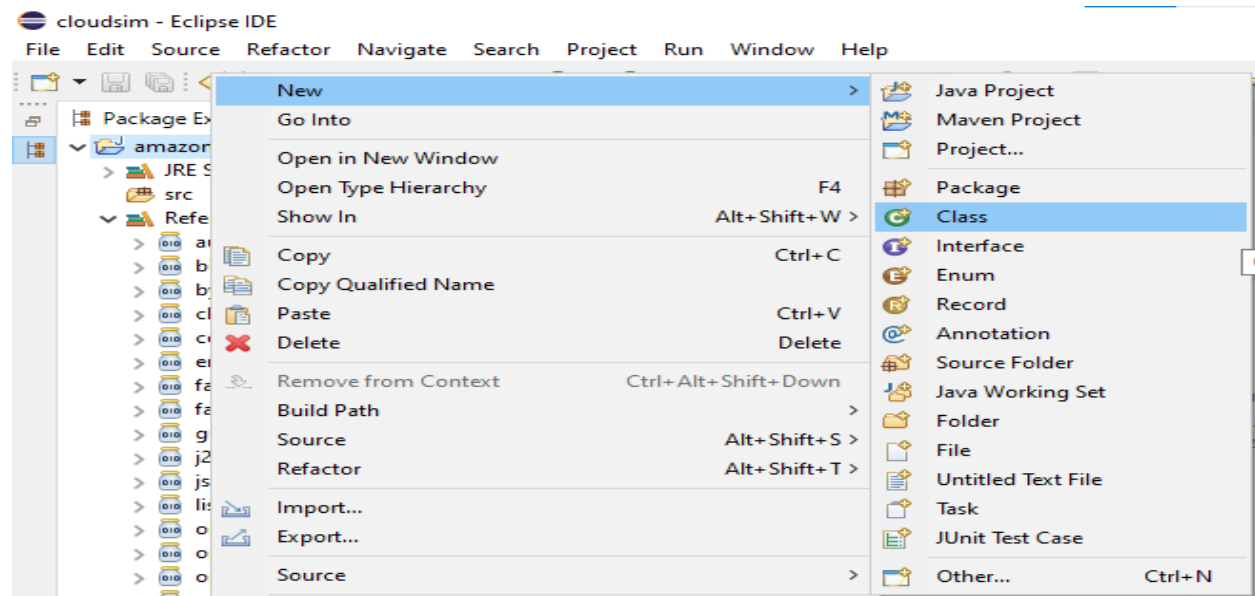


Select all the jars and then click on open to add selenium jars.now the selenium was successfully configured in the eclipse.

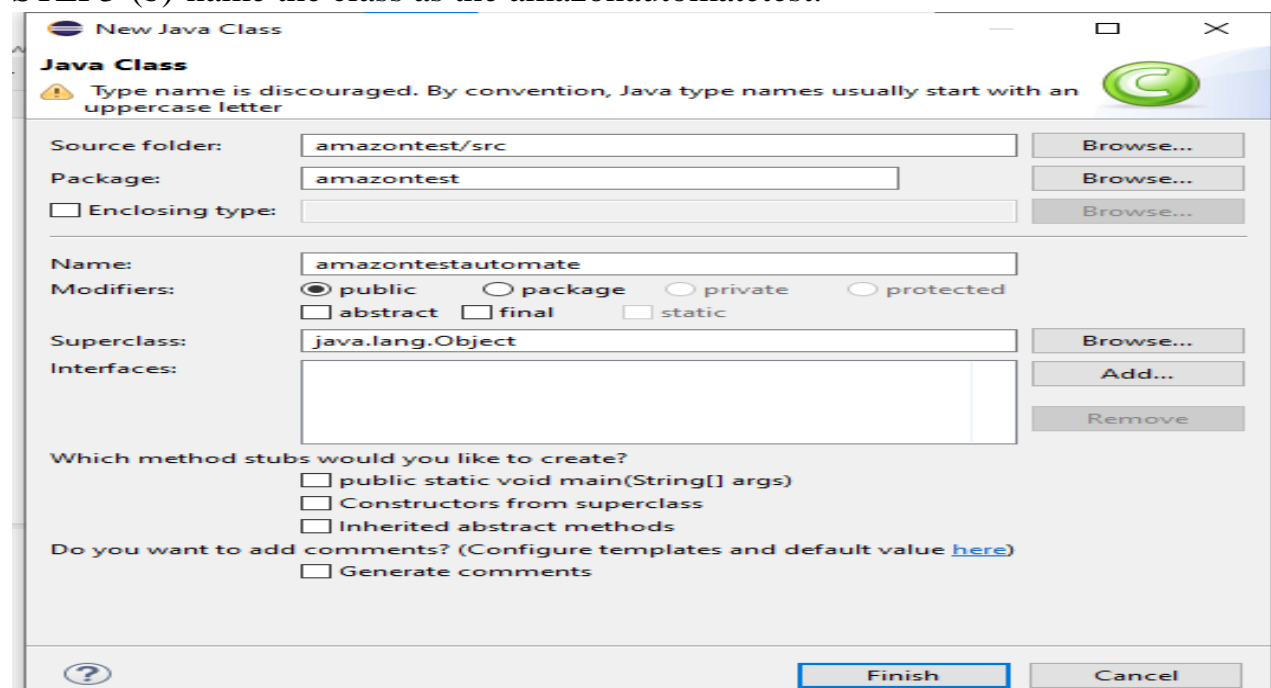
DEVELOPING THE TEST CASE BY INTEGRATING TESTNG WITH SELENIUM.

STEP3-(a)-right click on the project in which the selenium was configured.under this click on the new class and create the new class.

