**Name: Muhammad Affan Amir**

**CMS ID: 405260**

**Faculty of Computing**

**SE-314: Software Construction**

**Class: BESE 13AB**

# Lab 07: Recursion

**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: 28th Oct 2024**

**Time: 10:00 AM** **- 12:50 PM   
 02:30 PM – 04:50 PM**

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

**Introduction:**

# Lab 07: Recursion

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.

**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.

**Source Code: Zip your source code along with report and upload one file on LMS as well.  
   
 Upload Your Code on Github and share Github link in Report as well.**

**Solution**

### Github Repository: <https://github.com/Affan2900/Lab_07_SC>

### Task 1 code:

### //FileSearcher.java

*// FileSearcher.java*

import java.io.File;

import java.util.ArrayList;

import java.util.List;

*/\*\**

*\* A utility class that provides recursive file searching capabilities within directories.*

*\*/*

*public* *class* FileSearcher {

*private* *final* boolean caseSensitive;

*private* int totalMatches;

*private* *final* List<File> matchedFiles;

*/\*\**

*\* Constructs a new FileSearcher with specified case sensitivity.*

*\**

*\* @param caseSensitive whether the search should be case-sensitive*

*\*/*

*public* FileSearcher(boolean *caseSensitive*) {

        this.caseSensitive = *caseSensitive*;

        this.totalMatches = 0;

        this.matchedFiles = *new* ArrayList<>();

    }

*/\*\**

*\* Searches for a file in the specified directory and its subdirectories.*

*\**

*\* @param directoryPath the starting directory path*

*\* @param fileName      the name of the file to search for*

*\* @return List of File objects representing the matched files*

*\* @throws IllegalArgumentException if the directory path is invalid or inaccessible*

*\*/*

*public* List<File> searchFile(String *directoryPath*, String *fileName*) {

        matchedFiles.clear();

        totalMatches = 0;

        File directory = *new* File(*directoryPath*);

        validateDirectory(directory);

        searchFileRecursively(directory, *fileName*);

*return* matchedFiles;

    }

*private* void searchFileRecursively(File *directory*, String *fileName*) {

        File[] files = *directory*.listFiles();

*if* (files == null) {

*return*;

        }

*for* (File file *:* files) {

*if* (file.isFile()) {

                String currentFileName = file.getName();

                boolean matches = caseSensitive

*?* currentFileName.equals(*fileName*)

*:* currentFileName.equalsIgnoreCase(*fileName*);

*if* (matches) {

                    matchedFiles.add(file);

                    totalMatches++;

                }

            } *else* *if* (file.isDirectory()) {

                searchFileRecursively(file, *fileName*);

            }

        }

    }

*private* void validateDirectory(File *directory*) {

*if* (!*directory*.exists()) {

*throw* *new* IllegalArgumentException("Directory does not exist: " + *directory*.getPath());

        }

*if* (!*directory*.isDirectory()) {

*throw* *new* IllegalArgumentException("Path is not a directory: " + *directory*.getPath());

        }

*if* (!*directory*.canRead()) {

*throw* *new* IllegalArgumentException("Directory is not readable: " + *directory*.getPath());

        }

    }

*public* int getTotalMatches() {

*return* totalMatches;

    }

}

### //FileSearcherTest.java

*// Create this in a separate file named FileSearcherTest.java*

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.Test;

import org.junit.jupiter.api.io.TempDir;

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

import java.io.File;

import java.io.IOException;

import java.nio.file.Path;

import java.util.List;

*class* FileSearcherTest {

*private* FileSearcher caseSensitiveSearcher;

*private* FileSearcher caseInsensitiveSearcher;

    @BeforeEach

    void setUp() {

        caseSensitiveSearcher = *new* FileSearcher(true);

        caseInsensitiveSearcher = *new* FileSearcher(false);

    }

    @Test

    void testSearchNonExistentDirectory() {

        assertThrows(IllegalArgumentException.*class*,

            () -> caseSensitiveSearcher.searchFile("nonexistent/directory", "test.txt"));

