Day-1

1. Reverse a String

Problem Description:

Write a function to reverse a string. The function should take a string as input and return the reversed string.

Example:

Input: "hello"
Output: "olleh"

Java Code:

```
public class ReverseString {
   public static String reverse(String str) {
      StringBuilder reversed = new StringBuilder(str);
      return reversed.reverse().toString();
   }

   public static void main(String[] args) {
      String input = "hello";
      String output = reverse(input);
      System.out.println("Reversed String: " + output);
   }
}
```

2. Check for Anagram

Problem Description:

Given two strings, check whether they are anagrams of each other. Two strings are anagrams if they contain the same characters in the same frequency, but possibly in different order.

Example:

Input: str1 = "listen", str2 = "silent"
Output: true

Java Code:

```
import java.util.Arrays;

public class AnagramCheck {
```

```
    public static boolean isAnagram(String str1, String str2) {
        if (str1.length() != str2.length()) return false;

        char[] a1 = str1.toCharArray();
        char[] a2 = str2.toCharArray();
        Arrays.sort(a1);
        Arrays.sort(a2);

        return Arrays.equals(a1, a2);
    }

    public static void main(String[] args) {
        String str1 = "listen";
        String str2 = "silent";
        System.out.println("Are anagrams? " + isAnagram(str1, str2));
    }
}
```

3. Palindrome Check

Problem Description:

Check whether a given string is a palindrome. A string is a palindrome if it reads the same backward as forward.

Example:

Input: "racecar"
Output: true

Java Code:

```
public class PalindromeCheck {
    public static boolean isPalindrome(String str) {
        int left = 0;
        int right = str.length() - 1;

        while (left < right) {
            if (str.charAt(left) != str.charAt(right)) {
                return false;
            }
            left++;
            right--;
        }
        return true;
    }

    public static void main(String[] args) {
        String input = "racecar";
        System.out.println("Is Palindrome? " + isPalindrome(input));
    }
}
```

Day-2

1. Valid Palindrome (Ignore Case & Non-Alphanumerics)

Problem Description:

Given a string, determine if it is a palindrome considering **only alphanumeric characters** and **ignoring cases**.

Example:

Input: "A man, a plan, a canal: Panama"
Output: true

Java Code:

```
public class ValidPalindrome {
   public static boolean isPalindrome(String s) {
      int left = 0, right = s.length() - 1;

      while (left < right) {
         while (left < right && !Character.isLetterOrDigit(s.charAt(left))) left++;
         while (left < right && !Character.isLetterOrDigit(s.charAt(right))) right--;

         if (Character.toLowerCase(s.charAt(left)) != Character.toLowerCase(s.charAt(right))) {
            return false;
         }
         left++;
         right--;
      }
      return true;
   }

   public static void main(String[] args) {
      String input = "A man, a plan, a canal: Panama";
      System.out.println("Is valid palindrome? " + isPalindrome(input));
   }
}
```

2. Implement strStr() (Substring Search)

Problem Description:

Implement strStr() which returns the **index of the first occurrence** of needle in haystack, or -1 if needle is not part of haystack.

Example:

Input: haystack = "hello", needle = "ll"
Output: 2

Java Code:

```
public class StrStr {
   public static int strStr(String haystack, String needle) {
      if (needle.isEmpty()) return 0;
      return haystack.indexOf(needle);
   }

   public static void main(String[] args) {
      String haystack = "hello";
      String needle = "ll";
      System.out.println("Index of first occurrence: " + strStr(haystack, needle));
   }
}
```

3. Count Vowels and Consonants

Problem Description:

Given a string, count the number of vowels and consonants in it. Consider only alphabets and ignore case.

Example:

Input: "Hello World!"

Output: Vowels = 3, Consonants = 7

Java Code:

```
public class VowelConsonantCounter {
   public static void countVowelsAndConsonants(String str) {
      int vowels = 0, consonants = 0;
      str = str.toLowerCase();

      for (char c : str.toCharArray()) {
         if (Character.isLetter(c)) {
            if ("aeiou".indexOf(c) != -1) vowels++;
            else consonants++;
         }
      } System.out.println("Vowels: " + vowels);
      System.out.println("Consonants: " + consonants);
   }

   public static void main(String[] args) {
      String input = "Hello World!";
      countVowelsAndConsonants(input);
   }}
```

Day-3

1. Remove Duplicates from String

Problem Description:

Given a string, remove all duplicate characters and return the resulting string with only the first occurrences of each character, preserving the original order.

Example:

Input: "programming"
Output: "progamin"

Java Code:

```
import java.util.LinkedHashSet;

public class RemoveDuplicates {
   public static String removeDuplicates(String str) {
      LinkedHashSet<Character> set = new LinkedHashSet<>();
      for (char c : str.toCharArray()) {
         set.add(c);
      }

      StringBuilder sb = new StringBuilder();
      for (char c : set) {
         sb.append(c);
      }

      return sb.toString();
   }

   public static void main(String[] args) {
      String input = "programming";
      System.out.println("After removing duplicates: " + removeDuplicates(input));
   }
}
```

2. Longest Substring Without Repeating Characters

Problem Description:

Given a string s, find the length of the **longest substring** without repeating characters.

Example:

Input: "abcabcbb"
Output: 3
Explanation: The answer is "abc", with length 3.

Java Code:

```
import java.util.HashSet;

public class LongestUniqueSubstring {
   public static int lengthOfLongestSubstring(String s) {
      int left = 0, right = 0, maxLen = 0;
      HashSet<Character> seen = new HashSet<>();

      while (right < s.length()) {
         if (!seen.contains(s.charAt(right))) {
            seen.add(s.charAt(right));
            maxLen = Math.max(maxLen, right - left + 1);
            right++;
         } else {
            seen.remove(s.charAt(left));
            left++;
         }
      }

      return maxLen;
   }

   public static void main(String[] args) {
      String input = "abcabcbb";
      System.out.println("Length of longest substring: " + lengthOfLongestSubstring(input));
   }
}
```

Day-4

2. Toggle Case of Characters

Problem Description:

Given a string, toggle the case of each character (lowercase characters become uppercase and vice versa).

Example:

Input: "Hello World"
Output: "hELLO wORLD"

Java Code:

```
public class ToggleCase {
   public static String toggleCase(String str) {
      StringBuilder sb = new StringBuilder();
```

```
    for (char c : str.toCharArray()) {
      if (Character.isUpperCase(c)) {
        sb.append(Character.toLowerCase(c));
      } else if (Character.isLowerCase(c)) {
        sb.append(Character.toUpperCase(c));
      } else {
        sb.append(c);
      }
    }

    return sb.toString();
  }

  public static void main(String[] args) {
    String input = "Hello World";
    System.out.println("Toggled case: " + toggleCase(input));
  }
}
```

2. 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);
  }
}
```