New→Class



STEP3-(b)-name the class as the amazonautomatetest.



Then click on the finish button.

The java code for testing the searchbox of the e-commerce application(amazon.in) is given below.

```
package amazontest;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.testng.annotations.Test;
public class amazontestautomate{
    @Test
    public static void test()
    {
        System.setProperty("webdriver.chrome.driver","E:\\chromedriver-
win64\\chromedriver.exe");
        WebDriver driver=new ChromeDriver();
        driver.get("http://www.amazon.in/");
        WebElement searchbox=driver.findElement(By.id("twotabsearchtextbox"));
        searchbox.sendKeys("led tv");
        searchbox.submit();
    }
}
```

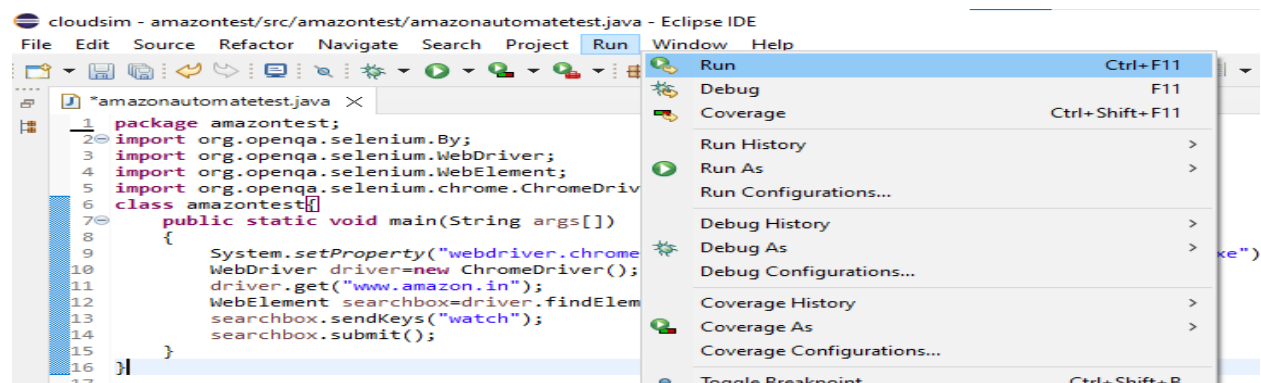
The above code is used to check whether the search box is working correctly or not by determining whether it shows the products which are related to which entered in the searchbox.add the testNg library by right ckick on the project and navigate as follows. build path→add libraries→testNg.

Now the test case was successfully developed using the java.

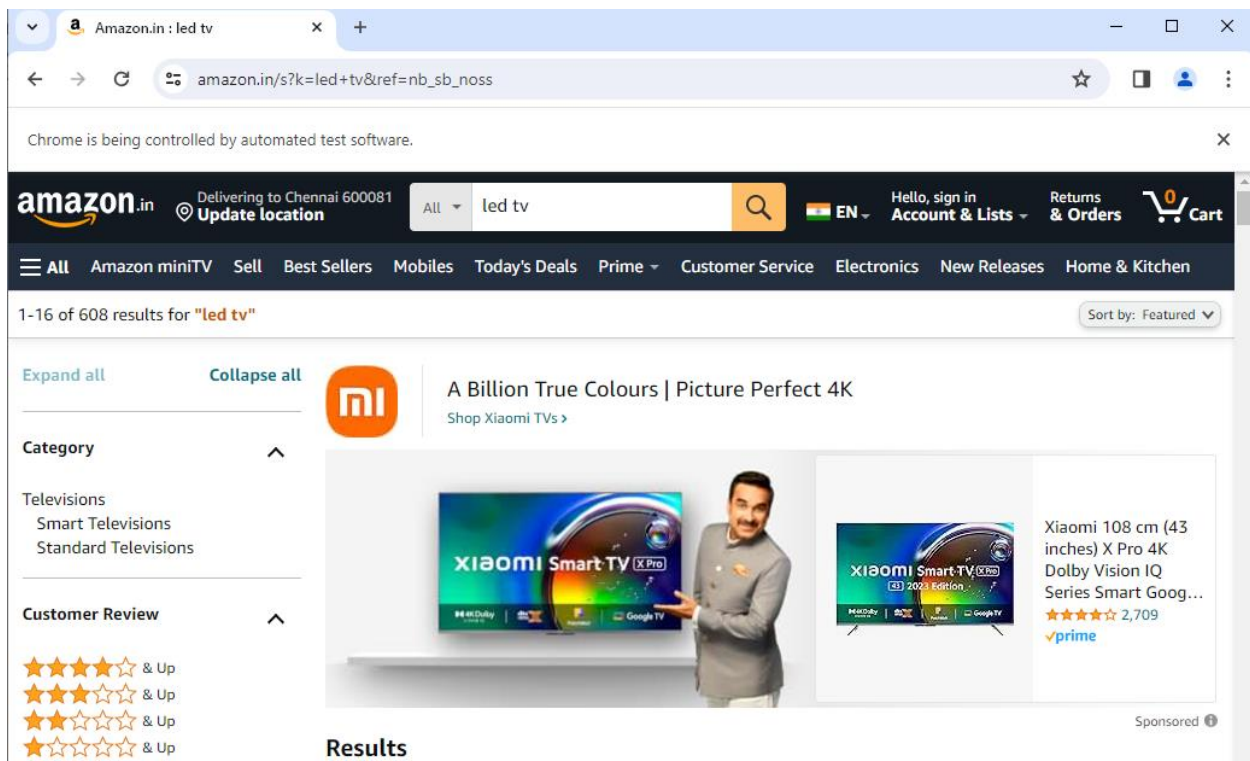
4.EXECUTING THE TEST CASE USING THE WEB DRIVERS.

