***Assignment 8*** *:* ***Thread and multithread***

**Q1. In this challenge, simulate a banking system. Create the Account and Transaction classes.**

**1. The Account class has a data member int balance, initially assigned to zero. The class should implement the following three methods:**

**String deposit(int money) to add money to the balance. This method should return a string that describes the deposit transaction, i.e., "Depositing $money".**

**String withdraw(int money) to subtract money from the balance. This method should return a string that describes the withdraw transaction, i.e., "Withdrawing $money". Note that, if there is insufficient balance to successfully withdraw the desired amount, then the balance should not be adjusted, and the returned string should be "Withdrawing $money (Insufficient Balance)".**

**int getBalance() to return the account balance.**

**2. The Transaction class has two data members Account account and List transactions. The class should implement the following three methods:**

**Ans:**

**import java.security.SecureRandom;**

**import java.util.List;**

**import java.util.Scanner;**

**import java.util.ArrayList;**

**class Account {**

**int balance=0;**

**public String deposit(int money){**

**balance += money;**

**return "Depositing"+money;**

**}**

**public String withdraw(int money){**

**if(balance<money){**

**return "Withdraw"+money+" (Insufficient Balance)";**

**} else{**

**balance -= money;**

**return "Withdraw"+money;**

**}**

**}**

**public int getBalance(){**

**return balance;**

**}**

**}**

**class Transaction {**

**Account account = new Account();**

**List<String> transactions = new ArrayList<>();**

**public Transaction(Account account){**

**this.account = account;**

**}**

**public void deposit(int money){**

**transactions.add(account.deposit(money));**

**}**

**public void withdraw(int money){**

**transactions.add(account.withdraw(money));**

**}**

**public List<String> getTransaction(){**

**return transactions;**

**}**

**}**

**class TransactionRunnable implements Runnable {**

**private static final SecureRandom RANDOM\_GENERATOR = new SecureRandom();**

**private final Transaction transaction;**

**private final int transactionsCount;**

**public TransactionRunnable(Transaction transaction, int transactionsCount) {**

**this.transaction = transaction;**

**this.transactionsCount = transactionsCount;**

**}**

**public void run() {**

**for (int i = 0; i < this.transactionsCount; i++) {**

**int transactionType = RANDOM\_GENERATOR.nextInt() % 2;**

**int money = RANDOM\_GENERATOR.nextInt(100) + 1;**

**if (transactionType == 0) {**

**this.transaction.deposit(money);**

**} else {**

**this.transaction.withdraw(money);**

**}**

**}**

**}**

**}**

**public class a8q1 {**

**private static final Scanner SCANNER = new Scanner(System.in);**

**private static final Account ACCOUNT = new Account();**

**private static final Transaction TRANSACTION = new Transaction(ACCOUNT);**

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

**int threadsCount = Integer.parseInt(SCANNER.nextLine());**

**Thread[] threads = new Thread[threadsCount];**

**int expectedTransactionsCount = 0;**

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

**int transactionsCount = Integer.parseInt(SCANNER.nextLine());**

**expectedTransactionsCount += transactionsCount;**

**threads[i] = new Thread(new TransactionRunnable(TRANSACTION, transactionsCount));**

**}**

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

**threads[i].start();**

**}**

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

**threads[i].join();**

**}**

**List<String> transactions = TRANSACTION.getTransaction();**

**if (transactions.size() != expectedTransactionsCount) {**

**System.out.println("Wrong Answer");**

**} else {**

**boolean correct = true;**

**for (String transaction: transactions) {**

**if (transaction == null) {**

**correct = false;**

**break;**

**}**

**}**

**if (!correct) {**

**System.out.println("Wrong Answer");**

**} else {**

**int balance = ACCOUNT.getBalance();**

**if (balance < 0) {**

**System.out.println("Wrong Answer");**

**} else {**

**for (String transaction: transactions) {**

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

