National University of Sciences and Technology (NUST)
School of Electrical Engineering and Computer Science

Faculty of Computing

SE-314: Software Construction

Class: BESE 13AB

Lab 09: Abstract Data Type - II

CLO-03: Design and develop solutions based on Software Construction principles.
CLO-04: Use modern tools such as Eclipse, NetBeans etc. for software construction.

Date: 18th Nov 2024

Time: 10:00 AM - 12:50 PM
02:00 PM – 04:50 PM

Farzan Saqib

417633

Instructor: Dr. Mehvish Rashid
Lab Engineer: Mr. Aftab Farooq

Introduction:

Lab 09: ADT- II

Students will have hands-on experience on designing, testing, and implementing abstract data types. Given a set of specifications, you will write unit tests that check for compliance with the specifications, and then implement code that meets the specifications. Use GitHub to collaborate with your group members and mention the work distribution.

Material:

https://ocw.mit.edu/ans7870/6/6.005/s16/psets/ps2/

Lectures on LMS regarding designing Abstract Data Types

Lab Tasks

Solve problem 4 of problem set 2 listed on the link.

Task1: Test Graph Poet

Devise, document, and implement tests for `GraphPoet` in **`GraphPoetTest.java`** .

Task2: Implement `GraphPoet` in `GraphPoet.java` .

You must use `Graph` in the rep of `GraphPoet` , but the implementation is otherwise entirely up to you.

Github branch for lab9:
https://github.com/EggsyOnCode/sc-labs/pull/new/lab9

GraphPoet Implementation:

```
public class GraphPoet {

    private final Graph<String> graph = new ConcreteGraph<>();

    // Abstraction function:
    //   Represents a word affinity graph where vertices are words, and edges
are weighted by adjacency frequency.
    // Representation invariant:
    //   - Graph vertices represent case-insensitive words extracted from the
corpus.
    //   - Edge weights are non-negative integers.
    // Safety from rep exposure:
    //   - The graph is encapsulated and not directly exposed.
    //   - The class does not expose any mutable references to internal state.

    /**
     * Create a new poet with the graph from corpus (as described above).
     *
     * @param corpus text file from which to derive the poet's affinity graph
     * @throws IOException if the corpus file cannot be found or read
     */
    public GraphPoet(File corpus) throws IOException {
        List<String> lines = Files.readAllLines(corpus.toPath());
        StringBuilder content = new StringBuilder();
        for (String line : lines) {
            content.append(line).append(" ");
        }

        String[] words = content.toString().toLowerCase().split("\\s+");
        for (int i = 0; i < words.length - 1; i++) {
            String word1 = words[i];
            String word2 = words[i + 1];
            int currentWeight = graph.set(word1, word2,
graph.targets(word1).getOrDefault(word2, 0) + 1);
        }

        checkRep();
    }
```

```
    private void checkRep() {
        for (String vertex : graph.vertices()) {
            for (int weight : graph.targets(vertex).values()) {
                assert weight ⩾ 0 : "Edge weight must be non-negative";
            }
        }
    }

    /**
     * Generate a poem.
     *
     * @param input string from which to create the poem
     * @return poem (as described above)
     */
    public String poem(String input) {
        String[] words = input.split("\\s+");
        StringBuilder result = new StringBuilder();

        for (int i = 0; i < words.length - 1; i++) {
            String word1 = words[i].toLowerCase();
            String word2 = words[i + 1].toLowerCase();

            // Find the best bridge word
            String bridge = null;
            int maxWeight = 0;

            for (Map.Entry<String, Integer> target :
graph.targets(word1).entrySet()) {
                String potentialBridge = target.getKey();
                int weight = target.getValue() +
graph.targets(potentialBridge).getOrDefault(word2, 0);
                if (graph.targets(potentialBridge).containsKey(word2) && weight
> maxWeight) {
                    bridge = potentialBridge;
                    maxWeight = weight;
                }
            }

            result.append(words[i]).append(" ");
            if (bridge ≠ null) {
                result.append(bridge).append(" ");
            }
        }

        result.append(words[words.length - 1]);
        return result.toString();
    }
```

```
    @Override
    public String toString() {
        return "GraphPoet with graph: " + graph.toString();
    }
}
```