STEP4-(a)-before executing the test case.check your chrome version if it is not updated then update it first and then download the chrome driver which corresponds to the updated version of the chrome browser.then pass the path of the chromedriver as the second parameter to the method setProperty("par1","par2").

STEP4-(b)-now execute the test by clicking on Run in eclipse.



OUTPUT OF THE DEVELOPED TEST CASE:



Thus the test was successfully executed and the search box of the amazon is working correctly.

The result of the test is shown below.

```
.metadata - amazontest/src/amazontest/amazontestautomate.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help

<terminated> amazontestautomate [TestNG] C:\eclipse\plugins\org.eclipse.justj.openj
[[RemoteTestNG] detected TestNG version 7.4.0
Apr 05, 2024 6:23:03 AM org.openqa.selenium.devtools.CdpVersionFi
WARNING: Unable to find an exact match for CDP version 123, retur
PASSED: test

=====
Default test
Tests run: 1, Failures: 0, Skips: 0
=====

Default suite
Total tests run: 1, Passes: 1, Failures: 0, Skips: 0
=====
```

RESULT

Thus the TestNg was successfully integrated with the selenium automation test tool.

EXP NO:9

DEVELOP A MINI PROJECT

DATE: a)build the data driven frame work using selenium and testng

AIM

To Build The Data Driven Frame Work Using The Selenium And Testng.

PROCEDURE

1.configuring the selenium external libraries in the eclipse project.

2.Developing the test case.

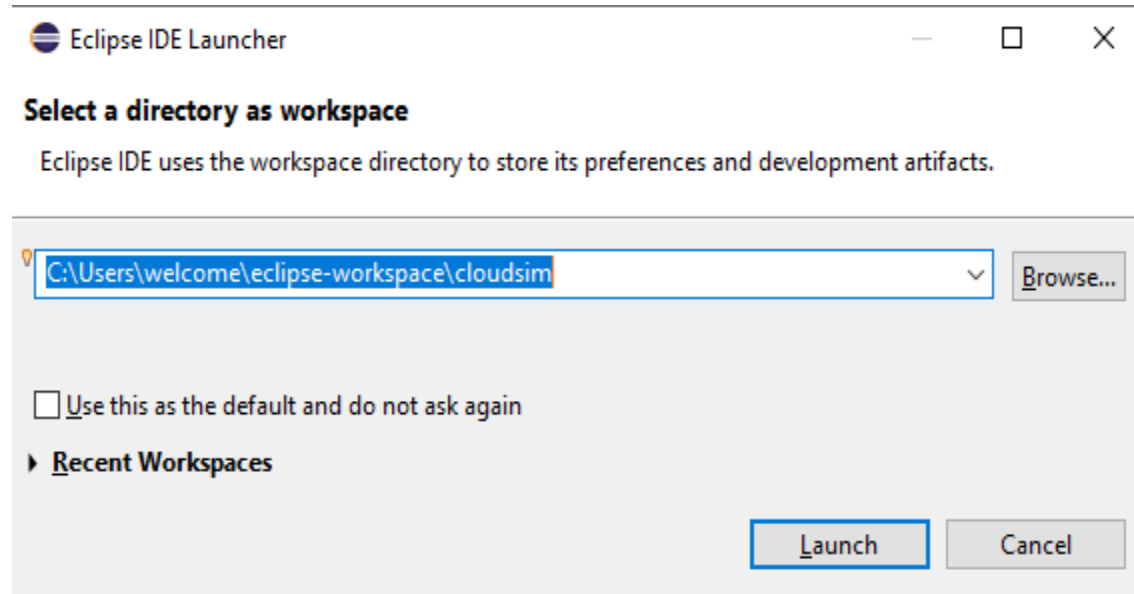
3.Executing the test case using the web drivers.

1.CONFIGURING THE SELENIUM EXTERNAL LIBRARIES IN THE ECLIPSE PROJECT

STEP1-(a)-download the selenium latest version from the official website.

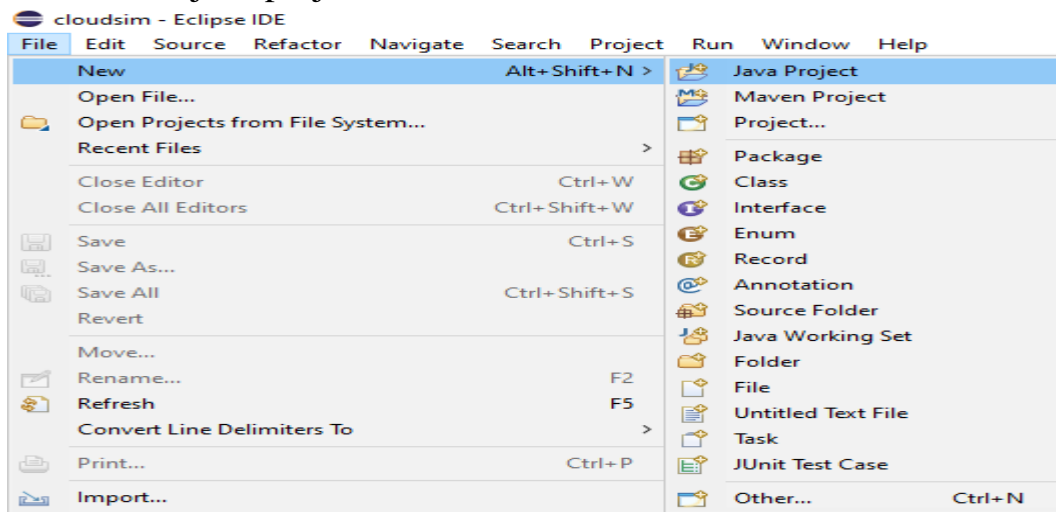
STEP1-(b)-extract the selenium Winrar into your directory.