**}**

**System.out.println("Balance " + balance);**

**}**

**}**

**}**

**}**

**}**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**2 Write a program of producer and consumer**

**Ans:**

**public class ProducerConsumerTest {**

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

**CubbyHole c = new CubbyHole();**

**Producer p1 = new Producer(c, 1);**

**Consumer c1 = new Consumer(c, 1);**

**p1.start();**

**c1.start();**

**}**

**}**

**class CubbyHole {**

**private int contents;**

**private boolean available = false;**

**public synchronized int get() {**

**while (available == false) {**

**try {**

**wait();**

**} catch (InterruptedException e) {}**

**}**

**available = false;**

**notifyAll();**

**return contents;**

**}**

**public synchronized void put(int value) {**

**while (available == true) {**

**try {**

**wait();**

**} catch (InterruptedException e) { }**

**}**

**contents = value;**

**available = true;**

**notifyAll();**

**}**

**}**

**class Consumer extends Thread {**

**private CubbyHole cubbyhole;**

**private int number;**

**public Consumer(CubbyHole c, int number) {**

**cubbyhole = c;**

**this.number = number;**

**}**

**public void run() {**

**int value = 0;**

**for (int i = 0; i < 10; i++) {**

**value = cubbyhole.get();**

**System.out.println("Consumer #" + this.number + " got: " + value);**

**}**

**}**

**}**

**class Producer extends Thread {**

**private CubbyHole cubbyhole;**

**private int number;**

**public Producer(CubbyHole c, int number) {**

**cubbyhole = c;**

**this.number = number;**

**}**

**public void run() {**

**for (int i = 0; i < 10; i++) {**

**cubbyhole.put(i);**

**System.out.println("Producer #" + this.number + " put: " + i);**

**try {**

**sleep((int)(Math.random() \* 100));**

**} catch (InterruptedException e) { }**

**}**

**}**

**}**

**Output:**

Producer #1 put: 0

Consumer #1 got: 0

Producer #1 put: 1

Consumer #1 got: 1

Producer #1 put: 2

Consumer #1 got: 2

Producer #1 put: 3

Consumer #1 got: 3

Producer #1 put: 4

Consumer #1 got: 4

Producer #1 put: 5

Consumer #1 got: 5

Producer #1 put: 6

Consumer #1 got: 6

Producer #1 put: 7

Consumer #1 got: 7

Producer #1 put: 8

Consumer #1 got: 8

Producer #1 put: 9

Consumer #1 got: 9

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

3: **Write a program of thread to used all the method of thread like (wait, sleep, notify, notifyall,join,getname, currentthreade islive?, priority ).**

**you have to write on sigle program to used this all.**

**Ans:**

**class A extends Thread**

**{**

**public void run()**

**{**

**for (int i = 1; i <= 4; i++) {**

**try {**

**wait();**

**}**

**catch (Exception e) {**

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

**}**

**System.out.print(i + " ");**

**}**

**}**

**}**

**class B extends Thread {**

**public void run()**

**{ System.out.println(Thread.currentThread().getName() + " in control");**

**for (char i = 'a'; i <= 'd'; i++) {**

**try {**

**Thread.sleep(100);**

**}**

**catch (Exception e) {**

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

**}**

**System.out.print(i + " ");**

**}**

**}**

**}**

**class a8q3 extends Thread {**

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

**{**

**// creating two threads**

**A a1 = new A();**

**B b1 = new B();**

**// starts second thread after when**

**// first thread a1 is died.**

**a1.start();**

**a1.setPriority(5);**

**System.out.println(a1.isAlive());**

**b1.start();**

**try {**

**a1.join();**

**}**

**catch (Exception e) {**

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

**}**

**}**

**}**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**4 You are required to compute the power of a number by implementing a calculator. Create a class MyCalculator which consists of a single method long power(int, int). This method takes two integers, n and p, as parameters and finds . If either p or n is negative, then the method must throw an exception which says "n and p should not be negative". Also, if both and are zero, then the method must throw an exception which says "n and p should not be zero"**

**For example, -4 and -5 would result in java.lang.Exception: n or p should not be negative.**

**Complete the function power in class MyCalculator and return the appropriate result after the power operation or an appropriate exception as detailed above.**

