Day-1

1. Group Anagrams

Problem Description:

Given an array of strings, group the anagrams together. You can return the answer in any order.

Example:

Input: ["eat", "tea", "tan", "ate", "nat", "bat"]
Output: [["eat","tea","ate"], ["tan","nat"], ["bat"]]

Java Code:

```
import java.util.*;

public class GroupAnagrams {
   public static List<List<String>> groupAnagrams(String[] strs) {
      Map<String, List<String>> map = new HashMap<>();

      for (String s : strs) {
         char[] chars = s.toCharArray();
         Arrays.sort(chars);
         String sorted = new String(chars);

         map.computeIfAbsent(sorted, k -> new ArrayList<>()).add(s);
      }

      return new ArrayList<>(map.values());
   }

   public static void main(String[] args) {
      String[] input = {"eat", "tea", "tan", "ate", "nat", "bat"};
      List<List<String>> result = groupAnagrams(input);
      System.out.println(result);
   }
}
```

2. Longest Palindromic Substring

Problem Description:

Given a string s, return the **longest palindromic substring** in s.

Example:

Input: "babad"

Output: "bab" or "aba"

Java Code:

```
public class LongestPalindromeSubstring {
   public static String longestPalindrome(String s) {
      if (s == null || s.length() < 1) return "";

      int start = 0, end = 0;
      for (int i = 0; i < s.length(); i++) {
         int len1 = expandAroundCenter(s, i, i);      // Odd length
         int len2 = expandAroundCenter(s, i, i + 1);   // Even length
         int len = Math.max(len1, len2);

         if (len > end - start) {
            start = i - (len - 1) / 2;
            end = i + len / 2;
         }
      }

      return s.substring(start, end + 1);
   }

   private static int expandAroundCenter(String s, int left, int right) {
      while (left >= 0 && right < s.length() && s.charAt(left) == s.charAt(right)) {
         left--;
         right++;
      }
      return right - left - 1;
   }

   public static void main(String[] args) {
      String input = "babad";
      System.out.println("Longest Palindromic Substring: " + longestPalindrome(input));
   }
}
```

Day-2

1. Minimum Window Substring

Problem Description:

Given two strings s and t, return the **minimum window substring** of s such that every character in t (including duplicates) is included in the window. If no such substring exists, return the empty string "".

Example:

Input: s = "ADOBECODEBANC", t = "ABC"
Output: "BANC"

Java Code:

```
import java.util.*;

public class MinWindowSubstring {
   public static String minWindow(String s, String t) {
      if (s.length() < t.length()) return "";

      Map<Character, Integer> tMap = new HashMap<>();
      for (char c : t.toCharArray())
         tMap.put(c, tMap.getOrDefault(c, 0) + 1);

      int left = 0, right = 0, minLen = Integer.MAX_VALUE, start = 0;
      int count = t.length();

      while (right < s.length()) {
         char c = s.charAt(right++);
         if (tMap.containsKey(c)) {
            if (tMap.get(c) > 0) count--;
            tMap.put(c, tMap.get(c) - 1);
         }

         while (count == 0) {
            if (right - left < minLen) {
               minLen = right - left;
               start = left;
            }
            char lChar = s.charAt(left++);
            if (tMap.containsKey(lChar)) {
               tMap.put(lChar, tMap.get(lChar) + 1);
               if (tMap.get(lChar) > 0) count++;
```

```
            }
        }
    }

    return minLen == Integer.MAX_VALUE ? "" : s.substring(start, start + minLen);
  }

  public static void main(String[] args) {
    String s = "ADOBECODEBANC";
    String t = "ABC";
    System.out.println("Minimum window substring: " + minWindow(s, t));
  }
}
```

2. Encode and Decode Strings

Problem Description:

Design an algorithm to **encode a list of strings** into a single string and **decode** it back to the original list. The encoded string should be able to handle empty strings and special characters.

Example:

Input: ["hello", "world"]

Encoded: "5#hello5#world"
Decoded Output: ["hello", "world"]

Java Code:

```
import java.util.*;

public class Codec {
  // Encodes a list of strings to a single string.
  public static String encode(List<String> strs) {
    StringBuilder sb = new StringBuilder();
    for (String str : strs) {
      sb.append(str.length()).append('#').append(str);
    }
    return sb.toString();
  }

  // Decodes a single string to a list of strings.
  public static List<String> decode(String s) {
    List<String> result = new ArrayList<>();
    int i = 0;
    while (i < s.length()) {
      int j = i;
      while (s.charAt(j) != '#') j++;
      int length = Integer.parseInt(s.substring(i, j));
      i = j + 1;
      result.add(s.substring(i, i + length));
      i = i + length;
    }
```

```
        return result;
    }

    public static void main(String[] args) {
        List<String> input = Arrays.asList("hello", "world");
        String encoded = encode(input);
        System.out.println("Encoded: " + encoded);
        System.out.println("Decoded: " + decode(encoded));
    }
}
```

Day-3

1. Multiply Strings

Problem Description:

Given two non-negative integers `num1` and `num2` represented as strings, return the product of `num1` and `num2`, also represented as a string. You must not use any built-in BigInteger library or directly convert the inputs to integers.

