Day-1

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. 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));
    }
}
```

3. Frequency of Characters

Problem Description:

Given a string, find the frequency of each character and print the result.

Example:

```
Input: "banana"
Output:
b: 1
a: 3
n: 2
```

Java Code:

```
import java.util.HashMap;
import java.util.Map;

public class FrequencyOfCharacters {
    public static void frequency(String str) {
        Map<Character, Integer> freqMap = new HashMap<>();

        for (char c : str.toCharArray()) {
            freqMap.put(c, freqMap.getOrDefault(c, 0) + 1);
        }

        for (Map.Entry<Character, Integer> entry : freqMap.entrySet()) {
            System.out.println(entry.getKey() + ": " + entry.getValue());
        }
    }

    public static void main(String[] args) {
        String input = "banana";
        frequency(input);
    }
}
```

Day-2

1. First Non-Repeating Character

Problem Description:

Given a string, find the **first non-repeating character** and return its index. If it doesn't exist, return `-1`.

Example:

```
Input: "leetcode"
Output: 0
Explanation: 'l' is the first non-repeating character.
```

Java Code:

```
import java.util.*;

public class FirstNonRepeatingChar {
    public static int firstUniqChar(String s) {
        int[] freq = new int[26];
        for (char c : s.toCharArray()) {
            freq[c - 'a']++;
        }

        for (int i = 0; i < s.length(); i++) {
            if (freq[s.charAt(i) - 'a'] == 1) return i;
        }

        return -1;
    }

    public static void main(String[] args) {
        String input = "leetcode";
        int index = firstUniqChar(input);
        System.out.println("First non-repeating character index: " + index);
    }
}
```

2. Check for Subsequence

Problem Description:

Given two strings `s` and `t`, check if `s` is a **subsequence** of `t`.

A subsequence is a sequence that can be derived from another string by deleting some characters without changing the order of the remaining characters.

Example:

```
Input: s = "abc", t = "ahbgdc"
Output: true
```

Java Code:

```java
public class IsSubsequence {
    public static boolean isSubsequence(String s, String t) {
        int i = 0, j = 0;

        while (i < s.length() && j < t.length()) {
            if (s.charAt(i) == t.charAt(j)) {
                i++;
            }
            j++;
        }

        return i == s.length();
    }

    public static void main(String[] args) {
        String s = "abc";
        String t = "ahbgdc";
        System.out.println("Is subsequence: " + isSubsequence(s, t));
    }
}
```

3. Remove Spaces from a String

Problem Description:

Given a string, remove all spaces from it and return the modified string.

Example:

```
Input: "  Hello   World   "
Output: "HelloWorld"
```

Java Code:

```
public class RemoveSpaces {
    public static String removeSpaces(String str) {
        return str.replaceAll("\\s+", "");
    }

    public static void main(String[] args) {
        String input = "  Hello   World  ";
        System.out.println("After removing spaces: " + removeSpaces(input));
    }
}
```

Day-3

1. Check if Two Strings are Isomorphic

Problem Description:

Two strings `s` and `t` are **isomorphic** if the characters in `s` can be replaced to get `t`, preserving the order of characters and maintaining a one-to-one mapping.

Example:

```
Input: s = "egg", t = "add"
Output: true
```

Java Code:

```
import java.util.HashMap;

public class IsomorphicStrings {
    public static boolean isIsomorphic(String s, String t) {
        if (s.length() != t.length()) return false;

        HashMap<Character, Character> mapS = new HashMap<>();
        HashMap<Character, Character> mapT = new HashMap<>();

        for (int i = 0; i < s.length(); i++) {
            char cs = s.charAt(i);
            char ct = t.charAt(i);

            if (mapS.containsKey(cs) && mapS.get(cs) != ct) return false;
            if (mapT.containsKey(ct) && mapT.get(ct) != cs) return false;

            mapS.put(cs, ct);
            mapT.put(ct, cs);
        }

        return true;
    }

    public static void main(String[] args) {
        System.out.println(isIsomorphic("egg", "add")); // true
        System.out.println(isIsomorphic("foo", "bar")); // false
    }
}
```

2. Check if Two Strings are Rotations of Each Other

Problem Description:

Given two strings, check if one string is a **rotation** of the other.

Example:

```
Input: s1 = "waterbottle", s2 = "erbottlewat"
Output: true
```

Java Code:

```
public class RotationCheck {
    public static boolean areRotations(String s1, String s2) {
        return s1.length() == s2.length() && (s1 + s1).contains(s2);
    }

    public static void main(String[] args) {
        String s1 = "waterbottle";
        String s2 = "erbottlewat";
        System.out.println("Are rotations: " + areRotations(s1, s2)); // true
    }
}
```

Day-4

1. Count Words in a String

Problem Description:

Given a string, count the number of **words** (separated by one or more spaces).

Example:

```
Input: "  Java is  awesome  "
Output: 3
```

Java Code:

```
public class WordCount {
    public static int countWords(String str) {
        if (str == null || str.trim().isEmpty()) return 0;
        return str.trim().split("\\s+").length;
    }

    public static void main(String[] args) {
        String input = "  Java is  awesome  ";
        System.out.println("Word count: " + countWords(input)); // 3
    }
}
```

2. Replace Spaces with `%20` (URLify)

Problem Description:

Write a method to replace all spaces in a string with `%20`. Assume the string has sufficient space at the end to hold the additional characters, and that you're given the "true" length of the string.

Example:

```
Input: "Mr John Smith    ", trueLength = 13
Output: "Mr%20John%20Smith"
```

Java Code:

```
public class URLify {
    public static String replaceSpaces(String str, int trueLength) {
        char[] chars = str.toCharArray();
        StringBuilder sb = new StringBuilder();

        for (int i = 0; i < trueLength; i++) {
            if (chars[i] == ' ') {
                sb.append("%20");
            } else {
                sb.append(chars[i]);
            }
        }

        return sb.toString();
    }

    public static void main(String[] args) {
        String input = "Mr John Smith    ";
        int trueLength = 13;
        System.out.println("URLified: " + replaceSpaces(input, trueLength));
    }
}
```