Lab Title: Exploring Java Concepts Together

Objective: This lab aims to introduce you to various fundamental Java concepts, including streams and filters, GUI development with Swing, the Singleton design pattern, concurrency, and basic unit testing with JUnit5. Through a step-by-step approach, you'll create a Java program that combines these concepts into a cohesive application.

Part 1: Numbers Magic (Streams and Filters)

Objective: Learn to manipulate data using Java streams and filter numbers based on their properties.

Task: Read numbers from a file and filter them into even and odd groups.

Hints:

Use FileReader and BufferedReader to read the numbers from a file.

Utilize the split method to separate the numbers.

Implement a logic using if statements to identify even and odd numbers.

Sample Input (numbers.txt):

Copy code

1 2 3 4 5 6 7 8 9

Sample Output:

yaml

Copy code

Even Numbers: 2 4 6 8

Odd Numbers: 1 3 5 7 9

Part 2: Fun Window (GUI with Swing)

Objective: Understand how to create a basic GUI using Swing components in Java.

Task: Build a small graphical window with interactive elements.

Hints:

Explore JFrame and JPanel to construct the window.

Implement a JTextArea for user input.

Use an ActionListener to count vowels and consonants as users type.

Sample Input (user types in GUI):

Copy code

Hello, Java!

Sample Output:

Type something: Hello, Java!

Vowels: 4

Consonants: 5

Part 3: Secret Keeper (Singleton Design Pattern)

Objective: Implement the Singleton design pattern to create a single instance for recording and accessing data.

Task: Develop a special logger class to store filtered numbers securely.

Hints:

Create a class with a private constructor to prevent multiple instances.

Include a static method to access the single instance of the logger.

Use this logger to record filtered numbers.

Sample Output (logs):

Log: Recorded even and odd numbers

Part 4: Speed Counter (Concurrency)

Objective: Learn about concurrent programming and handle shared resources safely.

Task: Create a multi-threaded program to count rapidly using shared resources.

Hints:

Develop a shared counter class.

Implement multiple threads using Runnable or Thread to increment this counter.

Synchronize the counter incrementation method to ensure thread safety.

Sample Output:

Final Count: 5000

Pass these tests:

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

import javax.swing.\*;

import java.io.\*;

import java.util.\*;

public class JavaConceptsTestSuite {

@Test

void testEvenAndOddNumbers() {

// Test Part 1: Numbers Magic (Streams and Filters)

List<Integer> evenNumbers = new ArrayList<>();

List<Integer> oddNumbers = new ArrayList<>();

try (BufferedReader reader = new BufferedReader(new FileReader("numbers.txt"))) {

String line;

while ((line = reader.readLine()) != null) {

String[] numbers = line.split("\\s+");

for (String number : numbers) {

int num = Integer.parseInt(number);

if (num % 2 == 0) {

evenNumbers.add(num);

} else {

oddNumbers.add(num);

}

}

}

} catch (IOException e) {

e.printStackTrace();

}

Assertions.assertEquals(Arrays.asList(2, 4, 6, 8), evenNumbers);

Assertions.assertEquals(Arrays.asList(1, 3, 5, 7, 9), oddNumbers);

}

@Test

void testVowelAndConsonantCounter() {

// Test Part 2: Fun Window (GUI with Swing)

JTextArea textArea = new JTextArea();

textArea.setText("Hello, Java!");

int vowels = 0;

int consonants = 0;

String text = textArea.getText().toLowerCase();

for (char c : text.toCharArray()) {

if (c >= 'a' && c <= 'z') {

if (c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u') {

vowels++;

} else {

consonants++;

}

}

}

Assertions.assertEquals(4, vowels);

Assertions.assertEquals(5, consonants);

}

@Test

void testLoggerInstance() {

// Test Part 3: Secret Keeper (Singleton Design Pattern)

Logger logger1 = Logger.getInstance();

Logger logger2 = Logger.getInstance();

Assertions.assertSame(logger1, logger2);

}

@Test

void testConcurrency() {

// Test Part 4: Speed Counter (Concurrency)

Counter counter = new Counter();

int numThreads = 5;

Thread[] threads = new Thread[numThreads];

for (int i = 0; i < numThreads; i++) {

threads[i] = new Thread(() -> {

for (int j = 0; j < 1000; j++) {

counter.increment();

}

});

threads[i].start();

}

for (Thread thread : threads) {

try {

thread.join();

} catch (InterruptedException e) {

e.printStackTrace();

}

}

Assertions.assertEquals(5000, counter.getCount());

}

}