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

Department of Computing

SE-314: SOFTWARE CONSTRUCTION

Class: BESE-13AB

Lab 07: Recursion

NAME:	Ahmad Shajee
CMS ID:	414764
Class/Section:	BESE 13-A | SEECS
Semester:	Fall - 2024

Date: November 3, 2023

LAB 07: RECURSION

Introduction:

Students will have hands-on experience on designing, testing, and implementing recursive problems. Given a scenario, you will write the specifications and implement it by dividing into base case and recursive step. You may design helper methods to simplify your implementations. Write unit tests that check for compliance with the specifications.

Lab Tasks

Task 1: Recursive File Search

Objective: The objective of this lab task is to create a Java program that recursively searches for a file within a directory and its subdirectories. This exercise will help you practice the principles of software construction and recursion.

Instructions:

1. Create a Java program that takes two command-line arguments: a directory path and a file name to search for.
2. Implement a recursive function to search for the specified file within the given directory and its subdirectories.
3. The program should display a message when it finds the file, including the full path to the file, or a message indicating that the file was not found.
4. Follow good coding practices, including meaningful variable names, comments, and modular code.
5. Implement error handling to handle cases where the specified directory does not exist or other exceptions may occur.
6. Use appropriate data structures and algorithms to efficiently search through the directory tree.
7. Test your program with different directory paths and file names to ensure its correctness and reliability.

Important: Do not forget to write the specifications and unit tests for the code.

Optional Enhancements:

1. Allow the program to search for multiple files in a single run.
2. Implement a feature to count the number of times a specific file appears within the directory and its subdirectories.

3. Provide an option to specify whether the search should be case-sensitive or case-insensitive.

Deliverables:

1. A well-commented (Specs) Java program that fulfills the requirements described above along with unit tests.
2. A brief report that explains the design decisions you made while implementing the program and any enhancements you added.

Grading Criteria:

Your lab task will be evaluated based on the following criteria:

1. Correctness: Does the program correctly find the specified file(s) and handle errors?
2. Code Quality: Is the code well-organized, readable, and maintainable?
3. Recursion: Is recursion used appropriately for the task?
4. Error Handling: Does the program handle exceptions and errors gracefully?
5. Optional Enhancements: Are any optional enhancements implemented effectively?

Remember to adhere to best practices in software construction, including modular design, meaningful variable names, and effective error handling.

Task 2: Recursive String Permutations

Objective: The objective of this lab task is to create a Java program that generates all permutations of a given string using a recursive algorithm. This exercise will help you practice recursion and algorithm design.

Instructions:

1. Create a Java program that generates all permutations of a given string using a recursive function.
2. Implement a recursive function **generatePermutations** that takes a string as input and returns a list of all its permutations.
3. Use a recursive approach to generate permutations. You can consider swapping characters in the string to create different permutations.
4. Follow good coding practices, including meaningful variable names, comments, and modular code.
5. Implement error handling to handle cases where the input string is empty or other exceptions may occur.
6. Analyze the time complexity of the recursive algorithm. How does the time complexity compare to an iterative solution for large strings?

Optional Enhancements:

1. Provide an option for the user to choose whether to include or exclude duplicate permutations, as some characters in the input string may be identical.
2. Implement a non-recursive algorithm for generating permutations and compare its performance with the recursive solution for large strings.

Deliverables:

1. A well-commented (specs) Java program that generates all permutations of a given string using a recursive algorithm and handles potential errors along with unit tests.
2. A brief report explaining the time complexity of the recursive algorithm and any optional enhancements you added.

Grading Criteria:

Your lab task will be evaluated based on the following criteria:

1. Correctness: Does the program correctly generate all permutations of the input string using recursion?
2. Code Quality: Is the code well-organized, readable, and maintainable?
3. Recursion: Is recursion used appropriately for the task?
4. Error Handling: Does the program handle exceptions and errors gracefully?
5. Optional Enhancements: Are any optional enhancements implemented effectively?

Remember to adhere to best practices in software construction, including modular design, meaningful variable names, and effective error handling.

