**CSE1007 –JAVA PROGRAMMING**

Programs on Looping, Array and String, Class Fundamentals

**ASSESSMENT 3**

**Multithreading, JavaFX**

1. **Given an array of int, write a Java program, which uses two threads one to print in ascending order and another to print in descending order. Write a driver class to test the functionality.**

**Approach:**

1. Import the required libraries: The program imports the java.util.Scanner and java.util.Arrays libraries.
2. Define the sorting class with the main method: This class is the entry point of the program.
3. Input array size and elements:

* Create a Scanner object sc to read user input.
* Prompt the user to enter the size of the array (n).
* Create an integer array arr of size n.
* Use a for loop to read n integers from the user and store them in the arr array.

1. Create sorting objects and threads:

* Create an instance of the Sorting1 class called sort and initialize it with the user's input array (arr).
* Create two thread objects, ascending and descending, representing the ascending and descending sorting processes, respectively.

1. Clone the array for descending sorting:

* Create a copy of the original array arr using Arrays.copyOf. This copy will be used by the descending thread to sort the array in descending order.

1. Start sorting threads:

* Start both the ascending and descending threads using the start method.

1. Wait for threads to complete:

* Use join to make the main thread wait until both the ascending and descending threads finish their sorting tasks.

1. Print sorted arrays:

* After both threads have completed, print the sorted array in ascending order using the print method of the sort object.
* Print the sorted array in descending order using the print method of the sortCopy object (the copied array).

1. Sorting1 class:

* Represents an object that encapsulates the array to be sorted.
* Contains synchronized methods ascending and descending to perform bubble sort in ascending and descending order, respectively.
* Both methods include a sleep of 1000 milliseconds to simulate the sorting process slowly.

1. print method:

* The print method prints the contents of the array.

1. AscendingThread and DescendingThread classes:

* These are thread classes that extend the Thread class.
* They override the run method to call the ascending and descending methods of the Sorting1 object, respectively.

|  |
| --- |
| **Program Code:**  package ASSIGNMENT3;  import java.util.Scanner;  import java.util.Arrays;  public class sorting {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  System.out.println("Enter the size of the array: ");  int n = sc.nextInt();  int[] arr = new int[n];  for (int i = 0; i < n; i++) {  arr[i] = sc.nextInt();  }  Sorting1 sort = new Sorting1(arr);  AscendingThread ascending = new AscendingThread(sort);  // Create a copy of the original array for the descending thread  int[] arrCopy = Arrays.copyOf(arr, arr.length);  Sorting1 sortCopy = new Sorting1(arrCopy);  DescendingThread descending = new DescendingThread(sortCopy);  ascending.start();  descending.start();  try {  ascending.join();  descending.join();  } catch (InterruptedException e) {  System.out.println("Interrupted");  }  // Print the sorted array after both threads have finished  sort.print();  sortCopy.print();  sc.close();  }  }  class Sorting1 {  private final int[] arr;  public Sorting1(int[] array) {  this.arr = array;  }  public synchronized void ascending() {  for (int i = 0; i < arr.length - 1; i++) {  boolean swapped = false;  for (int j = 0; j < arr.length - i - 1; j++) {  if (arr[j] > arr[j + 1]) {  swapped = true;  int temp = arr[j];  arr[j] = arr[j + 1];  arr[j + 1] = temp;  }  }  try {  Thread.sleep(1000);  } catch (InterruptedException e) {  System.out.println("Interrupted");  }  if (!swapped) {  break;  }  }  }  public synchronized void descending() {  for (int i = 0; i < arr.length - 1; i++) {  boolean swapped = false;  for (int j = 0; j < arr.length - i - 1; j++) {  if (arr[j] < arr[j + 1]) {  swapped = true;  int temp = arr[j];  arr[j] = arr[j + 1];  arr[j + 1] = temp;  }  }  try {  Thread.sleep(1000);  } catch (InterruptedException e) {  System.out.println("Interrupted");  }  if (!swapped) {  break;  }  }  System.out.println();  }  public void print() {  for (int i = 0; i < arr.length; i++) {  System.out.print(arr[i] + " ");  }  System.out.println();  }  }  class AscendingThread extends Thread {  private final Sorting1 sort;  public AscendingThread(Sorting1 sorting) {  this.sort = sorting;  }  public void run() {  sort.ascending();  }  }  class DescendingThread extends Thread {  private final Sorting1 sort;  public DescendingThread(Sorting1 sort) {  this.sort = sort;  }  public void run() {  sort.descending();  }  } |

**Code Screenshot:**

A screen shot of a computer screen

Description automatically generated

A screen shot of a computer screen

Description automatically generated

**OUTPUT:**

1. **Design a thread that generates odd numbers below 1000 and writes them into an array. Design another thread that access the array generated by thread1 and find the prime numbers within that array. Write a driver class to test the functionality.**

**Approach:**

1. Define the OddAndPrime class with the main method.
2. Initialize instances of the OddNumbers and PrimeNumbers threads.
3. Start the OddNumbers thread using the start method.
4. Use odd.join() to make the main thread wait for the OddNumbers thread to finish.
5. Print a message and the odd numbers from 1 to 100.
6. Start the PrimeNumbers thread using the start method.
7. Use prime.join() to make the main thread wait for the PrimeNumbers thread to finish.
8. Print a message and the prime numbers identified within the generated odd numbers.
9. The OddNumbers class generates odd numbers and stores them in an ArrayList.
10. The PrimeNumbers class identifies prime numbers among the odd numbers using the isprime method.
11. The isprime method checks whether a number is prime using a prime-checking algorithm.
12. The program execution ends after the PrimeNumbers thread completes its execution.

|  |
| --- |
| **Program code:**  package ASSIGNMENT3;  import java.util.ArrayList;  public class OddAndPrime {  public static void main(String[] args) {  OddNumbers odd = new OddNumbers();  PrimeNumbers prime = new PrimeNumbers(odd.getOddNumbers());  // Start the OddNumbers thread first  System.out.println("the Odd Numbers from 1 to 100 is ");  odd.start();  try {  odd.join(); // Wait for OddNumbers to finish  } catch (InterruptedException e) {  e.printStackTrace();  }  System.out.println("the prime numbers within the odd numbers are :");  // Start the PrimeNumbers thread after OddNumbers has finished  prime.start();  try {  prime.join(); // Wait for PrimeNumbers to finish  } catch (InterruptedException e) {  e.printStackTrace();  }  }  }  class OddNumbers extends Thread {  ArrayList<Integer> oddnumbers = new ArrayList<>();  public void run() {  for (int i = 1; i <= 100; i += 2) {  System.out.print(i + " ");  oddnumbers.add(i);  }  System.out.println();  }  public ArrayList<Integer> getOddNumbers() {  return oddnumbers;  }  }  class PrimeNumbers extends Thread {  private ArrayList<Integer> oddnumbers;  private ArrayList<Integer> primenumbers = new ArrayList<>();  public PrimeNumbers(ArrayList<Integer> odd) {  this.oddnumbers = odd;  }  public void run() {  for (int num : oddnumbers) {  if (isprime(num)) {  System.out.print(num + " ");  primenumbers.add(num);  }  }  }  public boolean isprime(int num) {  if (num <= 1) {  return false;  }  if (num == 2) {  return true;  }  if (num % 2 == 0) {  return false;  }  for (int i = 3; i \* i <= num; i += 2) {  if (num % i == 0) {  return false;  }  }  return true;  }  } |

**Code Screenshot:**

A screen shot of a computer

Description automatically generated



**Output:**

**A black background with blue and white text

Description automatically generated**