Assignment 2: Multithreading
Set A
a) Program to define a thread for printing text on output screen for 'n' number of times.
Create 3 threads and run them. Pass the text 'n' parameters to the thread constructor.
Example:
i. First thread prints "COVID19" 10 times.
ii. Second thread prints "LOCKDOWN2020" 20 times
iii. Third thread prints "VACCINATED2021" 30 times

```
import java.io.*;
import java.util.*;
public class tdemo4 extends Thread
{
int n;
String msg;
tdemo4(int n,String msg)
{
this.n=n;
this.msg=msg;
}
public void run()
{
try
{
for(int i=1;i<=n;i++)
{
System.out.println(msg);

}
}
catch(Exception e)
{
System.out.println(e);
}
}
public static void main(String args[])
{
```

```
Thread t1=new Thread(new tdemo4(10,"COVID19"));
t1.start();
Thread t2=new Thread(new tdemo4(20,"LOCKDOWN2020"));
t2.start();
Thread t3=new Thread(new tdemo4(30,"VACCINATED2021"));
t3.start();
}
}
```

b) Write a program in which threads sleep for 6 sec in the loop in reverse order from 100 to 1 and change the name of the thread.

```
public class ReverseCountdownThread extends Thread {

  public ReverseCountdownThread(String name) {
    super(name); // Call the superclass constructor to set the initial name
  }

  @Override
  public void run() {
    try {
      for (int i = 100; i >= 1; i--) {
        // Change the thread name in each iteration
        setName("CountdownThread-" + i);
        System.out.println(Thread.currentThread().getName() + ": " + i);
        Thread.sleep(6000); // Sleep for 6 seconds (6000 milliseconds)
      }
    } catch (InterruptedException e) {
      System.out.println(Thread.currentThread().getName() + " was interrupted.");
      Thread.currentThread().interrupt(); // Restore the interrupted status
    }
  }

  public static void main(String[] args) {
  // Create an instance of the custom thread with an initial name
  ReverseCountdownThread myThread = new
ReverseCountdownThread("InitialCountdownThread");
    myThread.start(); // Start the thread
  }
```

}

c) Write a program to solve producer consumer problem in which a producer produces a value and consumer consume the value before producer generate the next value.Hint: use thread synchronization)

```
class SharedBuffer {
    private int data;
    private boolean hasData = false; // indicates whether data is available

    // Producer puts data into buffer
    public synchronized void produce(int value) {
        while (hasData) {  // wait if data has not been consumed
            try {
                wait();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

        data = value;
        System.out.println("Produced: " + data);
        hasData = true;
        notify(); // notify consumer
    }

    // Consumer takes data from buffer
    public synchronized int consume() {
        while (!hasData) {  // wait if no data produced
            try {
                wait();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

        System.out.println("Consumed: " + data);
        hasData = false;
        notify(); // notify producer
        return data;
    }
```

```
}

// Producer thread
class Producer extends Thread {
    private SharedBuffer buffer;

    public Producer(SharedBuffer buffer) {
        this.buffer = buffer;
    }

    public void run() {
        for (int i = 1; i <= 10; i++) { // produce 10 items
            buffer.produce(i);
            try {
                Thread.sleep(500); // simulate production time
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}

// Consumer thread
class Consumer extends Thread {
    private SharedBuffer buffer;

    public Consumer(SharedBuffer buffer) {
        this.buffer = buffer;
    }

    public void run() {
        for (int i = 1; i <= 10; i++) { // consume 10 items
            buffer.consume();
            try {
                Thread.sleep(500); // simulate consumption time
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
```

```
}

// Main class
public class ProducerConsumerExample {
    public static void main(String[] args) {
        SharedBuffer buffer = new SharedBuffer();

        Producer producer = new Producer(buffer);
        Consumer consumer = new Consumer(buffer);

        producer.start();
        consumer.start();
    }
}
```

Set B

a) Write a program to calculate the sum and average of an array of 1000 integers (generated randomly) using 10 threads. Each thread calculates the sum of 100 integers. Use these values to calculate average. [Use join method].

```
import java.util.Random;

class SumThread extends Thread {
    private int[] arr;
    private int start, end;
    private long partialSum = 0;

    public SumThread(int[] arr, int start, int end) {
        this.arr = arr;
        this.start = start;
        this.end = end;
    }

    public void run() {
        for (int i = start; i < end; i++) {
            partialSum += arr[i];
        }
```

```
    }

    public long getPartialSum() {
        return partialSum;
    }
}

public class SumAverageUsingThreads {
    public static void main(String[] args) {
        int[] arr = new int[1000];
        Random rand = new Random();

        // Generate 1000 random integers
        for (int i = 0; i < arr.length; i++) {
            arr[i] = rand.nextInt(100);  // values 0–99
        }

        SumThread[] threads = new SumThread[10];

        // Create threads (each handles 100 integers)
        for (int i = 0; i < 10; i++) {
            threads[i] = new SumThread(arr, i * 100, (i + 1) * 100);
            threads[i].start();
        }

        // Wait for all threads using join
        long totalSum = 0;
        try {
            for (int i = 0; i < 10; i++) {
                threads[i].join();
                totalSum += threads[i].getPartialSum();
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        double average = totalSum / 1000.0;

        System.out.println("Total Sum = " + totalSum);
        System.out.println("Average = " + average);
```

```
    }
}
```