**Ans:**

**import java.util.Scanner;**

**class MyCalculator {**

**public static int power(int n,int p) throws Exception {**

**if(n<0 || p<0)**

**{**

**throw new Exception ("n or p should not be negative.");**

**}**

**else if(n==0 && p==0)**

**{**

**throw new Exception ("n and p should not be zero.");**

**}**

**else{return((int)Math.pow(n,p));**

**}**

**}**

**}**

**public class Solution**

**{**

**public static final MyCalculator my\_calculator = new MyCalculator();**

**public static final Scanner in = new Scanner(System.in);**

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

**{**

**while (in .hasNextInt())**

**{**

**int n = in .nextInt();**

**int p = in .nextInt();**

**try**

**{**

**System.out.println(my\_calculator.power(n, p));**

**}**

**catch (Exception e) {**

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

**}**

**}**

**}**

**}**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Q5.** **Java Program Showing Execution of Multiple Tasks with a Single Thread**

**Ans:**

**class Count extends Thread**

**{**

**Count()**

**{**

**super("my extending thread");**

**System.out.println("my thread created" + this);**

**start();**

**}**

**public void run()**

**{**

**try**

**{**

**for (int i=0 ;i<10;i++)**

**{**

**System.out.println("Printing the count " + i);**

**Thread.sleep(1000);**

**}**

**}**

**catch(InterruptedException e)**

**{**

**System.out.println("my thread interrupted");**

**}**

**System.out.println("My thread run is over" );**

**}**

**}**

**class ExtendingExample**

**{**

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

**{**

**Count cnt = new Count();**

**try**

**{**

**while(cnt.isAlive())**

**{**

**System.out.println("Main thread will be alive till the child thread is live");**

**Thread.sleep(1500);**

**}**

**}**

**catch(InterruptedException e)**

**{**

**System.out.println("Main thread interrupted");**

**}**

**System.out.println("Main thread's run is over”);**

**Output:**

**my thread createdThread[my runnable thread,5,main]**

**Main thread will be alive till the child thread is live**

**Printing the count 0**

**Printing the count 1**

**Main thread will be alive till the child thread is live**

**Printing the count 2**

**Main thread will be alive till the child thread is live**

**Printing the count 3**

**Printing the count 4**

**Main thread will be alive till the child thread is live**

**Printing the count 5**

**Main thread will be alive till the child thread is live**

**Printing the count 6**

**Printing the count 7**

**Main thread will be alive till the child thread is live**

**Printing the count 8**

**Main thread will be alive till the child thread is live**

**Printing the count 9**

**mythread run is over**

**Main thread run is over**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Q6.Java Program Showing Two Threads Working Simultaneously Upon Two Objects**

