LAB # 04

ARRAYS IN JAVA

OBJECTIVE: To understand arrays and its memory allocation.

LAB TASKS

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

```
public class SwapArrays {
    public static void main(String[] args) {
        int[] array1 = {101, 4, 101, 4};
        int[] array2 = {4, 101, 4, 101};

        for (int i = 0; i < array1.length; i++) {
            int temp = array1[i];
            array1[i] = array2[i];
            array2[i] = temp;
        }

        System.out.println("Array 1: " + java.util.Arrays.toString(array1));
        System.out.println("Array 2: " + java.util.Arrays.toString(array2));
    }
}
```

OUTPUT:

```
Array 1: [4, 101, 4, 101]
Array 2: [101, 4, 101, 4]
```

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

```
public class MergeArrays {
    public static void main(String[] args) {
        int[] array1 = {101, 4, 101, 4};
        int[] array2 = {4, 101, 4, 101};

        int[] mergedArray = new int[array1.length + array2.length];
        System.arraycopy(array1, 0, mergedArray, 0, array1.length);
        System.arraycopy(array2, 0, mergedArray, array1.length, array2.length);

        System.out.println("Merged Array: " + java.util.Arrays.toString(mergedArray));
    }
}
```

OUTPUT:

```
Merged Array: [101, 4, 101, 4, 4, 101, 4, 101]
```

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

```
public class PalindromeCheck {
    public static void main(String[] args) {
        String[] words = {"Affan", "radar", "madam", "level"};
        for (String word : words) {
            String result = word.equalsIgnoreCase(new StringBuilder(word).reverse().toString()) ?
                    "is a palindrome." : "is not a palindrome.";
            System.out.println(word + " " + result);
        }
    }
}
```

OUTPUT:

```
run:
Affan is not a palindrome.
radar is a palindrome.
madam is a palindrome.
level is a palindrome.
BUILD SUCCESSFUL (total time: 3 seconds)
```

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

```
public class EvenOddCount {
    public static void main(String[] args) {
        int[] numbers = {101, 4, 101, 4, 101};
        int evenCount = 0, oddCount = 0;

        for (int num : numbers) {
            if (num % 2 == 0) evenCount++;
            else oddCount++;
        }

        System.out.println("Even count: " + evenCount);
        System.out.println("Odd count: " + oddCount);
    }
}
```

OUTPUT:

```
Even count: 2
Odd count: 3
```

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

```
import java.util.HashSet;
import java.util.Set;

public class MergeAndRemoveDuplicates {
    public static void main(String[] args) {
        int[] array1 = {101, 4, 101, 4};
        int[] array2 = {4, 101, 4, 101};

        Set<Integer> uniqueNumbers = new HashSet<>();
        for (int num : array1) uniqueNumbers.add(num);
        for (int num : array2) uniqueNumbers.add(num);

        int[] resultArray = uniqueNumbers.stream().mapToInt(Integer::intValue).toArray();
        System.out.println("Merged Array without Duplicates: " + java.util.Arrays.toString(resultArray));
    }
}
```

OUTPUT:

```
Merged Array without Duplicates: [101, 4]
```

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.

```
public class ArraySumMean {
    public static void main(String[] args) {
        double[] numbers = {2.5, 3.0, 4.5, 5.5, 1.0, 6.5, 7.0};
        double sum = 0;

        for (double num : numbers) sum += num;
        double mean = sum / numbers.length;

        System.out.println("Sum: " + sum);
        System.out.println("Mean: " + mean);
    }
}
```

OUTPUT:

```
Sum: 30.0
Mean: 4.285714285714286
```

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.

```
import java.util.Arrays;
public class ArraySplitter {
    public static void splitArray(int[] array, int key) {
        int index = -1;
        for (int i = 0; i < array.length; i++) {
            if (array[i] == key) {
                index = i;
                break;
            }
        }

        if (index != -1) {
            System.out.println("Before key: " + Arrays.toString(Arrays.copyOfRange(array, 0, index)));
            System.out.println("From key: " + Arrays.toString(Arrays.copyOfRange(array, index, array.length)));
        } else {
            System.out.println("Key not found");
        }
    }
    public static void main(String[] args) {
        int[] array = {5, 10, 15, 20, 25, 30};
        splitArray(array, 20);
    }
}
```

OUTPUT:

```
Before key: [5, 10, 15]
From key: [20, 25, 30]
```

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.

```
import java.util.*;

public class CombinationSum {
    public static List<List<Integer>> combinationSum(int[] candidates, int target) {
        List<List<Integer>> result = new ArrayList<>();
        backtrack(candidates, target, 0, new ArrayList<>(), result);
        return result;
    }

    private static void backtrack(int[] candidates, int target, int start, List<Integer> current, List<List<Integer>> result) {
        if (target == 0) {
            result.add(new ArrayList<>(current));
            return;
        }
        for (int i = start; i < candidates.length; i++) {
            if (candidates[i] > target) break;
            current.add(candidates[i]);
            backtrack(candidates, target - candidates[i], i + 1, current, result);
            current.remove(current.size() - 1);
        }
    }

    public static void main(String[] args) {
        int[] candidates = {2, 3, 6, 7};
        int target = 7;
        System.out.println(combinationSum(candidates, target));
    }
}
```

OUTPUT:

```
[[2, 2, 3], [7]]
```

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.

```
public class MissingNumber {
    public static int findMissingNumber(int[] nums) {
        int n = nums.length;
        int sum = (n * (n + 1)) / 2;
        int actualSum = 0;
        for (int num : nums) actualSum += num;
        return sum - actualSum;
    }

    public static void main(String[] args) {
        int[] nums = {3, 0, 1};
        System.out.println(findMissingNumber(nums));
    }
}
```

OUTPUT: 2

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.

```java
import java.util.*;

public class ZigzagSort {
    public static void zigzagSort(int[] arr) {
        boolean flag = true;
        for (int i = 0; i < arr.length - 1; i++) {
            if (flag && arr[i] > arr[i + 1]) swap(arr, i, i + 1);
            else if (!flag && arr[i] < arr[i + 1]) swap(arr, i, i + 1);
            flag = !flag;
        }
    }

    private static void swap(int[] arr, int i, int j) {
        int temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
    }

    public static void main(String[] args) {
        int[] arr = {4, 3, 7, 8, 6, 2, 1};
        zigzagSort(arr);
        System.out.println(Arrays.toString(arr));
    }
}
```

OUTPUT:

```
[3, 7, 4, 8, 2, 6, 1]
```