**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.

|  |  |
| --- | --- |
| **NAME** | Muhammad Saad Umer |
| **CLASS** | BESE 13-A |
| **CMS** | 408485 |

**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.

|  |
| --- |
| **GITHUB LINK** |
| **https://github.com/MSaadUmer** |
| **SOURCE CODE** |
| TASK 1 (SPECS)  **Testing and Edge Cases:**  **Case 1**: Search for a file that exists in a directory (including subdirectories).  Example: If C:\Users\Documents contains file.txt, the program should print its full path.  **Case 2**: Search for a file that doesn't exist.  Example: If C:\Users\Documents does not contain missingfile.txt, the program should print that the file was not found.  **Case 3**: Invalid directory path.  Example: If the path is invalid or inaccessible, the program should print an error message.  **Case 4**: Search in a directory with many nested subdirectories.  The program should still find the file even if it is deep within the directory structure.  TASK 1 (CODE)  import java.io.File;  import java.io.IOException;  public class RecursiveFileSearch {  // Recursive function to search for the file  public static boolean searchFile(File directory, String fileName) {  // Check if the directory is valid  if (directory == null || !directory.exists()) {  System.out.println("Directory does not exist or is inaccessible.");  return false;  }    // If it's a file, check if it matches the target file name  if (directory.isFile()) {  if (directory.getName().equals(fileName)) {  System.out.println("File found: " + directory.getAbsolutePath());  return true;  }  return false;  }    // If it's a directory, list the files and check each one  File[] files = directory.listFiles();  if (files != null) {  for (File file : files) {  if (file.isDirectory()) {  // Recursively search in subdirectories  if (searchFile(file, fileName)) {  return true;  }  } else {  // Check if the current file is the one we are looking for  if (file.getName().equals(fileName)) {  System.out.println("File found: " + file.getAbsolutePath());  return true;  }  }  }  }  return false; // File not found in the directory or subdirectories  }  public static void main(String[] args) {  if (args.length != 2) {  System.out.println("Usage: java RecursiveFileSearch <directory-path> <file-name>");  return;  }  String directoryPath = args[0];  String fileName = args[1];    File directory = new File(directoryPath);    // Search for the file recursively  if (!searchFile(directory, fileName)) {  System.out.println("File '" + fileName + "' not found in the directory or its subdirectories.");  }  }  }  TASK 1 (UNIT TESTING)  import org.junit.jupiter.api.Test;  import static org.junit.jupiter.api.Assertions.\*;  import java.io.File;  class RecursiveFileSearchTest {  @Test  void testSearchFileFound() {  File testDir = new File("src/test/resources");  assertTrue(RecursiveFileSearch.searchFile(testDir, "file1.txt"));  }  @Test  void testSearchFileNotFound() {  File testDir = new File("src/test/resources");  assertFalse(RecursiveFileSearch.searchFile(testDir, "nonexistentfile.txt"));  }  @Test  void testSearchInEmptyDirectory() {  File testDir = new File("src/test/emptyDir");  assertFalse(RecursiveFileSearch.searchFile(testDir, "file.txt"));  }  @Test  void testSearchInvalidDirectory() {  File testDir = new File("invalid/path");  assertFalse(RecursiveFileSearch.searchFile(testDir, "file.txt"));  }  }  TASK 2 (SPECS)  Create a Java program that generates all permutations of a given string using a recursive algorithm. The program should output all permutations and handle edge cases such as empty or null strings.  TASK 2 (CODE)  import java.util.ArrayList;  import java.util.List;  public class StringPermutations {  /\*\*  \* This method generates all permutations of a given string.  \*  \* @param str The input string.  \* @return A list of all permutations of the input string.  \*/  public static List<String> generatePermutations(String str) {  List<String> permutations = new ArrayList<>();    // Handle the edge case where the input string is null or empty  if (str == null || str.length() == 0) {  return permutations;  }  // Start the recursive process  generatePermutationsHelper(str, 0, permutations);    return permutations;  }  /\*\*  \* This helper method performs the actual permutation generation recursively.  \*  \* @param str The input string.  \* @param index The index at which we are fixing the character.  \* @param permutations The list to store the generated permutations.  \*/  private static void generatePermutationsHelper(String str, int index, List<String> permutations) {  // Base case: if we reach the end of the string, add the permutation to the list  if (index == str.length()) {  permutations.add(str);  return;  }  // Iterate through all characters starting from the 'index' position  for (int i = index; i < str.length(); i++) {  // Swap characters at index and i  str = swap(str, index, i);  // Recurse with the next index  generatePermutationsHelper(str, index + 1, permutations);  // Backtrack by swapping the characters back  str = swap(str, index, i);  }  }  /\*\*  \* Helper method to swap characters at two given indices in the string.  \*  \* @param str The string to modify.  \* @param i The first index.  \* @param j The second index.  \* @return A new string with the characters at indices i and j swapped.  \*/  private static String swap(String str, int i, int j) {  char[] charArray = str.toCharArray();  char temp = charArray[i];  charArray[i] = charArray[j];  charArray[j] = temp;  return new String(charArray);  }  public static void main(String[] args) {  // Test case  String input = "abc";  List<String> permutations = generatePermutations(input);  // Display all permutations  System.out.println("All permutations of \"" + input + "\":");  for (String perm : permutations) {  System.out.println(perm);  }  }  }  TASK 2 (UNIT TESTING)  import org.junit.jupiter.api.Test;  import static org.junit.jupiter.api.Assertions.\*;  import java.util.List;  public class StringPermutationsTest {  @Test  public void testGeneratePermutations() {  List<String> result = StringPermutations.generatePermutations("abc");  assertEquals(6, result.size());  assertTrue(result.contains("abc"));  assertTrue(result.contains("acb"));  assertTrue(result.contains("bac"));  assertTrue(result.contains("bca"));  assertTrue(result.contains("cab"));  assertTrue(result.contains("cba"));  }  @Test  public void testEmptyString() {  List<String> result = StringPermutations.generatePermutations("");  assertTrue(result.isEmpty());  }  @Test  public void testSingleCharacter() {  List<String> result = StringPermutations.generatePermutations("a");  assertEquals(1, result.size());  assertTrue(result.contains("a"));  }  @Test  public void testNullInput() {  List<String> result = StringPermutations.generatePermutations(null);  assertTrue(result.isEmpty());  }  @Test  public void testStringWithDuplicates() {  List<String> result = StringPermutations.generatePermutations("aab");  assertEquals(3, result.size());  assertTrue(result.contains("aab"));  assertTrue(result.contains("aba"));  assertTrue(result.contains("baa"));  }  } |

**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.)