STEP1-(c)-launch the eclipse.

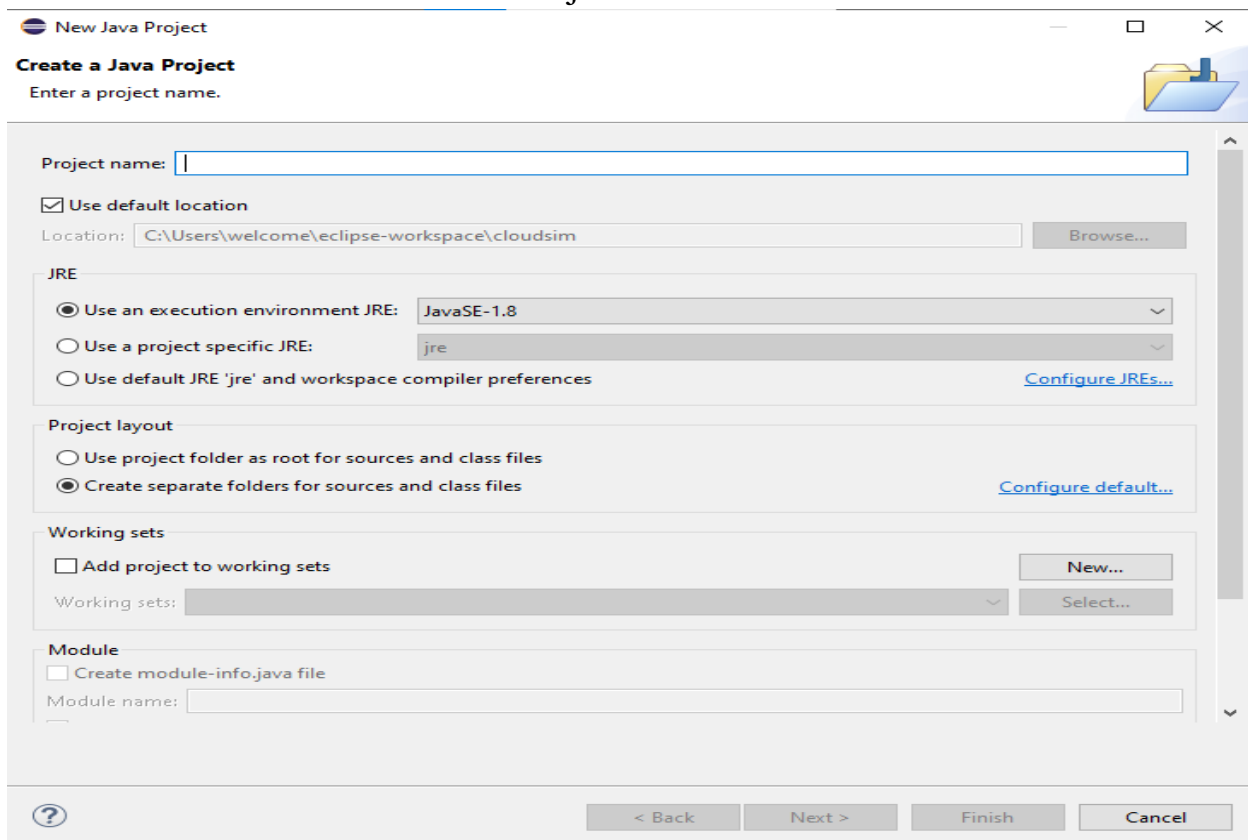


STEP1-(d)-then click on the new “file”,under this click on the “new”,under the new click on the “java project”.

file→new→java project.



