CSA0961 Programming in java for Distributed Applications

A.Manoj reddy(192311171)

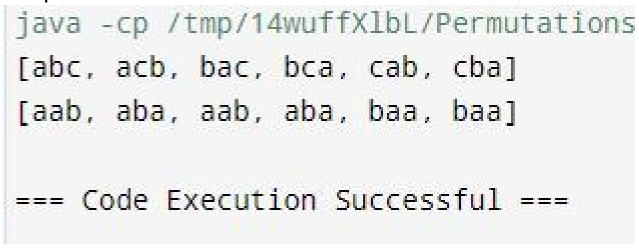
Test 9 (28.8.24) SET 2

1.Write a recursive method to generate all permutations of a given string. Test Cases: "abc" → [abc, bac, bca, cab, cba] "aab" → [aab, aba, baa]

Program:

```
# Permutations of a String

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

public class Permutations {

   public static void main(String[] args) {
      System.out.println(permute("abc"));
      System.out.println(permute("aab"));
   }

   public static List<String> permute(String str) {
      List<String> result = new ArrayList<>();
      permuteHelper("", str, result);
      return result;
   }

   private static void permuteHelper(String prefix, String remaining, List<String> result) {
      int n = remaining.length();
      if (n == 0) {
         result.add(prefix);
      } else {
         for (int i = 0; i < n; i++) {
            permuteHelper(prefix + remaining.charAt(i), remaining.substring(0, i) + remaining.substring(i
+ 1, n), result);
         }
      }
   }
}
```

Output:



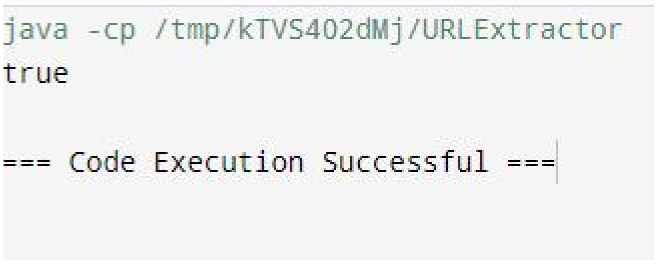
```
java -cp /tmp/14wuffXlbL/Permutations
[abc, acb, bac, bca, cab, cba]
[aab, aba, aab, aba, baa, baa]

=== Code Execution Successful ===
```

2. Write a Java method that extracts all valid URLs from a given string. A valid URL must start with http or https, followed by ://, and should contain a valid domain name. Test Cases: String text = "Visit https://www.example.com and http://www.test.com for more information."; List expected = Arrays.asList("https://www.example.com", "http://www.test.com"); assert extractURLs(text).equals(expected); // true

Program:

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class URLExtractor {

    public static List<String> extractURLs(String text) {
        List<String> urls = new ArrayList<>();
        String regex = "(http|https)://[\\w.-]+(?:\\.[\\w.-]+)+";
        Pattern pattern = Pattern.compile(regex);
        Matcher matcher = pattern.matcher(text);

        while (matcher.find()) {
            urls.add(matcher.group());
        }

        return urls;
    }

    public static void main(String[] args) {
        String text = "Visit https://www.example.com and http://www.test.com for 
more information.";
        List<String> expected = Arrays.asList("https://www.example.com", 
"http://www.test.com");
        assert extractURLs(text).equals(expected); // true
    }
}
```

Output:



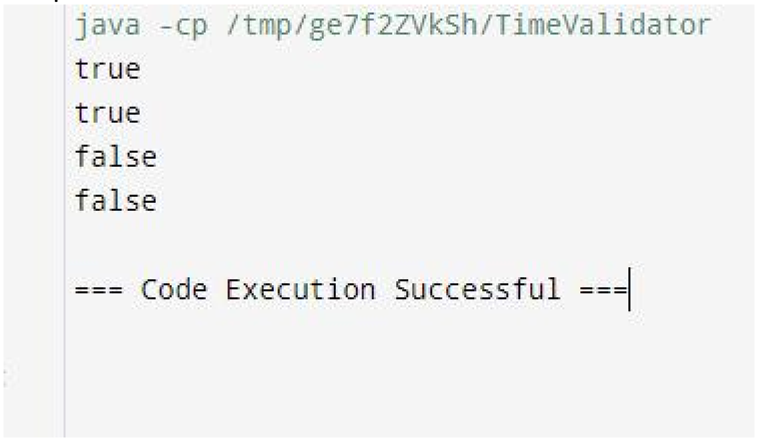
```
java -cp /tmp/kTVS402dMj/URLExtractor
true

=== Code Execution Successful ===
```

3. Write a Java method to validate if a given string represents a valid time in the 24-hour format (HH). Test Cases: assert isValidTime("23:59"); // true assert isValidTime("00:00"); // true assert !isValidTime("24:00"); // false
assert !isValidTime("12:60"); // false

Program:

```
public class TimeValidator {
    public static boolean isValidTime(String time) {
        if (time == null || !time.matches("\\d{2}:\\d{2}")) {
            return false;
        }
        String[] parts = time.split(":");
        int hours = Integer.parseInt(parts[0]);
        int minutes = Integer.parseInt(parts[1]);
        return hours >= 0 && hours < 24 && minutes >= 0 && minutes < 60;
    }

    public static void main(String[] args) {
        System.out.println(isValidTime("23:59")); // true
        System.out.println(isValidTime("00:00")); // true
        System.out.println(isValidTime("24:00")); // false
        System.out.println(isValidTime("12:60")); // false
    }
}
```

Output:



```
java -cp /tmp/ge7f2ZVkSh/TimeValidator
true
true
false
false

=== Code Execution Successful ===
```

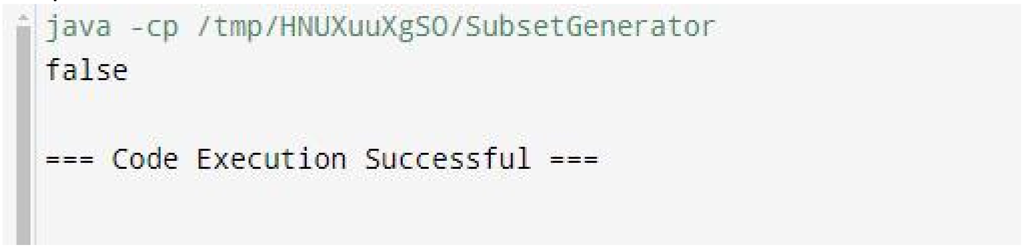
4. Write a recursive Java method to generate all subsets of a given set of integers. Test Cases: List set = Arrays.asList(1, 2, 3); List<List> expected =
Arrays.asList(Arrays.asList(), Arrays.asList(1), Arrays.asList(2), Arrays.asList(3),

```
Arrays.asList(1, 2), Arrays.asList(1, 3), Arrays.asList(2, 3), Arrays.asList(1, 2, 3) );
assert generateSubsets(set).equals(expected); // true
```

Program:

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class SubsetGenerator {

    public static List<List<Integer>> generateSubsets(List<Integer> set) {
        // Helper method to handle recursion
        return generateSubsetsHelper(set, 0);
    }

    private static List<List<Integer>> generateSubsetsHelper(List<Integer> set, int
index) {
        List<List<Integer>> allSubsets;

        // Base case: If we've considered all elements, return a list with an empty subset
        if (index == set.size()) {
            allSubsets = new ArrayList<>();
            allSubsets.add(new ArrayList<>()); // The empty subset
        } else {
            // Recursive case
            int currentElement = set.get(index);
            allSubsets = generateSubsetsHelper(set, index + 1);

            // For each subset already found, create a new subset that includes the
current element
            List<List<Integer>> moreSubsets = new ArrayList<>();
            for (List<Integer> subset : allSubsets) {
                List<Integer> newSubset = new ArrayList<>(subset);
                newSubset.add(currentElement);
                moreSubsets.add(newSubset);
            }

            // Combine the subsets with and without the current element
            allSubsets.addAll(moreSubsets);
        }

        return allSubsets;
    }

    public static void main(String[] args) {
        List<Integer> set = Arrays.asList(1, 2, 3);
        List<List<Integer>> expected = Arrays.asList(
            Arrays.asList(),
            Arrays.asList(1),
            Arrays.asList(2),
            Arrays.asList(3),
            Arrays.asList(1, 2),
            Arrays.asList(1, 3),
            Arrays.asList(2, 3),
            Arrays.asList(1, 2, 3)
        );

        List<List<Integer>> result = generateSubsets(set);
        System.out.println(result.equals(expected)); // Should print true
    }
}
```

Output:



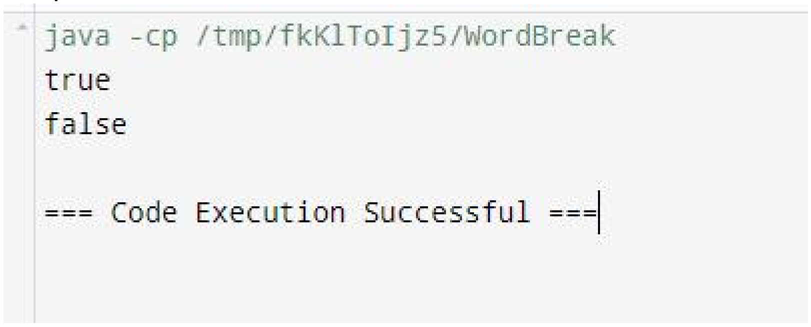
```
java -cp /tmp/HNUXuuXgSO/SubsetGenerator
false

=== Code Execution Successful ===
```

5. Write a recursive Java method to determine if a string can be segmented into a space-separated sequence of one or more dictionary words. Test Cases: Set wordDict = new HashSet<>(Arrays.asList("apple", "pen", "applepen", "pine", "pineapple")); assert wordBreak("pineapplepenapple", wordDict) == true; // true assert wordBreak("catsandog", wordDict) == false; // true

Program:

```
import java.util.Arrays;
import java.util.HashSet;
import java.util.Set;

public class WordBreak {

    public static void main(String[] args) {
        Set<String> wordDict = new HashSet<>(Arrays.asList("apple", "pen", "applepen",
"pine", "pineapple"));

        System.out.println(wordBreak("pineapplepenapple", wordDict)); // true
        System.out.println(wordBreak("catsandog", wordDict)); // false
    }

    public static boolean wordBreak(String s, Set<String> wordDict) {
        return wordBreakHelper(s, wordDict, 0);
    }

    private static boolean wordBreakHelper(String s, Set<String> wordDict, int start) {
        if (start == s.length()) {
          return true;
        }
        for (int end = start + 1; end <= s.length(); end++) {
          if (wordDict.contains(s.substring(start, end)) && wordBreakHelper(s, wordDict,
end)) {
            return true;
          }
        }
        return false;
    }
}
```

Output:



```
java -cp /tmp/fkKlToIjz5/WordBreak
true
false

=== Code Execution Successful ===
```