**PROGRAM 1:**

**Segregate 0’s and 1’s**

Input -> arr[] = { 0, 1, 0, 1, 1, 0, 1 };

Output -> Array after segregation is 0 0 0 1 1 1

**DEFINE A FUNCTIONAL INTERFACE:**

**package** javaexample.segregate;

@FunctionalInterface

**public** **interface** SegregateZeroAndOnesService {

**public** **int**[] segregateZeroAndOnes(**int**[] arr);

}

**INVOKE SERVICE CLASS METHOD:**

**package** javaexample.segregate;

**public** **class** SegregateZeroAndOnes {

SegregateZeroAndOnesService service;

**public** SegregateZeroAndOnes() {

}

**public** SegregateZeroAndOnes(SegregateZeroAndOnesService service) {

**this**.service = service;

}

**public** **int**[] perform(**int**[] arr) {

**return** service.segregateZeroAndOnes(arr);

}

}

**MAIN CLASS TO TEST THE APPLICATION:**

**package** javaexample.segregate;

**import** java.util.Arrays;

**public** **class** TesterSegregateZeroAndOnes {

**public** **static** **void** main(String[] args) {

**int**[] arr = { 0, 1, 0, 1, 1, 0, 1 };

SegregateZeroAndOnesService service = **new** SegregateZeroAndOnesService() {

@Override

**public** **int**[] segregateZeroAndOnes(**int**[] arr) {

**int** zeroCount = 0;

**int** n = arr.length;

**for** (**int** i = 0; i < n; i++) {

**if** (arr[i] == 0) {

arr[zeroCount++] = 0;

}

}

**for** (**int** i = zeroCount; i < n; i++) {

arr[i] = 1;

}

**return** arr;

}

};

SegregateZeroAndOnes obj = **new** SegregateZeroAndOnes(service);

**int**[] result = obj.perform(arr);

Arrays.*stream*(result).forEach(System.***out***::print);

}

}

**JUNIT USING MOCKITO:**

**package** javaexample.segregate;

**import** **static** org.junit.Assert.assertEquals;

**import** **static** org.mockito.Mockito.verify;

**import** **static** org.mockito.Mockito.when;

**import** org.junit.Before;

**import** org.junit.Rule;

**import** org.junit.Test;

**import** org.mockito.Mock;

**import** org.mockito.junit.MockitoJUnit;

**import** org.mockito.junit.MockitoRule;

**public** **class** TestSegregateZeroAndOnes {

SegregateZeroAndOnes segregateZeroAndOnes = **null**;

@Rule

**public** MockitoRule rule = MockitoJUnit.rule();

@Mock

SegregateZeroAndOnesService service;

@Before

**public** **void** setUp() {

segregateZeroAndOnes = **new** SegregateZeroAndOnes(service);

}

@Test

**public** **void** testPerform1() {

**int**[] input = { 0, 1, 0, 1, 1, 0, 1 };

**int**[] expected = { 0, 0, 0, 1, 1, 1, 1 };

when(service.segregateZeroAndOnes(input)).thenReturn(expected);

assertEquals(expected, segregateZeroAndOnes.perform(input));

verify(service).segregateZeroAndOnes(input);

}

@Test

**public** **void** testPerform2() {

**int**[] input = { 0, 1, 1, 1, 0, 1, 0, 0, 1, 1 };

**int**[] expected = { 0, 0, 0, 0, 1, 1, 1, 1, 1, 1 };

when(service.segregateZeroAndOnes(input)).thenReturn(expected);

assertEquals(expected, segregateZeroAndOnes.perform(input));

verify(service).segregateZeroAndOnes(input);

}

}

**PROGRAM 2:**

**Find Max Zero’s Between Ones**

Input -> String a = "1000110000100"

Output -> 4

**DEFINE A FUNCTIONAL INTERFACE:**

**package** javaexample.max;

@FunctionalInterface

**public** **interface** MaxZeroesBetweenOnesService {

**public** **int** maxConsecutiveZeroes(String str);

}

**INVOKE SERVICE CLASS METHOD:**

**package** javaexample.max;

**public** **class** MaxZeroesBetweenOnes {

MaxZeroesBetweenOnesService service;

**public** MaxZeroesBetweenOnes() {

}

**public** MaxZeroesBetweenOnes(MaxZeroesBetweenOnesService service) {

**this**.service = service;

}

**public** **int** perform(String str) {

**return** service.maxConsecutiveZeroes(str);

}

}

**MAIN CLASS TO TEST THE APPLICATION:**

**package** javaexample.max;

**public** **class** TesterMaxZeroesBetweenOnes {

**public** **static** **void** main(String[] args) {

String str = "1000110000100";

MaxZeroesBetweenOnesService service = **new** MaxZeroesBetweenOnesService() {

@Override

**public** **int** maxConsecutiveZeroes(String str) {

**int** n = str.length();

String[] strArr = str.split("");

**int**[] arr = **new** **int**[n];

**for** (**int** i = 0; i < n; i++) {

arr[i] = Integer.*parseInt*(strArr[i]);

}

**int** count = 0;

**int** maxConsecutiveZeroes = 0;

**for** (**int** i = 0; i < n; i++) {

**if** (arr[i] == 0) {

count++;

**if** (maxConsecutiveZeroes < count) {

maxConsecutiveZeroes = count;

