LAB # 04

OBJECTIVE

To understand arrays and its memory allocation.

LAB TASK

1. Write a program that takes two arrays of size 4 and swap the elements of those arrays.

Main.java Share Run Output

```
public class ArraySwap {
    // Method to print array
    public static void printArray(int[] array) {
        for (int value : array) {
            System.out.print(value + " ");
        }
        System.out.println();
    }

    public static void main(String[] args) {
        // Initialize two arrays of size 4
        int[] array1 = {1, 2, 3, 4};
        int[] array2 = {5, 6, 7, 8};

        // Display original arrays
        System.out.println("Original Array 1:");
        printArray(array1);
        System.out.println("Original Array 2:");
        printArray(array2);

        // Swap elements of array1 and array2
        for (int i = 0; i < 4; i++) {
            int temp = array1[i];
            array1[i] = array2[i];
            array2[i] = temp;
        }

        // Display arrays after swapping
        System.out.println("Array 1 after swap:");
        printArray(array1);
        System.out.println("Array 2 after swap:");
        printArray(array2);
    }
}
```

```
java -cp /tmp/YrtIpXxrq2/ArraySwap
Original Array 1:
1 2 3 4
Original Array 2:
5 6 7 8
Array 1 after swap:
5 6 7 8
Array 2 after swap:
1 2 3 4

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

2. Add a method in the class that takes array and merge it with the existing one.

Programiz Online Java Compiler

Main.java

```java
import java.util.Arrays;

public class ArrayMerger {
    private int[] existingArray;

    public ArrayMerger(int[] initialArray) {
        this.existingArray = initialArray;
    }

    public void mergeArray(int[] newArray) {
        int[] mergedArray = Arrays.copyOf(existingArray, existingArray.length + newArray
            .length);
        System.arraycopy(newArray, 0, mergedArray, existingArray.length, newArray.length
            );
        existingArray = mergedArray;
    }

    public void displayArray() {
        System.out.println(Arrays.toString(existingArray));
    }

    public static void main(String[] args) {
        ArrayMerger arrayMerger = new ArrayMerger(new int[]{9, 8, 7});
        System.out.print("Existing Array: ");
        arrayMerger.displayArray();

        arrayMerger.mergeArray(new int[]{6, 5, 4});
        System.out.print("Merged Array: ");
        arrayMerger.displayArray();
    }
}
```

Output

```
java -cp /tmp/3aOM6bUyah/ArrayMerg
Existing Array: [9, 8, 7]
Merged Array: [9, 8, 7, 6, 5, 4]

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

3. In a JAVA program, take an array of type string and then check whether the strings are palindrome or not.

Main.java

```java
public class PalindromeCheck {
    public static void main(String[] args) {
        // Define an array of strings
        String[] words = {"noon", "deed", "hello", "level", "wow"};

        // Check each string in the array
        for (String word : words) {
            if (isPalindrome(word)) {
                System.out.println(word + " is a palindrome.");
            } else {
                System.out.println(word + " is not a palindrome.");
            }
        }
    }

    // Method to check if a string is a palindrome
    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;
    }
}
```

Output

```
java -cp /tmp/3vQ8rBMGJM/PalindromeCheck
noon is a palindrome.
deed is a palindrome.
hello is not a palindrome.
level is a palindrome.
wow is a palindrome.

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

4. Given an array of integers, count how many numbers are even and how many are odd.

Main.java

```java
public class EvenOddCount {
    public static void main(String[] args) {
        // Define an array of integers
        int[] numbers = {11, 24, 37, 43, 52, 66, 78, 89, 99, 10};

        // Variables to count even and odd numbers
        int evenCount = 0;
        int oddCount = 0;

        // Loop through each number in the array
        for (int num : numbers) {
            if (num % 2 == 0) {
                evenCount++;
            } else {
                oddCount++;
            }
        }

        // Print the results
        System.out.println("Even numbers: " + evenCount);
        System.out.println("Odd numbers: " + oddCount);
    }
```

Output