Task 1:

Code:

```
package lab07;

import java.io.File;

/**
 * This Java program recursively searches for a file within a directory and its subdirectories.
 */
public class FileSearch {


    private boolean fileFound;
    private String foundFilePath;
```

```java
    public FileSearch() {
        this.fileFound = false;
        this.foundFilePath = null;
    }

    public boolean isFileFound() {
        return fileFound;
    }

    public String getFoundFile() {
        return foundFilePath;
    }
    /**
     * Main method to execute the program.
     *
     * @param args Command-line arguments: directory path and target filename.
     */
    public static void main(String[] args) {
        if (args.length != 2) {
            System.out.println("Usage: java FileSearch <directory> <filename>");
            System.exit(1);
        }

        FileSearch f = new FileSearch();
        String directoryPath = args[0];
        String targetFileName = args[1];

        File directory = new File(directoryPath);

        if (!directory.exists() || !directory.isDirectory()) {
            System.out.println("Invalid directory path.");
            System.exit(1);
        }

        f.searchFile(directory, targetFileName);
    }

    /**
     * Recursively searches for the specified file within the given directory and its subdirectories.
     *
     * @param directory       The directory to start the search from.
     * @param targetFileName  The name of the file to search for.
     */

    public void searchFile(File directory, String targetFileName) {
        File[] files = directory.listFiles();
        if (files != null) {
            for (File file : files) {
                if (file.isDirectory()) {
                    searchFile(file, targetFileName); // Recursively search subdirectories
                } else {
                    if (file.getName().equals(targetFileName)) {
                        System.out.println("File found at: " + file.getAbsolutePath());
                        fileFound = true; // Set the flag to true when the file is found
                        foundFilePath = file.getAbsolutePath(); // Store the found file path
                        return; // Stop searching after finding the file
                    }
                }
            }
        }
    }

}
```

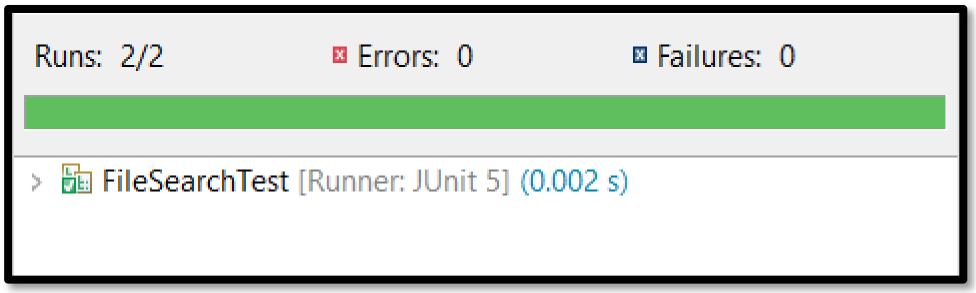
Output:

```
File found at: C:\Users\Asfand Yar\eclipse-workspace\HelloWorld\src\lab07\text1.txt
```

Test Case Code:

```
package lab07;

import static org.junit.Assert.*;

import java.io.File;

import org.junit.Before;
import org.junit.Test;

public class FileSearchTest {
    private FileSearch fileSearch;

    @Before
    public void setUp() {
        fileSearch = new FileSearch();
    }

    @Test
    public void testFileFound() {
        // Provide the directory path and target file name
        String directoryPath = "C:\\Users\\Asfand Yar\\eclipse-workspace\\HelloWorld\\src";
        String targetFileName = "text1.txt";

        // Call the searchFile method
        fileSearch.searchFile(new File(directoryPath), targetFileName);

        // Assert that the file is found
        assertTrue(fileSearch.isFileFound());
    }

    @Test
    public void testFileNotFound() {
        // Provide the directory path and a non-existent target file name
        String directoryPath = "src/test/resources";
        String targetFileName = "nonExistentFile.txt";

        // Call the searchFile method
        fileSearch.searchFile(new File(directoryPath), targetFileName);

        // Assert that the file is not found
        assertFalse(fileSearch.isFileFound());
        assertNull(fileSearch.getFoundFile());
    }
}
```

<u>Output Screenshot</u>:



```
Runs:  2/2          Errors:  0          Failures:  0

> FileSearchTest [Runner: JUnit 5] (0.002 s)
```

Task 2:

Code:

```
package lab07;

import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

/**
 * This Java program generates all permutations of a given string using a recursive algorithm.
 */
public class StringPermutations {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a string: ");
        String input = scanner.nextLine();
        scanner.close();

        if (input.isEmpty()) {
            System.out.println("Input string is empty.");
        } else {
            List<String> permutations = generatePermutations(input);

            if (permutations.isEmpty()) {
                System.out.println("No permutations found for the input string.");
            } else {
                System.out.println("Permutations of the input string:");
                for (String permutation : permutations) {
                    System.out.println(permutation);
                }
            }
        }
    }

    /**
     * Generates all permutations of the input string.
     *
     * @param input The input string for which permutations are to be generated.
     * @return A list of all permutations of the input string.
     */
    public static List<String> generatePermutations(String input) {
        List<String> permutations = new ArrayList<>();
        generatePermutationsRecursive("", input, permutations);
        return permutations;
    }

    /**
     * Recursively generates permutations of the input string.
     *
     * @param prefix      The current permutation being built.
     * @param remaining   The remaining characters for permutation.
     * @param permutations The list to store generated permutations.
     */
    private static void generatePermutationsRecursive(String prefix, String remaining,
List<String> permutations) {
        int n = remaining.length();
        if (n == 0) {
            permutations.add(prefix);
        } else {
            for (int i = 0; i < n; i++) {
                generatePermutationsRecursive(prefix + remaining.charAt(i), remaining.substring(0, i)
+ remaining.substring(i + 1), permutations);
            }
        }
    }
}
```

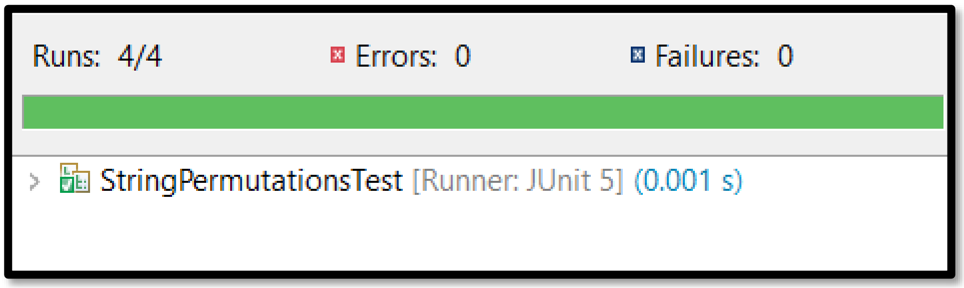
<u>Output</u>:

```
Enter a string: asf
Permutations of the input string:
asf
afs
saf
sfa
fas
fsa
```

Test Case Code:

```
package lab07;

import static org.junit.Assert.*;
import org.junit.Before;
import org.junit.Test;

public class StringPermutationsTest {
    private StringPermutations stringPermutations;

    @Before
    public void setUp() {
        stringPermutations = new StringPermutations();
    }

    @Test
    public void testPermutationsForEmptyString() {
        String input = "";
        String[] expected = { "" };
        assertPermutations(input, expected);
    }

    @Test
    public void testPermutationsForSingleCharacterString() {
        String input = "a";
        String[] expected = { "a" };
        assertPermutations(input, expected);
    }

    @Test
    public void testPermutationsForShortString() {
        String input = "abc";
        String[] expected = { "abc", "acb", "bac", "bca", "cab", "cba" };
        assertPermutations(input, expected);
    }

    @Test
    public void testPermutationsForLongerString() {
        String input = "abcd";
        String[] expected = { "abcd", "abdc", "acbd", "acdb", "adbc", "adcb",
                              "bacd", "badc", "bcad", "bcda", "bdac", "bdca",
                              "cabd", "cadb", "cbad", "cbda", "cdab", "cdba",
                              "dabc", "dacb", "dbac", "dbca", "dcab", "dcba" };
        assertPermutations(input, expected);
    }

    private void assertPermutations(String input, String[] expected) {
        String[] actualPermutations = stringPermutations.generatePermutations(input).toArray(new
String[0]);
        assertArrayEquals(expected, actualPermutations);
    }
}
```

Output Screenshot:



```
Runs: 4/4        Errors: 0        Failures: 0

> StringPermutationsTest [Runner: JUnit 5] (0.001 s)
```

Deliverables

Compile a single word document by filling in the solution part and submit this Word file on LMS. This lab grading policy is as follows: The lab is graded between 0 to 10 marks. Insert the solution/answer in this document. You must show the implementation of the tasks in the designing tool, along with your completed Word document to get your work graded. You must also submit this Word document on the LMS. In case of any problems with submissions on LMS, submit your Lab assignmentsby emailing it to aftab.farooq@seecs.edu.pk.