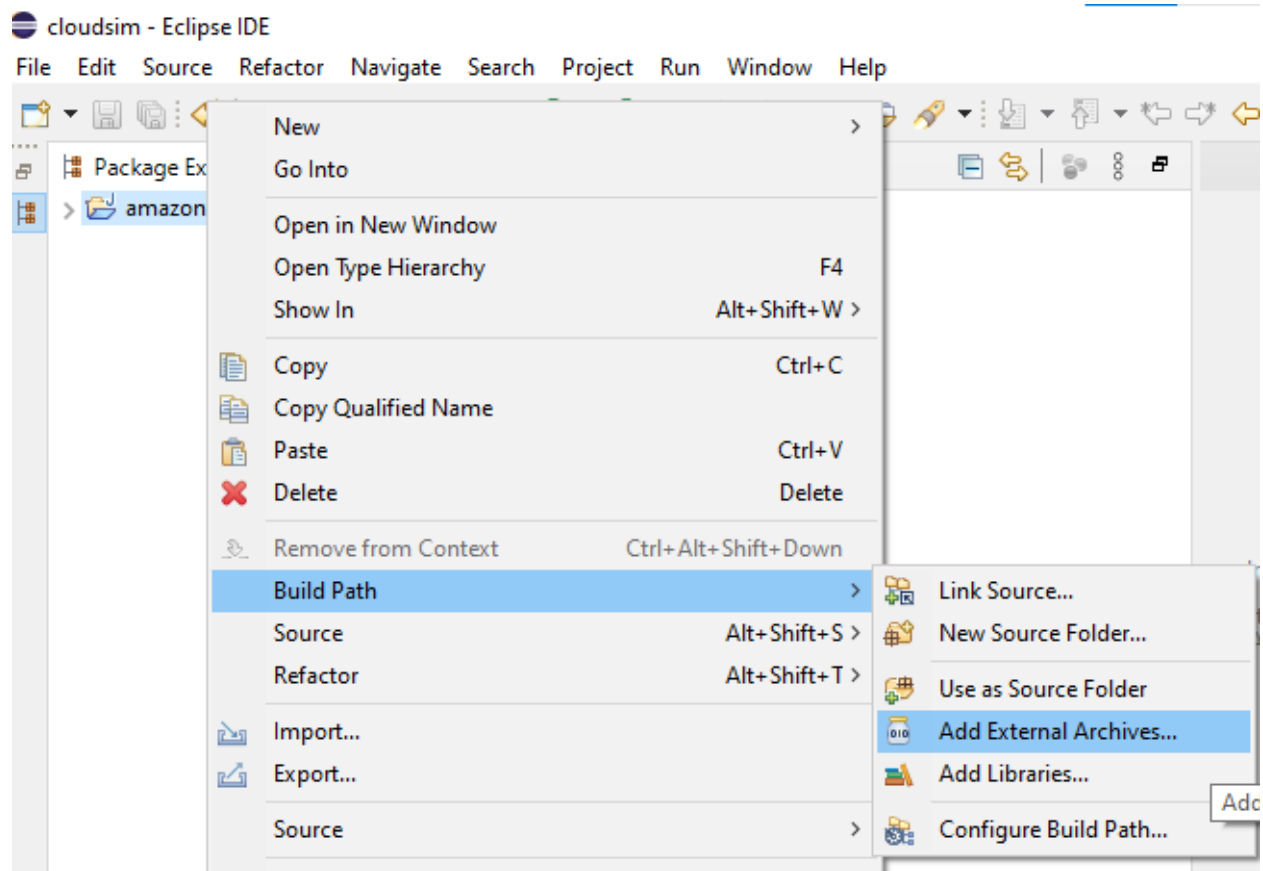
STEP1-(e)-after clicking on the “java project”.enter the name of the java project and set the execution environment as java S.E 1.8.



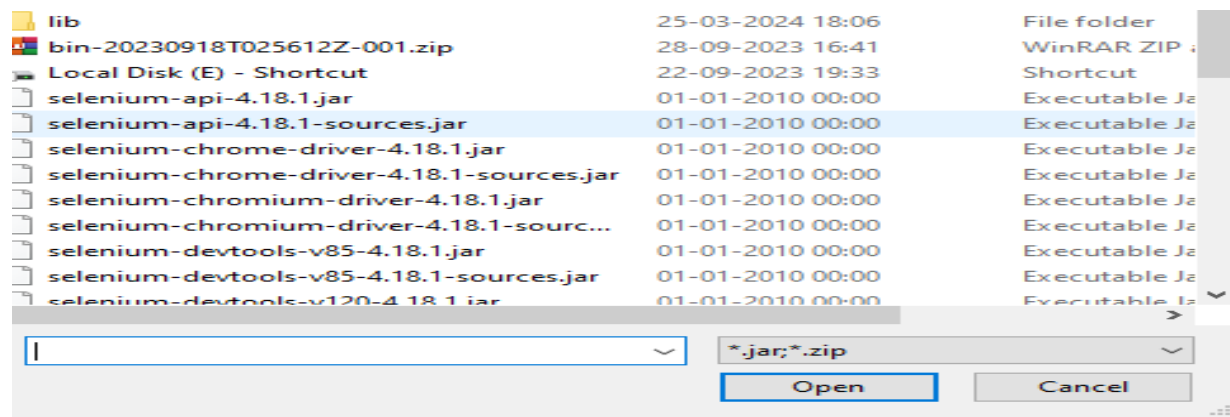
Then click on the finish button.

STEP1-(f) then right click on the project which is created on the project explorer. now click on the “build path”.under this click on the “add external archives”.

Build path→Add External Archives



STEP1-(g)- In the “add external archives“ add all the necessary jars of the selenium that you previously extracted in your directory as shown below.