    }

    @Test

    void testSearchFileFound(@TempDir Path *tempDir*) throws IOException {

*// Create test file*

        File testFile = *new* File(*tempDir*.toFile(), "test.txt");

        assertTrue(testFile.createNewFile());

*// Test search*

        List<File> results = caseSensitiveSearcher.searchFile(

*tempDir*.toString(), "test.txt");

        assertEquals(1, results.size());

        assertTrue(results.contains(testFile));

    }

    @Test

    void testCaseInsensitiveSearch(@TempDir Path *tempDir*) throws IOException {

        File testFile = *new* File(*tempDir*.toFile(), "Test.txt");

        assertTrue(testFile.createNewFile());

        List<File> results = caseInsensitiveSearcher.searchFile(

*tempDir*.toString(), "test.txt");

        assertEquals(1, results.size());

        assertEquals(testFile, results.get(0));

    }

}

### //Main.java

// Main.java

import java.io.File;

import java.util.List;

public class Main {

public static void main(String[] args) {

// Ensure at least two arguments are provided: directory path and file name.

if (args.length < 2) {

System.***out***.println("Usage: java Main <directory\_path> <file\_name> [case\_sensitive]");

return;

}

// Parse the command-line arguments.

String directoryPath = args[0];

String fileName = args[1];

boolean caseSensitive = args.length > 2 && Boolean.*parseBoolean*(args[2]);

try {

// Create a new FileSearcher instance with the specified case sensitivity.

FileSearcher searcher = new FileSearcher(caseSensitive);

// Perform the search in the specified directory for the given file name.

List<File> foundFiles = searcher.searchFile(directoryPath, fileName);

// Display the results.

if (foundFiles.isEmpty()) {

System.***out***.println("No files found matching: " + fileName);

} else {

System.***out***.println("Found " + searcher.getTotalMatches() + " matching file(s):");

for (File file : foundFiles) {

System.***out***.println(file.getAbsolutePath());

}

}

} catch (IllegalArgumentException e) {

// Handle any errors encountered during the search.

System.***err***.println("Error: " + e.getMessage());

}

}

}

### Task 1 output:

### Running FileSearcher by specifying arguments:

### True – means the search should be Case Sensitive

### False – means the search should be Case Insensitive

### 

### 

### Task 2 code:

import java.util.ArrayList;

import java.util.HashSet;

import java.util.List;

import java.util.Scanner;

public class StringPermutations {

public static List<String> generatePermutations(String str) {

List<String> result = new ArrayList<>();

if (str == null || str.isEmpty()) {

System.***out***.println("Error: Input string cannot be null or empty.");

return result;

}

*permute*(str.toCharArray(), 0, result, new HashSet<>());

return result;

}

private static void permute(char[] chars, int index, List<String> result, HashSet<String> uniquePermutations) {

if (index == chars.length - 1) {

String permutation = new String(chars);

if (!uniquePermutations.contains(permutation)) {

result.add(permutation);

uniquePermutations.add(permutation);

}

return;

}

for (int i = index; i < chars.length; i++) {

*swap*(chars, index, i);

*permute*(chars, index + 1, result, uniquePermutations);

*swap*(chars, index, i); // backtrack

}

}

private static void swap(char[] chars, int i, int j) {

char temp = chars[i];

chars[i] = chars[j];

chars[j] = temp;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.***in***);

System.***out***.print("Enter a string to generate permutations: ");

String input = scanner.nextLine();

List<String> permutations = *generatePermutations*(input);

System.***out***.println("Permutations:");

for (String permutation : permutations) {

System.***out***.println(permutation);

}

System.***out***.println("Total permutations: " + permutations.size());

scanner.close();

}

}

### Task 2 output:

### 

### Time Complexity:

### The time complexity for this algorithm is O(n x n!):

### For a string of length n, there are n! permutations

### Each permutation takes a string to be printed using System.out.println(), as a result the time complexity of total algorithm in recursive approach is O(n x n!)

### 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.](mailto:aftab.farooq@seecs.edu.pk.)