GraphPoet Test Implementation:

```
package poet;

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


import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertThrows;

import graph.ConcreteGraph;
import graph.Graph;
import org.junit.jupiter.api.Test;

import java.io.File;
import java.io.IOException;
import java.nio.file.Files;

class GraphPoetTest {

    @Test
    void testPoemGeneration() throws IOException {
        // Create a temporary file with a simple corpus
        File corpus = File.createTempFile("corpus", ".txt");
        corpus.deleteOnExit();
        Files.writeString(corpus.toPath(), "This is a test of the Mugar Omni
Theater sound system.");

        GraphPoet poet = new GraphPoet(corpus);

        String input = "Test the system.";
        String expectedOutput = "Test of the system.";
        assertEquals(expectedOutput, poet.poem(input));
    }

    @Test
    void testPoemNoBridgeWords() throws IOException {
        File corpus = File.createTempFile("corpus", ".txt");
        corpus.deleteOnExit();
```

```
        Files.writeString(corpus.toPath(), "Hello world!");

        GraphPoet poet = new GraphPoet(corpus);

        String input = "Goodbye world.";
        String expectedOutput = "Goodbye world.";
        assertEquals(expectedOutput, poet.poem(input));
    }

    @Test
    void testEmptyInput() throws IOException {
        File corpus = File.createTempFile("corpus", ".txt");
        corpus.deleteOnExit();
        Files.writeString(corpus.toPath(), "Hello world!");

        GraphPoet poet = new GraphPoet(corpus);

        String input = "";
        String expectedOutput = "";
        assertEquals(expectedOutput, poet.poem(input));
    }

    @Test
    void testInvalidFile() {
        assertThrows(IOException.class, () -> new GraphPoet(new
File("nonexistent.txt")));
    }


    @Test
    public void testToStringWithGraphData() throws IOException {
        // Arrange: Create a poet with a simple corpus
        File corpus = createTempCorpus("Hello world. Hello everyone.");
        GraphPoet poet = new GraphPoet(corpus);

        // Act: Call the toString() method
        String result = poet.toString();

        // Assert: Verify that the string contains graph details
        assertTrue(result.contains("GraphPoet with graph: "), "toString()
should include 'GraphPoet with graph:'.");
        assertTrue(result.contains("hello"), "Graph should include the word
'hello'.");
        assertTrue(result.contains("world"), "Graph should include the word
'world'.");
        assertTrue(result.contains("everyone"), "Graph should include the word
'everyone'.");
    }
```

```
    // Helper to create a temporary corpus file
    private File createTempCorpus(String content) throws IOException {
        File tempFile = File.createTempFile("corpus", ".txt");
        tempFile.deleteOnExit();
        try (java.io.FileWriter writer = new java.io.FileWriter(tempFile)) {
            writer.write(content);
        }
        return tempFile;
    }
}
```

Test Case Documentation for `GraphPoetTest`

This documentation outlines the purpose, setup, execution, and assertions for each test case in the `GraphPoetTest` suite. The tests aim to ensure the correctness of the `GraphPoet` class implementation, particularly focusing on poem generation and the `toString()` method.

---

1. `testPoemGeneration`

- **Purpose**:
  Verifies that the poem generation function correctly inserts bridge words when there is a two-edge-long path with maximum weight in the affinity graph.

- **Setup**:
  A temporary corpus is created containing the text:
  `"This is a test of the Mugar Omni Theater sound system."`

- **Execution**:
  Input string: `"Test the system."`
  Expected output: `"Test of the system."`

- **Assertions**:

  - The generated poem matches the expected output.

---

2. `testPoemNoBridgeWords`

- **Purpose**:

Ensures that the poem generation function does not modify the input when no bridge words exist in the affinity graph.

- **Setup**:
  A temporary corpus is created containing the text:
  `"Hello world!"`

- **Execution**:
  Input string: `"Goodbye world."`
  Expected output: `"Goodbye world."`

- **Assertions**:

  - The generated poem matches the input string, as no bridge words exist.

---

3. `testEmptyInput`

- **Purpose**:
  Tests the edge case where the input string is empty. The generated poem should also be empty.

- **Setup**:
  A temporary corpus is created containing the text:
  `"Hello world!"`

- **Execution**:
  Input string: `""`
  Expected output: `""`

- **Assertions**:

  - The generated poem is an empty string.

---

4. `testInvalidFile`

- **Purpose**:
  Verifies that an appropriate exception is thrown when the corpus file does not exist.

- **Setup**:
  No actual corpus file is provided; instead, a non-existent file path is used.

- **Execution**:
  Attempt to create a `GraphPoet` instance with the non-existent file.

- **Assertions**:

  - An `IOException` is thrown.

---

5. `testToStringWithGraphData`

- **Purpose**:
  Verifies the correctness of the `toString()` method by ensuring it includes details about the graph's vertices and edges.

- **Setup**:
  A temporary corpus is created containing the text:
  `"Hello world. Hello everyone."`

- **Execution**:
  Call the `toString()` method on a `GraphPoet` instance created with the above corpus.

