- 1. Reverse an Array: Swaps elements from the start and end moving towards the center to reverse the array.

```
public class ReverseArray {
    public static void main(String[] args) {
        int[] array = {1, 2, 3, 4, 5};
        reverseArray(array);
        for (int num : array) {
            System.out.print(num + " ");
        }
    }

    public static void reverseArray(int[] array) {
        int left = 0, right = array.length - 1;
        while (left < right) {
            int temp = array[left];
            array[left] = array[right];
            array[right] = temp;
            left++;
            right--;
        }
    }
}
```

- 2. Check if a String is a Palindrome: Compares characters from the start and end moving towards the center to check for equality.

```
public class PalindromeCheck {
    public static void main(String[] args) {
        String str = "madam";
        boolean isPalindrome = isPalindrome(str);
        System.out.println("Is the string a palindrome? " +
isPalindrome);
    }

    public static boolean isPalindrome(String str) {
        int left = 0, right = str.length() - 1;
        while (left < right) {
            if (str.charAt(left) != str.charAt(right)) {
                return false;
            }
            left++;
            right--;
        }
        return true;
    }
}
```

- 3. Remove Duplicates from an Array: Uses a two-pointer technique to remove duplicates from a sorted array.

```
import java.util.Arrays;

public class RemoveDuplicates {
    public static void main(String[] args) {
        int[] array = {1, 1, 2, 2, 3, 4, 4, 5};
        int newLength = removeDuplicates(array);
        System.out.println("Array after removing duplicates: " +
Arrays.toString(Arrays.copyOf(array, newLength)));
    }

    public static int removeDuplicates(int[] array) {
        if (array.length == 0) return 0;

        int j = 0;
        for (int i = 1; i < array.length; i++) {
            if (array[i] != array[j]) {
                j++;
                array[j] = array[i];
            }
        }
        return j + 1;
    }
}
```

- 4. Find the First Non-Repeating Character in a String: Uses a linked hash map to count character occurrences and finds the first character with a count of one.

```
import java.util.LinkedHashMap;
import java.util.Map;

public class FirstNonRepeatingChar {
    public static void main(String[] args) {
        String str = "swiss";
        char result = firstNonRepeatingChar(str);
        System.out.println("First non-repeating character is: "
+ result);
    }

    public static char firstNonRepeatingChar(String str) {
        Map charCountMap = new LinkedHashMap<>();

        for (char c : str.toCharArray()) {
```

```
            charCountMap.put(c, charCountMap.getOrDefault(c, 0)
+ 1);
        }

        for (Map.Entry entry : charCountMap.entrySet()) {
            if (entry.getValue() == 1) {
                return entry.getKey();
            }
        }

        return '\0'; // Indicates no non-repeating character
found
    }
}
```