**Ans:**

**class ThreadDemo extends Thread {**

**private Thread t;**

**private String threadName;**

**ThreadDemo( String name) {**

**threadName = name;**

**System.out.println("Creating " + threadName );**

**}**

**public void run() {**

**System.out.println("Running " + threadName );**

**try {**

**for(int i = 4; i > 0; i--) {**

**System.out.println("Thread: " + threadName + ", " + i);**

**// Let the thread sleep for a while.**

**Thread.sleep(50);**

**}**

**}**

**catch (InterruptedException e)**

**{**

**System.out.println("Thread " + threadName + " interrupted.");**

**}**

**System.out.println("Thread " + threadName + " exiting.");**

**}**

**public void start () {**

**System.out.println("Starting " + threadName );**

**if (t == null) {**

**t = new Thread (this, threadName);**

**t.start ();**

**}**

**}**

**}**

**public class TestThread {**

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

**ThreadDemo T1 = new ThreadDemo( "Thread-1");**

**T1.start();**

**ThreadDemo T2 = new ThreadDemo( "Thread-2");**

**T2.start();**

**}**

**}**

**Output::**

**Creating Thread-1**

**Starting Thread-1**

**Creating Thread-2**

**Starting Thread-2**

**Running Thread-1**

**Thread: Thread-1, 4**

**Running Thread-2**

**Thread: Thread-2, 4**

**Thread: Thread-1, 3**

**Thread: Thread-2, 3**

**Thread: Thread-1, 2**

**Thread: Thread-2, 2**

**Thread: Thread-1, 1**

**Thread: Thread-2, 1**

**Thread Thread-1 exiting.**

**Thread Thread-2 exiting.**

**Q7.Java Program Showing Two Threads Acting Upon a Single Object**

**Ans:**

**public class PrintTest {**

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

**PrintNumbers b = new PrintNumbers();**

**One firstThread = new One(b);**

**Two secondThread = new Two(b);**

**Three thirdThread = new Three(b);**

**secondThread.setName("second: ");**

**thirdThread.setName("third: ");**

**firstThread.setName("first: ");**

**firstThread.start();**

**secondThread.start();**

**thirdThread.start();**

**}**

**}**

**o/p:**

**first: 1**

**second: 2**

**third: 3**

**first: 11**

**second: 22**

**third: 33**

**first: 111**

**second: 222**

**third: 333**

**...and so on**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Q8.Java Program with 2 Threads Which Prints Alternatively**

**Ans:**

**class One extends Thread**

**{**

**public void run()**

**{**

**for(int i=0;i<10;i++)**

**{**

**System.out.println("1");}**

**}**

**}**

**class Two extends Thread**

**{**

**public void run()**

**{**

**for(int j=0;j<10;j++)**

**{**

**System.out.println("2");**

**}**

**}**

**}**

**class Test**

**{**

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

**{**

**One thr1=new One();**

**Two thr2=new Two();**

**thr1.start();**

**thr2.start();**

**}**

**}**

**Q9.Java Program to Start One Thread More than Once.**

**Ans:**

**public class TestThreadTwice1 extends Thread{**

**public void run()**

**{**

**System.out.println("running...");**

**}**

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

**{**

**TestThreadTwice1 t1=new TestThreadTwice1();**

**t1.start();**

**t1.start();**

**}**

**}**

**Output :**

**running**

**Exception in thread "main"**

**java.lang.IllegalThreadStateException**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Q10.Java Program to Check CurrentThread in Multi Threading**

**Ans:**

**public class Test extends Thread**

**{**

**public static void main(String[] args) { // getting reference to Main thread**

**Thread t = Thread.currentThread(); // getting name of Main thread System.out.println("Current**

**thread: " + t.getName());**

**// changing the name of Main thread**

**t.setName("Geeks");**

**System.out.println("After name change: " + t.getName());**

**// getting priority of Main thread**

**System.out.println("Main thread priority: "+ t.getPriority());**

**// setting priority of Main thread to MAX(10)**

**t.setPriority(MAX\_PRIORITY);**

**System.out.println("Main thread new priority: "+ t.getPriority());**

**for (int i = 0; i < 2; i++){**

**System.out.println("Main thread");**

**}**

**ChildThread ct = new ChildThread();**

**// getting priority of child thread**

**// which will be inherited from Main thread**

**// as it is created by Main thread**

**System.out.println("Child thread priority: "+ ct.getPriority());**

**// setting priority of Main thread to MIN(1)**

**ct.setPriority(MIN\_PRIORITY);**

**System.out.println("Child thread new priority: "+ ct.getPriority());**

**// starting child thread**

**ct.start();**

**}**

**}**

**// Child Thread class**

**class ChildThread extends Thread {**

**public void run()**

**{**

**for (int i = 0; i < 2; i++)**

**{**

**System.out.println("Child thread");**

**}**

**}**

**}**

**Output:**

**Current thread: main**

**After name change: Geeks**

**Main thread priority: 5**

**Main thread new priority: 10**

**Main thread**

**Main thread**

**Child thread priority: 10**

**Child thread new priority: 1**

**Child thread**

**Child thread**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Q11.Java Program to Create a Server with 2 Threads to Communicate with Serveral Clients**