Example:

Input: num1 = "123", num2 = "456"
Output: "56088"

Java Code:

```
public class MultiplyStrings {
    public static String multiply(String num1, String num2) {
        int m = num1.length(), n = num2.length();
        int[] product = new int[m + n];

        for (int i = m - 1; i >= 0; i--) {
            int d1 = num1.charAt(i) - '0';
            for (int j = n - 1; j >= 0; j--) {
                int d2 = num2.charAt(j) - '0';
                int sum = d1 * d2 + product[i + j + 1];
                product[i + j + 1] = sum % 10;
                product[i + j] += sum / 10;
            }
        }

        StringBuilder sb = new StringBuilder();
        for (int p : product) {
            if (!(sb.length() == 0 && p == 0)) sb.append(p);
        }
```

```
    return sb.length() == 0 ? "0" : sb.toString();
  }

  public static void main(String[] args) {
    String num1 = "123";
    String num2 = "456";
    System.out.println("Product: " + multiply(num1, num2));
  }
}
```

2. Zigzag Conversion

Problem Description:

The string "PAYPALISHIRING" is written in a zigzag pattern on a given number of rows like this:

```
P   A   H   N
A P L S I I G
Y   I   R
```

Then read line by line: "PAHNAPLSIIGYIR"

Write the code that will take a string and make this conversion given a number of rows.

Example:

```
Input: s = "PAYPALISHIRING", numRows = 3
Output: "PAHNAPLSIIGYIR"
```

Java Code:

```
public class ZigzagConversion {
  public static String convert(String s, int numRows) {
    if (numRows == 1 || s.length() <= numRows) return s;

    StringBuilder[] rows = new StringBuilder[numRows];
    for (int i = 0; i < numRows; i++)
      rows[i] = new StringBuilder();

    int i = 0;
    boolean goingDown = false;

    for (char c : s.toCharArray()) {
      rows[i].append(c);
      if (i == 0 || i == numRows - 1) goingDown = !goingDown;
      i += goingDown ? 1 : -1;
    }

    StringBuilder result = new StringBuilder();
```

```
    for (StringBuilder row : rows)
      result.append(row);

    return result.toString();
  }

  public static void main(String[] args) {
    String input = "PAYPALISHIRING";
    int rows = 3;
    System.out.println("Zigzag Conversion: " + convert(input, rows));
  }
}
```

Day-4

14. String Compression

Problem Description:

Given an array of characters, compress it **in-place**. The length after compression should be returned. The array should be modified in place so that every character followed by its count (if > 1) replaces the original sequence.

Example:

Input: ['a','a','b','b','c','c','c']
Output: 6
Modified Input: ['a','2','b','2','c','3']

Java Code:

```
public class StringCompression {
  public static int compress(char[] chars) {
    int index = 0, i = 0;

    while (i < chars.length) {
      char currentChar = chars[i];
      int count = 0;

      while (i < chars.length && chars[i] == currentChar) {
        i++;
        count++;
      }

      chars[index++] = currentChar;
      if (count > 1) {
        for (char c : String.valueOf(count).toCharArray()) {
          chars[index++] = c;
        }
      }
    }

    return index;
```

```
    }

    public static void main(String[] args) {
        char[] input = {'a','a','b','b','c','c','c'};
        int newLength = compress(input);

        System.out.print("Compressed: ");
        for (int i = 0; i < newLength; i++) {
            System.out.print(input[i]);
        }
        System.out.println("\nLength: " + newLength);
    }
}
```

2. Count Distinct Substrings

Problem Statement:
Given a string S, return the **total number of distinct substrings** of S.
(This includes the empty substring as well.)

Example:

Input: "ab"
Output: 4
Explanation: "", "a", "b", "ab"

Approaches:

1. Using Trie (Prefix Tree)

The idea is:

- Insert all **suffixes** of the string into a Trie.
- The number of **nodes created** in the Trie = number of **distinct substrings**.

Java Code (Using Trie):

```
import java.util.*;

class TrieNode {
    TrieNode[] children = new TrieNode[26];
}

public class DistinctSubstrings {
    public static int countDistinctSubstrings(String s) {
        TrieNode root = new TrieNode();
        int count = 0;

        // Insert all suffixes
        for (int i = 0; i < s.length(); i++) {
```

```
        TrieNode node = root;
        for (int j = i; j < s.length(); j++) {
            int index = s.charAt(j) - 'a';
            if (node.children[index] == null) {
                node.children[index] = new TrieNode();
                count++; // New node = new substring
            }
            node = node.children[index];
        }
    }

    return count + 1; // Include empty substring
  }

  public static void main(String[] args) {
    String s = "ab";
    System.out.println("Distinct substrings count: " + countDistinctSubstrings(s));
  }
}
```