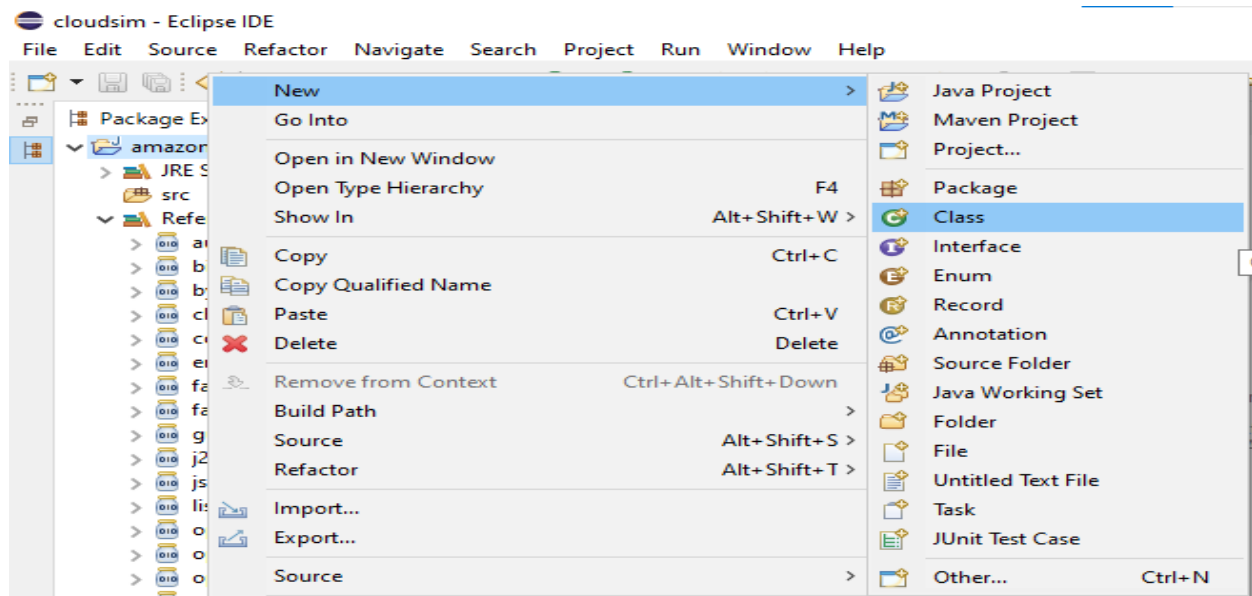


Select all the jars and then click on open to add selenium jars.now the selenium was successfully configured in the eclipse.

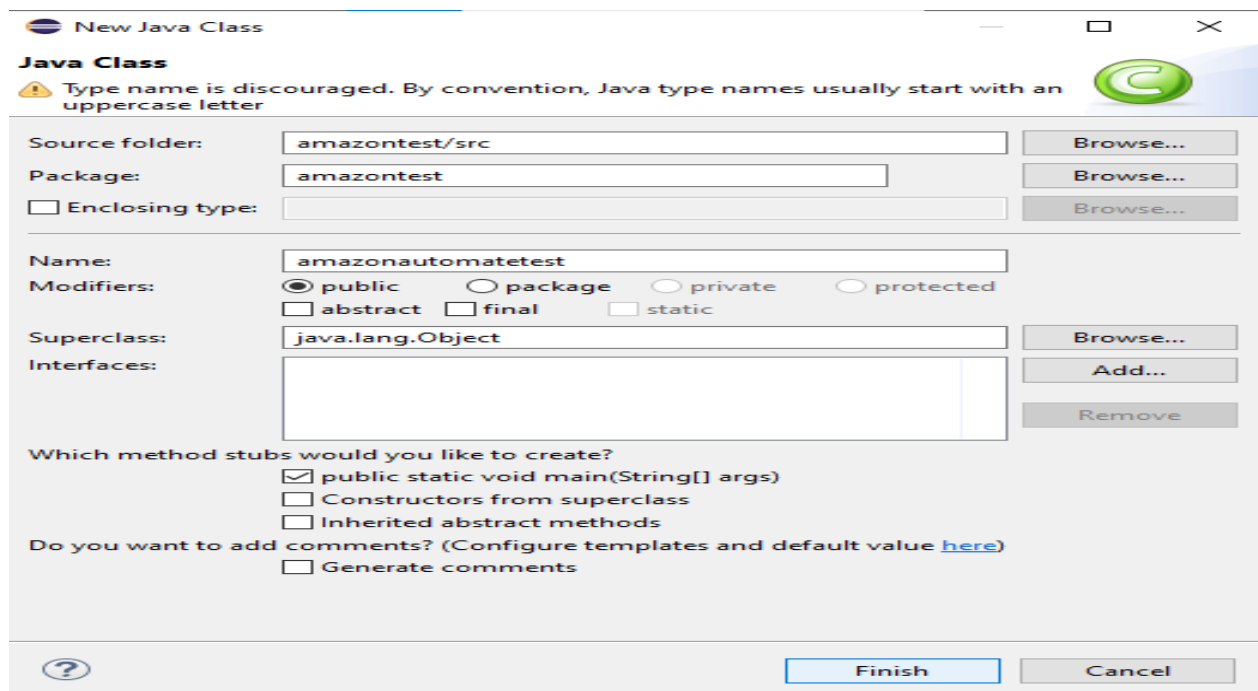
2.DEVELOPING THE TEST CASE.

STEP2-(a)-right click on the project in which the selenium was configured.under this click on the new class and create the new class.

New→Class



STEP2-(b)-name the class as the datadriven and make it as the main class.



Then click on the finish button.

The java code for testing the searchbox of the e-commerce application(amazon.in) is given below.

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
public class DataDriven{

    public static void main(String[] args) throws IOException {
        String textFilePath = "E:\\test.txt";
        BufferedReader reader = new BufferedReader(new FileReader(textFilePath));

        System.setProperty("webdriver.chrome.driver", "E:\\chromedriver-
win64\\chromedriver.exe"); // Set path to chromedriver.exe
        WebDriver driver = new ChromeDriver();

        driver.get("https://amazon.in"); // Replace with your eCommerce website URL

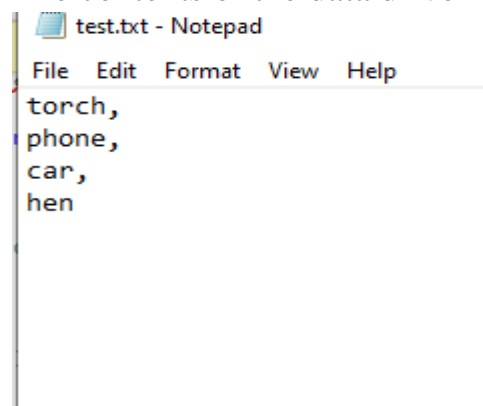
        String line;
        while ((line = reader.readLine()) != null) {
            String[] data = line.split(","); // Assuming the delimiter is comma (,)

            String username = data[0].trim();

            driver.get("http://www.amazon.in/");
            WebElement searchbox=driver.findElement(By.id("twotabsearchtextbox"));
            searchbox.sendKeys(username);
            searchbox.submit();
        }