**import java.io.\*;**

**import java.text.\*;**

**import java.util.\*;**

**import java.net.\*;**

**// Server class**

**public class Server**

**{**

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

**throwsIOException**

**{**

**// server is listening on port 5056**

**ServerSocket ss = new ServerSocket(5056);**

**// running infinite loop for getting client request**

**while (true)**

**{**

**Socket s = null;**

**Try**

**{**

**// socket object to receive incoming client requests**

**s = ss.accept();**

**System.out.println("A new client is connected : " + s);**

**DataInputStream dis = new DataInputStream(s.getInputStream()); DataOutputStream dos = new**

**DataOutputStream(s.getOutputStream());**

**System.out.println("Assigning new thread for this client"); // create a new thread object**

**Thread t = new ClientHandler(s, dis, dos);**

**// Invoking the start() method**

**t.start();**

**}**

**catch (Exception e)**

**{**

**s.close();**

**e.printStackTrace();**

**}**

**}**

**}**

**} // ClientHandler class**

**class ClientHandler extends Thread**

**{**

**DateFormat fordate = new SimpleDateFormat("yyyy/MM/dd");**

**DateFormat fortime = new SimpleDateFormat("hh:mm:ss");**

**final DataInputStream dis;**

**final DataOutputStream dos;**

**final Socket s;**

**// Constructor**

**public ClientHandler(Socket s, DataInputStream dis, DataOutputStream dos)**

**{**

**this.s = s;**

**this.dis = dis;**

**this.dos = dos;**

**}**

**@Overrid**

**public void run()**

**{**

**String received;**

**String toreturn;**

**while (true)**

**{**

**try**

**{**

**// Ask user what he wants**

**dos.writeUTF("What do you want?[Date | Time]..\n"+ "Type Exit to terminate connection.");**

**// receive the answer from client**

**received = dis.readUTF();**

**if(received.equals("Exit"))**

**{**

**System.out.println("Client " + this.s + " sends exit...");**

**System.out.println("Closing this connection.");**

**this.s.close();**

**System.out.println("Connection closed");**

**break;**

**}**

**// creating Date object**

**Date date = new Date();**

**// write on output stream based on t**

**// answer from the client**

**switch (received)**

**{**

**case "Date" :**

**toreturn = fordate.format(date); dos.writeUTF(toreturn);**

**break;**

**case "Time" :**

**toreturn = fortime.format(date); dos.writeUTF(toreturn);**

**break;**

**default:**

**dos.writeUTF("Invalid input");**

**break;**

**}**

**}**

**catch (IOException e)**

**{**

**e.printStackTrace();**

**}**

**}**

**try**

**{ // closing resources**

**this.dis.close();**

**this.dos.close();**

**}**

**catch(IOException e)**

**{**

**e.printStackTrace();**

**}**

**}**

**}**

**Output:**

**A new client is connected : Socket[addr=/127.0.0.1,port=60536,localport=5056]**

**Assigning new thread for this client**

**Client Socket[addr=/127.0.0.1,port=60536,localport=5056] sends exit...**

**Closing this connection.**

**Connection closed**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Q12.Java Program to Create a Client That Receive Message From the Server**

