**Lab Exercise 2- Code Refactoring & Analysis in Java Using GitHub Copilot in IntelliJ IDEA**

This step-by-step Exercise will help you **analyze and refactor Java code** using **GitHub Copilot** in **IntelliJ IDEA**. The exercise will cover **identifying code smells**, **applying refactoring techniques**, and **using Copilot prompts** to improve the code.

**Prerequisites**

Ensure you have the following installed:

1. **IntelliJ IDEA (Community or Ultimate)** – [Download Here](https://www.jetbrains.com/idea/download/)
2. **Java JDK (11 or later)** – [Install Guide](https://adoptopenjdk.net/)
3. **Maven (or Gradle)** – [Install Guide](https://maven.apache.org/install.html)
4. **Git** – [Download Git](https://git-scm.com/)
5. **GitHub Copilot** – Follow the official [GitHub Copilot Setup Guide](https://github.com/features/copilot)
6. **GitHub Account** – Create an account at [GitHub](https://github.com/)

**Step 1: Set Up a Java Project in IntelliJ IDEA**

1. Open **IntelliJ IDEA**.
2. Click **New Project**.
3. Select **Java** and click **Next**.
4. Choose **Maven** as the build system.
5. Set **GroupId** to com.example and **ArtifactId** to RefactoringLab.
6. Choose a project location and click **Finish**.
7. Open the **Terminal** (View → Tool Windows → Terminal) and initialize Git:

git init

**Step 2: Install and Enable GitHub Copilot**

1. Go to **File** → **Settings** → **Plugins**.
2. Search for **GitHub Copilot**, install it, and restart IntelliJ IDEA.
3. Sign in to GitHub when prompted.
4. Enable Copilot in **Settings** → **GitHub Copilot**.

**Step 3: Write an Initial Java Class with Code Smells**

1. Navigate to src/main/java/com/example/.
2. Create a new Java class named **OrderProcessor.java**.

package com.example;

public class OrderProcessor {

public void processOrder(String customerName, double orderAmount) {

System.out.println("Processing order for " + customerName);

if (orderAmount > 1000) {

System.out.println("Applying discount...");

orderAmount = orderAmount \* 0.9;

}

System.out.println("Final Order Amount: " + orderAmount);

System.out.println("Sending confirmation email to " + customerName);

}

public void processSpecialOrder(String customerName, double orderAmount) {

System.out.println("Processing special order for " + customerName);

if (orderAmount > 5000) {

System.out.println("Applying special discount...");

orderAmount = orderAmount \* 0.8;

}

System.out.println("Final Order Amount: " + orderAmount);

System.out.println("Sending confirmation email to " + customerName);

}

}

1. This code has **code smells**:
   * **Duplicated code** in processOrder and processSpecialOrder.
   * **Long methods** with multiple responsibilities.

**Step 4: Identify Code Smells and Generate a Refactoring Plan**

1. **Use IntelliJ Code Analysis**:
   * Right-click the class → Select **Analyze** → **Inspect Code**.
   * IntelliJ will highlight duplicated code and long methods.
2. **Use Copilot for Suggestions**:
   * Add this **prompt as a comment** above the class:

// Suggest refactoring for this class to remove duplication and improve maintainability

* + **Press Enter** and review Copilot’s suggestions.

**Step 5: Refactor Using GitHub Copilot**

**Step 5.1: Extract Duplicate Code into a Separate Method**

1. Add this **prompt as a comment** inside OrderProcessor.java:

// Extract the common logic into a separate method

1. **Press Enter** and let Copilot suggest refactoring:

private double applyDiscount(double orderAmount, double discountThreshold, double discountRate) {

if (orderAmount > discountThreshold) {

System.out.println("Applying discount...");

return orderAmount \* discountRate;

}

return orderAmount;

}

1. **Modify processOrder and processSpecialOrder to use this method**:

public void processOrder(String customerName, double orderAmount) {

System.out.println("Processing order for " + customerName);

orderAmount = applyDiscount(orderAmount, 1000, 0.9);

finalizeOrder(customerName, orderAmount);

}

public void processSpecialOrder(String customerName, double orderAmount) {

System.out.println("Processing special order for " + customerName);

orderAmount = applyDiscount(orderAmount, 5000, 0.8);

finalizeOrder(customerName, orderAmount);

}

**Step 5.2: Extract Common Finalization Logic**

1. Add this **prompt as a comment**:

// Extract the common order finalization logic into a separate method

1. **Press Enter** and let Copilot generate:

private void finalizeOrder(String customerName, double orderAmount) {

System.out.println("Final Order Amount: " + orderAmount);

System.out.println("Sending confirmation email to " + customerName);

}

1. **Run tests to ensure functionality is preserved.**

**Step 6: Write Unit Tests for Refactored Code**

1. Create a new test class **OrderProcessorTest.java** in src/test/java/com/example/.
2. Use the following **Copilot prompt**:

// Write JUnit tests for the refactored OrderProcessor class

1. **Press Enter** to let Copilot generate:

package com.example;

import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.\*;

class OrderProcessorTest {

@Test

void testApplyDiscount() {

OrderProcessor processor = new OrderProcessor();

double discountedPrice = processor.applyDiscount(2000, 1000, 0.9);

assertEquals(1800, discountedPrice, "Discount should be applied correctly");

}

}

1. Run the tests to validate the refactoring.

**Step 7: Commit and Push to GitHub**

1. Initialize a repository and push the changes:

git add .

git commit -m "Refactored OrderProcessor to remove duplication"

git push -u origin main