        // Close the browser
        driver.quit();
        reader.close();
    }
}
```

The contents of the data driven text file is shown below.



OUTPUT OF DATA DRIVEN TESTING

```
Problems @ Javadoc Declaration Console
terminated> jnk [Java Application] C:\Users\welcome\Desktop\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full
pr 22, 2024 7:08:24 AM org.openqa.selenium.devtools.CdpVersionFinder findNearestMatch
WARNING: Unable to find an exact match for CDP version 124, returning the closest version;
```

Search bar: All torch

EN Hello, sign in Account & Lists Returns & Orders

Mobiles Today's Deals Prime Customer Service Electronics New Releases

HALONIX A Complete Range of Downlighters. Save up to 73% on Halonix >

Product	Price	Discount
HALONIX Downlighter	₹239	32% off (₹349)
HALONIX Downlighter	₹779	73% off (₹2,900)
HALONIX Downlighter	₹399	43% off (₹699)

product page for other buying options. Price and other details may vary based on product

ted test software.

Coimbatore 641014

Search bar: All phone

EN Hello, sign in Account & Lists Returns & Orders

Sell Best Sellers Mobiles Today's Deals Prime Customer Service Electronics

Apple iPhone

Newphoria
Shop Apple India



Apple iPhone 15 (256 GB) - Green
★★★★★ 1,157
prime



Apple iPhone 15 Plus (256 GB) - Blue
★★★★★ 556
prime



Cruise in Style with Jammbo Cars
Shop Jammbo



Deal of the Day

₹9,447
₹14,999 (37% off)



₹37,994

Results

Check each product page for other buying options.

Results

Price and other details may vary based on product size and colour.



Sponsored ⓘ

SKOLA - SK 034 Toys Nesting
Hen - Stack and Nest The Eggs

★★★★★ ~ 2



Narmu Toys Soft Rooster Toys
for Kids with jumbling Soft Hen
Toy for Kids

★★★★★ ~ 51



Moirs 7pcs X
Miniature De
Multi Color f
Terrariums, I

RESULT

Thus the data driven framework was built successfully using the selenium.

(b) Build Page Object Model Using Selenium And TestNg.

PAGE OBJECT MODEL (POM):

The Page Object model is an object design pattern in Selenium, where web pages are represented as classes, the various elements on the page are defined as variables in the class and all possible user interaction can then be implemented as methods in the class.

AIM

to build the page object model using selenium and TestNg.

PROCEDURE

Step-1-launch the eclipse.

Step-2-create the new project and also add the Library Junit,testing,selenium.

Step-3-create the two new classes as shown below.

create the new class by right clicking on the project created.

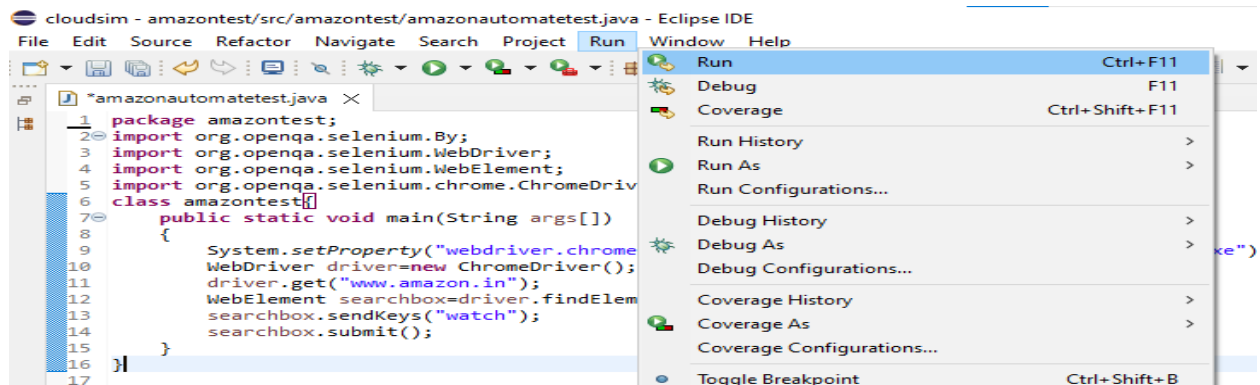
new → class.

name the class as test and type the following code.

