**Lab Exercise 3- Optimizing Java Code for Performance Using GitHub Copilot in IntelliJ IDEA**

This step-by-step guide will help you **optimize Java code for performance** using **GitHub Copilot** in **IntelliJ IDEA**. The exercise covers **identifying bottlenecks, refactoring for efficiency, and measuring performance improvements**.

**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 PerformanceOptimizationLab.
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 Inefficient Java Class**

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

package com.example;

public class PrimeNumberChecker {

public boolean isPrime(int number) {

if (number < 2) return false;

for (int i = 2; i < number; i++) {

if (number % i == 0) {

return false;

}

}

return true;

}

}

1. **Issue**: This algorithm is **inefficient** for large numbers (O(n) complexity).

**Step 4: Measure Performance Using System.nanoTime()**

1. Create a test class **PerformanceTest.java** in src/test/java/com/example/.
2. Add the following **GitHub Copilot prompt as a comment**:

// Write a performance test for the isPrime method using System.nanoTime()

1. **Press Enter**, and Copilot will generate:

package com.example;

import org.junit.jupiter.api.Test;

class PerformanceTest {

@Test

void testPrimeNumberPerformance() {

PrimeNumberChecker checker = new PrimeNumberChecker();

int testNumber = 10000019; // A large prime number

long startTime = System.nanoTime();

boolean result = checker.isPrime(testNumber);

long endTime = System.nanoTime();

System.out.println("Execution Time: " + (endTime - startTime) + " ns");

System.out.println("Is Prime: " + result);

}

}

1. **Run the test** and observe the execution time.

**Step 5: Optimize the Code with GitHub Copilot**

1. Open PrimeNumberChecker.java.
2. Add this **Copilot prompt as a comment**:

// Optimize the isPrime method using the square root method

1. **Press Enter**, and Copilot will suggest a more efficient version:

public boolean isPrime(int number) {

if (number < 2) return false;

if (number == 2 || number == 3) return true;

if (number % 2 == 0 || number % 3 == 0) return false;

for (int i = 5; i \* i <= number; i += 6) {

if (number % i == 0 || number % (i + 2) == 0) {

return false;

}

}

return true;

}

1. **Rationale**:
   * Reduces unnecessary iterations (O(√n) complexity).
   * Skips even numbers after checking for 2 and 3.

**Step 6: Re-run Performance Test**

1. Run PerformanceTest.java again.
2. Compare execution times **before and after optimization**.
3. Observe a significant reduction in execution time.