```
java -cp /tmp/A89ff74N8L/EvenOddCount
Even numbers: 5
Odd numbers: 5

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

5. Given two integer arrays, merge them and remove any duplicate values from the resulting array.

Main.java

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

public class MergeAndRemoveDuplicates {
    public static void main(String[] args) {
        // Initialize two arrays of integers
        int[] array1 = {1, 2, 3, 4, 5};
        int[] array2 = {4, 5, 6, 7, 8};

        // Merge the arrays and eliminate any duplicate values
        int[] mergedArray = mergeAndRemoveDuplicates(array1, array2);

        // Display the final merged array without duplicates
        System.out.println("Merged array without duplicates: " + Arrays.toString
            (mergedArray));
    }

    public static int[] mergeAndRemoveDuplicates(int[] arr1, int[] arr2) {
        // Use a HashSet to collect elements, automatically discarding duplicates
        HashSet<Integer> set = new HashSet<>();

        // Insert all elements from the first array into the set
        for (int num : arr1) {
            set.add(num);
        }

        // Insert all elements from the second array into the set
        for (int num : arr2) {
            set.add(num);
        }

        // Convert the set back into an array form
        int[] result = new int[set.size()];
        int index = 0;
        for (int num : set) {
            result[index++] = num;
```

Output

```
java -cp /tmp/uiwOjRwWZY/MergeAndRemoveDuplicates
Merged array without duplicates: [1, 2, 3, 4, 5, 6, 7, 8]

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

HOME TASKs

1. Write a program that takes an array of Real numbers having size 7 and calculate the sum and mean of all the elements. Also depict the memory management of this task.

Main.java Share Run Output

```java
public class ArraySumAndMean {
    public static void main(String[] args) {
        // Declare and initialize an array of 7 real numbers
        double[] numbers = {2.5, 3.1, 4.7, 6.0, 1.8, 9.3, 7.2};

        // Variables to store the sum and mean of the array elements
        double sum = 0;
        double mean;

        // Calculate the sum of the elements in the array
        for (double num : numbers) {
            sum += num;  // Accumulate each number into sum
        }

        // Calculate the mean by dividing the sum by the number of
            elements
        mean = sum / numbers.length;

        // Print the results
        System.out.println("Sum of elements: " + sum);
        System.out.println("Mean of elements: " + mean);
    }
```

```
java -cp /tmp/uiwOjRwWZY/MergeAndRemoveDuplicates
Merged array without duplicates: [1, 2, 3, 4, 5, 6, 7, 8]

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

2. Add a method in the same class that splits the existing array into two. The method should search a key in array and if found splits the array from that index of the key.

Main.java

Share Run

```java
public class ArraySplitter {
    // Method to split the array
    public static int[][] splitArray(int[] arr, int key) {
        int index = -1;

        // Find the index of the key in the array
        for (int i = 0; i < arr.length; i++) {
            if (arr[i] == key) {
                index = i;
                break;
            }
        }

        // If the key is found
        if (index != -1) {
            int[] firstPart = new int[index + 1];
            int[] secondPart = new int[arr.length - index - 1];

            // Fill the first part with elements before and including the key
            for (int i = 0; i <= index; i++) {
                firstPart[i] = arr[i];
            }

            // Fill the second part with elements after the key
            for (int i = index + 1; i < arr.length; i++) {
                secondPart[i - index - 1] = arr[i];
            }

            return new int[][]{firstPart, secondPart};
        } else {
            // If key is not found, return the original array
            return new int[][]{arr};
        }
    }

    public static void main(String[] args) {
        int[] arr = {1, 2, 3, 4, 5, 6};
        int key = 3;

        int[][] result = splitArray(arr, key);

        // Print the first part
        System.out.println("First part:");
        for (int i : result[0]) {
            System.out.print(i + " ");
        }

        // Print the second part (if exists)
        System.out.println("\nSecond part:");
        if (result.length > 1) {
            for (int i : result[1]) {
                System.out.print(i + " ");
            }
        } else {
            System.out.println("Key not found, array is not split.");
        }
    }
}
```

Output