```
import org.junit.Test;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
public class SearchPage{
    @Test
    public void test()
    {
        System.setProperty("webdriver.chrome.driver","E:\\sta\\chromedriver-
win64\\chromedriver.exe");
        WebDriver driver=new ChromeDriver();
        driver.get("http://www.google.in/");
        WebElement searchbox=driver.findElement(By.id("APjFqb"));
        searchbox.sendKeys("led tv");
        searchbox.submit();
    }
}
```

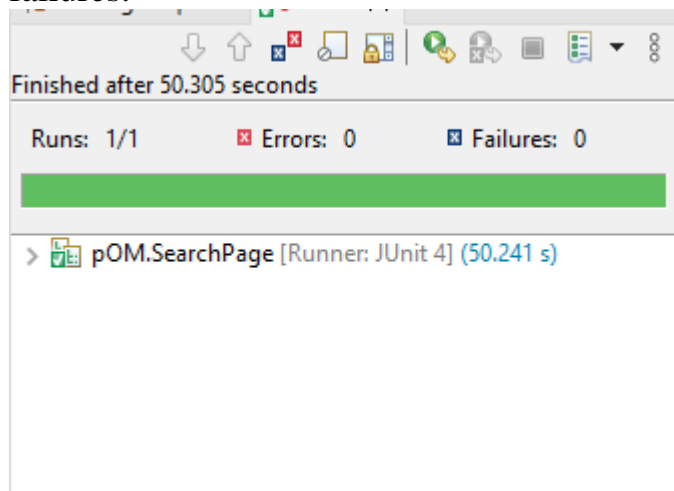
Step-4-before executing the test case.check your chrome version if it is not updated then update it first and then download the chrome driver which corresponds to the updated version of the chrome browser.then pass the path of the chromedriver as the second parameter to the method setProperty("par1","par2").

Step-5-now execute the test by clicking on Run in eclipse.



OUTPUT:

on the successful run it shows the following summary of output and there was no failures.



RESULT

Thus the POM was built successfully using the selenium and TestNg.

c)The Bdd Was Build Successfully Using The Selenium And Cucumber AIM

To the BDD was build successfully using the selenium and cucumber.

CUCUMBER

Cucumber is one such open-source tool, which supports Behavior Driven Development(BDD). In simple words, Cucumber can be defined as a testing framework, driven by plain English. It serves as documentation, automated tests, and development aid – all in one.

- Dependency List
1. Cucumber Java – 7.14.0
 2. Cucumber TestNG – 7.14.0
 3. Java 11
 4. Maven – 3.8.6
 5. Selenium – 4.14.0
 6. TestNG – 7.8.0
 7. WebDriverManager – 5.5.3

Below is an example of Test Scenarios in the feature file. I have failed one test scenario intentionally –

@MissingUsername.

1 Feature: Login to HRM Application

2

3 Background:

4 Given User is on HRMLLogin page "<https://opensource-demo.orangehrmlive.com/>"

5

6 @ValidCredentials

7 Scenario: Login with valid credentials

8

9 When User enters username as "Admin" and password as "admin123"

10 Then User should be able to login successfully and new page open

11

12 @InvalidCredentials

13 Scenario Outline: Login with invalid credentials

15 When User enters username as "<username>" and password
as "<password>"

16 Then User should be able to see error message
"<errorMessage>" 17 18 Examples:

19	username	password	errorMessage
20	Admin	admin12\$\$	Invalid credentials
21	admin\$\$	admin123	Invalid credentials
22	abc123	xyz\$\$	Invalid credentials

25 @MissingUsername

26 Scenario Outline: Login with blank username

28 When User enters username as " " and password as "admin123"

29 Then User should be able to see a message "Required1" below
Username

PROCEDURE

Step-1-launch the eclipse.

Step-2-create the new project and also add the Library JUnit.

Step-3-create the two new classes as shown below.

create the new class by right clicking on the project created.
new → class.

name the class as Step and type the following code.

```
package cucumber;

//Import necessary packages
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;

import cucumber.api.java.en.Given;
import cucumber.api.java.en.Then;
import cucumber.api.java.en.When;
import static org.junit.Assert.*;

import org.junit.Test;

//Step definitions class
public class Step {

    // WebDriver instance
    WebDriver driver;

    // Given step
```

```

@Given("^I open the browser$")
public void i_open_the_browser() {
    // Set ChromeDriver path
    System.setProperty("webdriver.chrome.driver", "E:\\sta\\chromedriver-
win64\\chromedriver.exe");

    // Initialize ChromeDriver
    driver = new ChromeDriver();
    throw new io.cucumber.java.PendingException();
}

// When step
@When("^I navigate to Google$")
public void i_navigate_to_Google() {
    // Open Google homepage
    driver.get("https://www.google.com");
    throw new io.cucumber.java.PendingException();
}

// Then step
@Then("^I should see the search box$")
public void i_should_see_the_search_box() {
    // Verify search box is displayed
    assertTrue(driver.findElement(By.name("q")).isDisplayed());
    throw new io.cucumber.java.PendingException();
}

// Close the browser
@Then("^I close the browser$")
public void i_close_the_browser() {
    // Close the browser
    driver.quit();
    throw new io.cucumber.java.PendingException();
}
}

```

OUTPUT:

Test	# Passed	# Skipped	# Retried	# Failed	Time (ms)	Included Groups	Excluded Groups
Suite							
Cucumber6 with TestNG	4	0	0	1	49,301		
Class	Method	Start	Time (ms)				
Suite							
Cucumber6 with TestNG --- failed							
Cucumber6 with TestNG --- passed							
com.example.runner.CucumberRunnerTests	runScenario	10:05:09.110	10:05:10.110	10:05:10.110	49,301		
com.example.runner.CucumberRunnerTests	runScenario	10:05:10.110	10:05:11.110	10:05:11.110	49,301		
com.example.runner.CucumberRunnerTests	runScenario	10:05:11.110	10:05:12.110	10:05:12.110	49,301		
com.example.runner.CucumberRunnerTests	runScenario	10:05:12.110	10:05:13.110	10:05:13.110	49,301		

Cucumber6 with TestNG

com.example.runner.CucumberRunnerTests#runScenario

Parameter #1	Parameter #2
"Login with blank username"	"Login to HRM Application"
Exception	
<pre> java.lang.AssertionError: expected [Required] but found [Required] at com.example.runner.CucumberRunnerTests.runScenario(CucumberRunnerTests.java:63) at @User should be able to see a message "Required" below Username(File:///C:/Users/Vijay/Automation/pom_cucumber_testingdemo/src/test/resources/features/LoginPage.feature:29) ... Removed 6 stack frames </pre>	
back to summary	

com.example.runner.CucumberRunnerTests#runScenario

Parameter #1	Parameter #2
"Login with invalid credentials"	"Login to HRM Application"

[back to summary](#)

RESULT

Thus the BDD was build successfully using the selenium and cucumber.