- **Assertions**:

  - The output string includes the phrase `"GraphPoet with graph: "`.
  - The output string includes the vertices `"hello"`, `"world"`, and `"everyone"`.
  - The graph structure in the string is consistent with the input corpus.

Helper Method Documentation

`createTempCorpus`

- **Purpose**:
  Provides a utility to create a temporary file with specified text content for testing purposes.

- **Execution**:

  - A temporary file is created and written with the provided content.
  - The file is marked for deletion upon JVM exit to avoid leftover files.

- **Parameters**:

    - `content` (`String`): Text to be written to the temporary corpus file.
- **Returns**:
    - A `File` object pointing to the created temporary file.

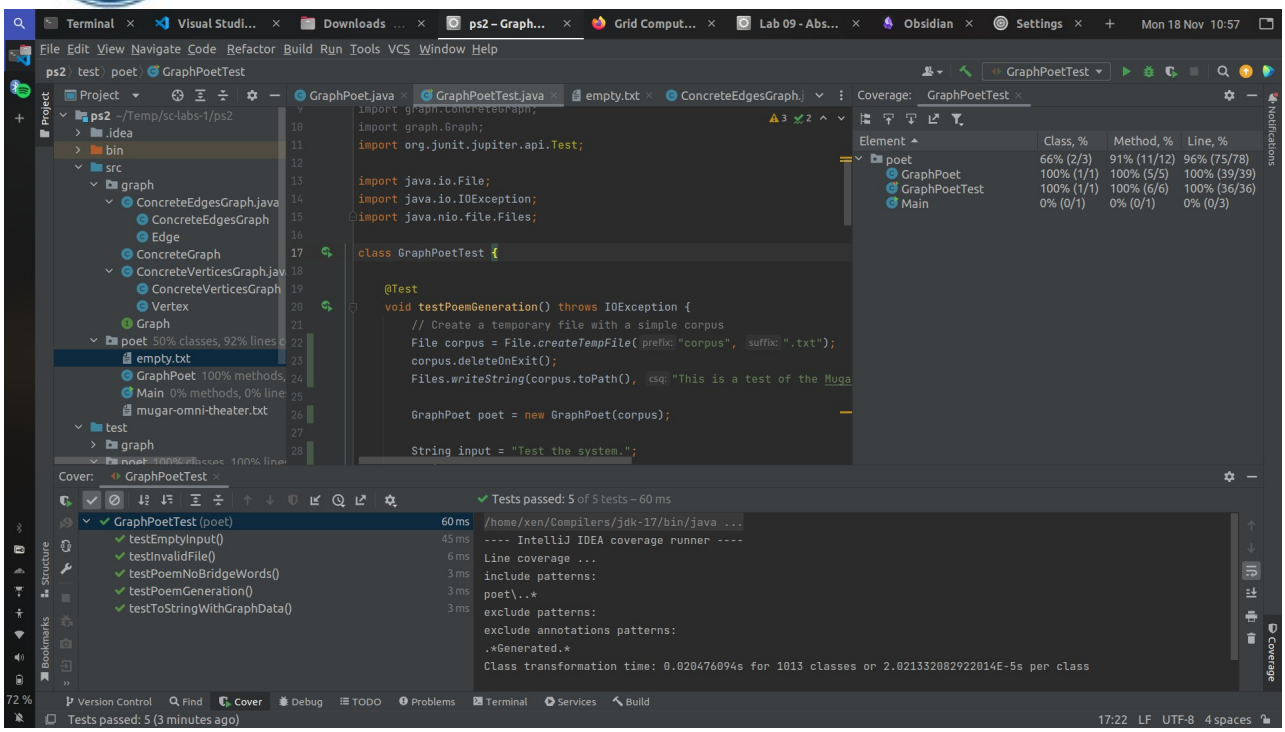
Running the Tests

- **Testing Framework**: JUnit 5
- **Dependencies**: Ensure `org.junit.jupiter` is added to the project.
- **Execution**:
  Run the test suite using an IDE, `mvn test` (Maven), or `gradle test` (Gradle).

Expected Results

All tests should pass if the `GraphPoet` class is implemented correctly:

- Poem generation produces the correct bridge words.
- Edge cases like empty input or missing corpus file are handled gracefully.
- The `toString()` method reflects the correct state of the graph.



100% coverage and test success

Solution
- Push Your Code on GitHub- Add Git Link in Document. **Source Code: Zip your source code and upload one file (Including Git link) on LMS as well.**

Deliverables:

Compile a single word document by filling in the solution part and submit this Word file on LMS. In case of any problems with submissions on LMS, submit your Lab assignments by emailing it to aftab.farooq@seecs.edu.pk.