**Ans:**

**import java.io.\*;**

**import java.net.\*;**

**import java.util.Scanner;**

**public class Client {**

**final static int ServerPort = 1234;**

**public static void main(String args[]) throws UnknownHostException, IOException {**

**Scanner scn = new Scanner(System.in);**

**ip = InetAddress.getByName("localhost");**

**Socket s = new Socket(ip, ServerPort);**

**// obtaining input and out streams**

**DataInputStream dis = new DataInputStream(s.getInputStream()); DataOutputStream dos = new**

**DataOutputStream(s.getOutputStream());**

**// sendMessage thread**

**Thread sendMessage = new Thread(new Runnable()**

**{**

**@Override**

**public void run()**

**{**

**while (true)**

**{**

**String msg = scn.nextLine();**

**Try**

**{ // write on the output stream**

**dos.writeUTF(msg);**

**}**

**catch (IOException e)**

**{**

**e.printStackTrace();**

**}**

**}**

**}**

**});**

**// readMessage thread**

**Thread readMessage = new Thread(new Runnable()**

**{**

**@Override**

**public void run()**

**{**

**while (true)**

**{**

**try {**

**// read the message sent to this client**

**String msg = dis.readUTF();**

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

**}**

**catch (IOException e)**

**{**

**e.printStackTrace();**

**}**

**}**

**}**

**});**

**sendMessage.start();**

**readMessage.start();**

**}**

**}**

**Output :**

**From client 0:**

**Hello # client1**

**Client 1:hii**

**How are you #client 1**

**Client 1:fine…how are you?**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Q13.Java Program of a Multithreaded Implementation of Any Parallelized Divide-Conquer Algorithm**

**Ans :**

**import java.io.IOException;**

**import java.util.Arrays;**

**import java.util.Random;**

**import java.util.Scanner;**

**public class a8q13{**

**public static int[] inputArray;**

**public static int[] arr1;**

**public static int[] arr2;**

**public static int[] arr3;**

**public static int t1\_status=0;**

**public static int t2\_status=0;**

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

**Scanner in =new Scanner(System.in);**

**int arraySize=5;**

**inputArray = new int[arraySize];**

**Random rand=new Random();**

**for(int i=0;i<arraySize;i++)**

**{**

**inputArray[i]=rand.nextInt(100);**

**}**

**arr1=Arrays.copyOfRange(inputArray, 0, inputArray. Length/2);**

**arr2=Arrays.copyOfRange(inputArray, (inputArray.length)/2,inputArray.length);**

**System.out.print("The original array is array is ");**

**for(int h:inputArray)**

**{**

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

**}**

**Thread t1=new Thread(new Runnable(){**

**public void run()**

**{**

**mergeSort(arr1);**

**System.out.println("t1 started");**

**}**

**});**

**Thread t2=new Thread(new Runnable(){**

**public void run()**

**{**

**mergeSort(arr2);**

**System.out.println("t2 started");**

**}**

**});**

**t1.start();**

**t2.start();**

**try {**

**t1.join();**

**t2.join();**

**}**

**catch (InterruptedException e) {**

**e.printStackTrace();**

**}**

**if(t1.isAlive())**

**{**

**t1\_status=1;**

**}**

**if(t2.isAlive())**

**{**

**t2\_status=1;**

**}**

**t1.stop();**

**t2.stop();**

**arr3=new int[inputArray.length];**

**merge(arr3,arr1,arr2);**

**System.out.println("The sorted array using divide and conquer algorithm = ");**

**for(int m:arr3)**

**{**

**System.out.print(m);**

**System.out.print(" ");**

**}**

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

**}**

**static void mergeSort(int[] A)**

**{**

**if (A.length > 1)**

**{**

**int q = A.length/2;**

**// divide the array in half**

**int[] leftArray = Arrays.copyOfRange(A, 0, q);**

**int[] rightArray = Arrays.copyOfRange(A,q,A.length);**

**// sort/conquer each half**

**mergeSort(leftArray);**

**mergeSort(rightArray);**

**merge(A,leftArray,rightArray);**

**}**

**}**

**static void merge(int[] a, int[] l, int[] r) {**

**int totElem = l.length + r.length;**

**int i,li,ri;**

**i = li = ri = 0;**

**while ( i < totElem) {**

**if ((li < l.length) && (ri<r.length)) {**

**if (l[li] < r[ri]) {**

**a[i] = l[li];**

**i++;**

**li++;**

**}**

**else {**

**a[i] = r[ri];**

**i++;**

**ri++;**

**}**

**}l**

**else {**

**if (li >= l.length)**

**{**

**while (ri < r.length)**

**{**

**a[i] = r[ri];**

**i++;**

**ri++;**

**}**

**}**

**if (ri >= r.length)**

**{**

**while (li < l.length)**

**{**

**a[i] = l[li];**

**li++;**

**i++;**

**}**

**}**

**}**

**}**

**if(t1\_status==1){arr1=a;}**

**else if(t2\_status==1){arr2=a;}**

**else{arr3=a;}**

**}**

**}**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**