```
java -cp /tmp/anyR5p5X53/ArraySplitter
First part:
1 2 3
Second part:
4 5 6
=== Code Execution Successful ===
```

3. Given an array of distinct integers and a target integer, return all unique combinations of numbers that add up to the target. Each number can be used only once in the combination.

Main.java

Share Run

Output

```java
import java.util.*;

public class CombinationSum {
    // Function to find all unique combinations
    public static List<List<Integer>> combinationSum(int[] candidates, int target) {
        List<List<Integer>> result = new ArrayList<>();
        List<Integer> currentCombination = new ArrayList<>();
        Arrays.sort(candidates); // Sort to handle duplicates (though we don't have duplicates here)
        backtrack(candidates, target, 0, currentCombination, result);
        return result;
    }

    // Backtracking helper function
    private static void backtrack(int[] candidates, int target, int start,
                                  List<Integer> currentCombination, List<List<Integer>> result) {
        // Base case: if the target becomes 0, we found a valid combination
        if (target == 0) {
            result.add(new ArrayList<>(currentCombination));
            return;
        }

        // Loop through the candidates array
        for (int i = start; i < candidates.length; i++) {
            // If the current number is greater than the target, break as further numbers will be too large
            if (candidates[i] > target) {
                break;
            }
            // Add the current number to the combination
            currentCombination.add(candidates[i]);
            // Recurse with the reduced target and next start index (i + 1 to ensure uniqueness)
            backtrack(candidates, target - candidates[i], i + 1, currentCombination, result);
            // Backtrack by removing the last added number
            currentCombination.remove(currentCombination.size() - 1);
        }
    }

    public static void main(String[] args) {
        int[] candidates = {2, 3, 6, 7};
        int target = 7;

        List<List<Integer>> combinations = combinationSum(candidates, target);

        // Print all unique combinations
        System.out.println("Unique combinations that sum up to " + target + ":");
        for (List<Integer> combination : combinations) {
            System.out.println(combination);
```

```
java -cp /tmp/svGWNkmx2M/CombinationSum
Unique combinations that sum up to 7:
[7]

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

4. You are given an array containing n distinct numbers taken from 0, 1, 2, ..., n. Write a program to find the one number that is missing from the array.

Main.java

Share Run

Output

```java
public class MissingNumber {
    public static int findMissingNumber(int[] nums) {
        int n = nums.length;

        // Calculate the sum of numbers from 0 to n
        int expectedSum = n * (n + 1) / 2;

        // Calculate the sum of elements in the array
        int actualSum = 0;
        for (int num : nums) {
            actualSum += num;
        }

        // The missing number is the difference between the expected
            sum and actual sum
        return expectedSum - actualSum;
    }

    public static void main(String[] args) {
        int[] nums = {0,3,2};  // Example input array
        System.out.println("The missing number is: " +
            findMissingNumber(nums));
```

```
java -cp /tmp/4VxBdpAKtF/MissingNumber
The missing number is: 1

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

5. You are given an array of integers. Write a program to sort the array such that it follows a zigzag pattern: the first element is less than the second, the second is greater than the third, and so on.

Main.java

```java
import java.util.Arrays;

public class ZigzagSort {
    public static void zigzagSort(int[] arr) {
        // Sort the array first
        Arrays.sort(arr);

        // Swap adjacent elements to get the zigzag pattern
        for (int i = 1; i < arr.length; i += 2) {
            // Swap the elements at positions i-1 and i
            if (i < arr.length) {
                int temp = arr[i];
                arr[i] = arr[i - 1];
                arr[i - 1] = temp;
            }
        }
    }

    public static void main(String[] args) {
        int[] arr = {1,2,3,4,5,6,7,8,9,10};

        System.out.println("Original Array: " + Arrays.toString(arr));

        zigzagSort(arr);  // Sort the array in zigzag pattern

        System.out.println("Zigzag Sorted Array: " + Arrays.toString(arr));
```

Output

```
java -cp /tmp/uZ1gSj82bP/ZigzagSort
Original Array: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Zigzag Sorted Array: [2, 1, 4, 3, 6, 5, 8, 7, 10, 9]

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