**Lab Exercise - Exploratory Testing using JUnit**

**Objective:**

To perform **Exploratory Testing** using **JUnit** in Java by testing an **E-commerce Checkout System** without predefined test cases.

**What is Exploratory Testing?**

Exploratory testing involves **simultaneous learning, test design, and execution**. Instead of following scripted test cases, the tester dynamically **explores the software** to find potential issues.

**Scenario: E-Commerce Checkout System**

We will test an **Order Processing System** that calculates the total price based on:

* **Item price**
* **Quantity**
* **Discounts (if applicable)**
* **Shipping fees**

**Business Logic:**

* If **quantity is 0 or negative**, return "Invalid Order".
* If **total price is above $100**, apply a **10% discount**.
* If **total price is below $10**, add a **$2 shipping fee**.
* Otherwise, return the **final payable amount**.

**Steps to Perform**

1. Implement a **basic checkout system** in Java.
2. Write **JUnit tests** that explore **edge cases and unexpected inputs**.
3. Run the tests and **log defects (if found)**.

**Step 1: Implement the Checkout System**

Create a class **CheckoutSystem.java**:

public class CheckoutSystem {

public static String processOrder(double price, int quantity) {

if (quantity <= 0) {

return "Invalid Order";

}

double total = price \* quantity;

// Apply discount if total > $100

if (total > 100) {

total \*= 0.9;

}

// Add shipping fee if total < $10

if (total < 10) {

total += 2;

}

return "Total Payable: $" + String.format("%.2f", total);

}

}

**Step 2: Write JUnit Test Cases for Exploratory Testing**

Create a test class **CheckoutSystemTest.java**:

import static org.junit.Assert.assertEquals;

import org.junit.Test;

public class CheckoutSystemTest {

@Test

public void testNegativeQuantity() {

assertEquals("Invalid Order", CheckoutSystem.processOrder(10.0, -1));

}

@Test

public void testZeroQuantity() {

assertEquals("Invalid Order", CheckoutSystem.processOrder(10.0, 0));

}

@Test

public void testLowTotalPrice() {

assertEquals("Total Payable: $4.00", CheckoutSystem.processOrder(2.0, 2)); // Below $10, add $2 shipping

}

@Test

public void testDiscountApplication() {

assertEquals("Total Payable: $90.00", CheckoutSystem.processOrder(10.0, 10)); // $100 total, 10% discount

}

@Test

public void testNoDiscount() {

assertEquals("Total Payable: $50.00", CheckoutSystem.processOrder(10.0, 5)); // $50, no discount

}

@Test

public void testExactDiscountThreshold() {

assertEquals("Total Payable: $90.00", CheckoutSystem.processOrder(25.0, 4)); // $100 total, 10% discount

}

@Test

public void testFreeShippingThreshold() {

assertEquals("Total Payable: $10.00", CheckoutSystem.processOrder(5.0, 2)); // $10 total, no shipping fee

}

@Test

public void testLargeOrder() {

assertEquals("Total Payable: $900.00", CheckoutSystem.processOrder(50.0, 20)); // $1000 total, 10% discount

}

}

**Step 3: Execute the Tests**

Run **JUnit 4 or JUnit 5** in your IDE (**Eclipse, IntelliJ, VS Code**).

**Expected Output (JUnit Results)**

✔ **testNegativeQuantity()** → **Passed**  
✔ **testZeroQuantity()** → **Passed**  
✔ **testLowTotalPrice()** → **Passed**  
✔ **testDiscountApplication()** → **Passed**  
✔ **testNoDiscount()** → **Passed**  
✔ **testExactDiscountThreshold()** → **Passed**  
✔ **testFreeShippingThreshold()** → **Passed**  
✔ **testLargeOrder()** → **Passed**

**Key Takeaways from Exploratory Testing**

✅ **Explores real-world scenarios** instead of following fixed scripts.  
✅ **Uncovers edge cases**, like **zero or negative quantities**.  
✅ **Finds defects dynamically**, ensuring **robust test coverage**.