}

} **else** {

count = 0;

}

}

**return** maxConsecutiveZeroes;

}

};

MaxZeroesBetweenOnes obj = **new** MaxZeroesBetweenOnes(service);

**int** result = obj.perform(str);

System.***out***.println(result);

}

}

**JUNIT USING MOCKITO:**

**package** javaexample.max;

**import** **static** org.junit.Assert.assertEquals;

**import** **static** org.mockito.Mockito.verify;

**import** **static** org.mockito.Mockito.when;

**import** org.junit.Before;

**import** org.junit.Rule;

**import** org.junit.Test;

**import** org.mockito.Mock;

**import** org.mockito.junit.MockitoJUnit;

**import** org.mockito.junit.MockitoRule;

**public** **class** TestMaxZeroesBetweenOnes {

MaxZeroesBetweenOnes maxZeroesBetweenOnes = **null**;

@Rule

**public** MockitoRule rule = MockitoJUnit.rule();

@Mock

MaxZeroesBetweenOnesService service;

@Before

**public** **void** setUp() {

maxZeroesBetweenOnes = **new** MaxZeroesBetweenOnes(service);

}

@Test

**public** **void** testPerform1() {

String str = "1000110000100";

when(service.maxConsecutiveZeroes(str)).thenReturn(4);

assertEquals(4, maxZeroesBetweenOnes.perform(str));

verify(service).maxConsecutiveZeroes(str);

}

@Test

**public** **void** testPerform2() {

String str = "10000100010010000010000100001";

when(service.maxConsecutiveZeroes(str)).thenReturn(5);

assertEquals(5, maxZeroesBetweenOnes.perform(str));

verify(service).maxConsecutiveZeroes(str);

}

}

**PROGRAM 3:**

**Given an array arr[] and a number K where K is smaller than size of array, the task is to find the Kth smallest element in the given array. It is given that all array elements are distinct.**

Input -> arr[] = {7,10, 4, 3, 20, 15};  K=3

Output -> 7

**DEFINE A FUNCTIONAL INTERFACE:**

**package** javaexample.kth;

@FunctionalInterface

**public** **interface** KthSmallestElementInArrayService {

**public** **int** getKthSmallestElementUsingHeap(**int**[] arr, **int** k);

}

**INVOKE SERVICE CLASS METHOD:**

**package** javaexample.kth;

**public** **class** KthSmallestElementInArray {

KthSmallestElementInArrayService service;

**public** KthSmallestElementInArray() {

}

**public** KthSmallestElementInArray(KthSmallestElementInArrayService service) {

**this**.service = service;

}

**public** **int** perform(**int**[] arr, **int** k) {

**return** service.getKthSmallestElementUsingHeap(arr, k);

}

}

**MAIN CLASS TO TEST THE APPLICATION:**

**package** javaexample.kth;

**import** java.util.Collections;

**import** java.util.PriorityQueue;

**public** **class** TesterKthSmallestElementInArray {

**public** **static** **void** main(String[] args) {

**int**[] arr = { 7, 10, 4, 3, 20, 15 };

**int** k = 3;

KthSmallestElementInArrayService service = **new** KthSmallestElementInArrayService() {

@Override

**public** **int** getKthSmallestElementUsingHeap(**int**[] arr, **int** k) {

PriorityQueue<Integer> heap = **new** PriorityQueue<Integer>(Collections.*reverseOrder*());

**for** (**int** i : arr) {

heap.add(i);

**if** (heap.size() > k) {

heap.remove();

}

}

**return** heap.remove();

}

};

KthSmallestElementInArray obj = **new** KthSmallestElementInArray(service);

**int** result = obj.perform(arr, k);

System.***out***.println(result);

}

}

**JUNIT USING MOCKITO:**

**package** javaexample.kth;

**import** **static** org.junit.Assert.assertEquals;

**import** **static** org.mockito.Mockito.verify;

**import** **static** org.mockito.Mockito.when;

**import** org.junit.Before;

**import** org.junit.Rule;

**import** org.junit.Test;

**import** org.mockito.Mock;

**import** org.mockito.junit.MockitoJUnit;

**import** org.mockito.junit.MockitoRule;

**public** **class** TestKthSmallestElementInArray {

KthSmallestElementInArray kthSmallestElementInArray = **null**;

// 1 way

@Rule

**public** MockitoRule rule = MockitoJUnit.rule();

@Mock

KthSmallestElementInArrayService service;

@Before

**public** **void** setUp() {

kthSmallestElementInArray = **new** KthSmallestElementInArray(service);

}

@Test

**public** **void** testPerform1() {

**int**[] arr = { 7, 10, 4, 3, 20, 15 };

**int** k = 3;

when(service.getKthSmallestElementUsingHeap(arr, k)).thenReturn(7);

assertEquals(7, kthSmallestElementInArray.perform(arr, k));

verify(service).getKthSmallestElementUsingHeap(arr, k);

}

@Test

**public** **void** testPerform2() {

**int**[] arr = { 9, 8, 2, 6, 4, 19, 11, 5 };

**int** k = 7;

when(service.getKthSmallestElementUsingHeap(arr, k)).thenReturn(11);

assertEquals(11, kthSmallestElementInArray.perform(arr, k));

verify(service).getKthSmallestElementUsingHeap(arr, k);

}

}