1.  Create a class illustrating all the three types of constructors

● No arguments constructor

● Default constructor

● Parameterised constructor (can create more than one with different type of parameters)

public class MyClass {

// Data members or fields

private int num;

private String name;

// No-arguments constructor (explicitly defined for clarity)

public MyClass() {

System.out.println("No-arguments constructor called.");

num = 0;

name = "Default";

}

// Parameterized constructor 1 (with int parameter)

public MyClass(int num) {

System.out.println("Parameterized constructor (int) called.");

this.num = num;

name = "Default";

}

// Parameterized constructor 2 (with String parameter)

public MyClass(String name) {

System.out.println("Parameterized constructor (String) called.");

num = 0;

this.name = name;

}

// Parameterized constructor 3 (with both int and String parameters)

public MyClass(int num, String name) {

System.out.println("Parameterized constructor (int, String) called.");

this.num = num;

this.name = name;

}

// Method to display the values of the data members

public void display() {

System.out.println("Number: " + num);

System.out.println("Name: " + name);

}

// Main method to test the constructors

public static void main(String[] args) {

// Create objects using different constructors

MyClass obj1 = new MyClass(); // No-arguments constructor

MyClass obj2 = new MyClass(10); // Parameterized constructor (int)

MyClass obj3 = new MyClass("Alice"); // Parameterized constructor (String)

MyClass obj4 = new MyClass(20, "Bob"); // Parameterized constructor (int, String)

// Call the display method to show the initialized values

obj1.display();

obj2.display();

obj3.display();

obj4.display();

}

}

2.  Given a sorted integer array (in increasing order), remove duplicates in-place such that each unique element appears only once. The relative order of the elements should be kept the same. Then return the number of unique elements in the array.

**Input**

[22,22,77,77,88, 89,89]

**Output**

4

**Explanation :**After removing duplicates -> [22, 77, 88, 89, \_, \_, \_ ]

No. of unique elements = 4

public class RemoveDuplicates {

public static int removeDuplicates(int[] nums) {

if (nums.length == 0) {

return 0;

}

int i = 0;

for (int j = 1; j < nums.length; j++) {

if (nums[i] != nums[j]) {

i++;

nums[i] = nums[j];

}

}

return i + 1;

}

public static void main(String[] args) {

int[] nums = {22, 22, 77, 77, 88, 89, 89};

int uniqueElements = removeDuplicates(nums);

System.out.println("Number of unique elements: " + uniqueElements);

System.out.println("Modified array: " + Arrays.toString(nums));

}

}

3 .  An array contains both positive and negative numbers in random order. Rearrange the array elements so that all negative numbers appear before all positive numbers. Don’t use .sort() method

**Input**[-12, 11, -13, -5, 6, -7, 5, -3, -6]

**Output**[-12, -13, -5, -7, -3, -6, 11, 6, 5]

public class RearrangeArray {

public static void rearrange(int[] nums) {

int i = 0, j = nums.length - 1;

while (i < j) {

while (i < j && nums[i] < 0) { // Move i forward until it points to a positive number

i++;

}

while (i < j && nums[j] >= 0) { // Move j backward until it points to a negative number

j--;

}

if (i < j) { // Swap nums[i] and nums[j] if they are not in their correct positions

int temp = nums[i];

nums[i] = nums[j];

nums[j] = temp;

}

}

}

public static void main(String[] args) {

int[] nums = {-12, 11, -13, -5, 6, -7, 5, -3, -6};

rearrange(nums);

System.out.println("Rearranged array: " + Arrays.toString(nums));

}

}