b) Write a program for a simple search engine. Accept a string to be searched. Search for the string in all text files in the current folder. Use a separate thread for each file. The result should display the filename, line number where the string is found.

```
import java.io.*;
import java.util.*;

class SearchThread extends Thread {
    private File file;
    private String searchString;

    public SearchThread(File file, String searchString) {
        this.file = file;
        this.searchString = searchString;
    }

    public void run() {
        try (BufferedReader br = new BufferedReader(new FileReader(file))) {
            String line;
            int lineNum = 1;

            while ((line = br.readLine()) != null) {
                if (line.contains(searchString)) {
                    System.out.println("Found in " + file.getName() +
                                       " at line " + lineNum);
                }
                lineNum++;
            }
        } catch (Exception e) {
            System.out.println("Error reading file: " + file.getName());
        }
    }
}
```

```
public class SimpleSearchEngine {
    public static void main(String[] args) throws IOException {

        // Read input search term
        BufferedReader input = new BufferedReader(new
InputStreamReader(System.in));
        System.out.print("Enter string to search: ");
        String searchString = input.readLine();

        // Get all .txt files in current folder
        File directory = new File(".");
        File[] files = directory.listFiles((dir, name) -> name.endsWith(".txt"));

        if (files == null || files.length == 0) {
            System.out.println("No text files found.");
            return;
        }

        // Create and start a thread for each file
        List<SearchThread> threads = new ArrayList<>();
        for (File file : files) {
            SearchThread t = new SearchThread(file, searchString);
            threads.add(t);
            t.start();
        }

        // Wait for all threads to finish
        for (SearchThread t : threads) {
            try {
                t.join();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

        System.out.println("Search completed.");
    }
}
```

c) Write a program that implements a multi-thread application that has three threads.
First thread generates random integer every 1 second and if the value is even, second thread computes the square of the number and prints. If the value is odd, the third thread will print the value of cube of the number.

```
import java.io.*;
import java.util.*;
class square extends Thread
{
int n;
square(int n)
{
this.n=n;
}
public void run()
{
System.out.println("Square:"+n*n);
}
}
class cube extends Thread
{
int n;
cube(int n)
{
this.n=n;
}
public void run()
{

System.out.println("Cube:"+n*n*n);
}
}
class number extends Thread
{
int n;
public void run()
{
try
```

```
{
Random r=new Random();
for(int i=1;i<=10;i++)
{
n=r.nextInt(20);
System.out.println("generated number is:"+n);
if(n%2==0)
{
square s=new square(n);
s.start();
}
else
{
cube c=new cube(n);
c.start();
}
sleep(2000);
}
}
catch(Exception e)
{
System.out.println(e);
}
}
}
class tdemo7
{
public static void main(String args[])
{
number n=new number();
n.start();
}
}
```

Set C

a) Write a program that simulates a traffic light. The program lets the user select one of three lights: red, yellow, or green with radio buttons. On selecting a button, an appropriate message with “stop” or “ready” or “go”should appear above the buttons in a selected color. Initially there is no message shown.

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class TrafficLightSimulation extends JFrame implements ActionListener
{

    JRadioButton red, yellow, green;
    JLabel message;

    public TrafficLightSimulation() {

        setTitle("Traffic Light Simulation");
        setSize(400, 200);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new FlowLayout());

        // Label for displaying message
        message = new JLabel("");
        message.setFont(new Font("Arial", Font.BOLD, 24));
        add(message);

        // Radio buttons
        red = new JRadioButton("Red");
        yellow = new JRadioButton("Yellow");
        green = new JRadioButton("Green");

        // Group the radio buttons
        ButtonGroup group = new ButtonGroup();
        group.add(red);
        group.add(yellow);
        group.add(green);

        // Add listeners
        red.addActionListener(this);
```

```
        yellow.addActionListener(this);
        green.addActionListener(this);

        // Add buttons to frame
        add(red);
        add(yellow);
        add(green);

        setVisible(true);
    }

    public void actionPerformed(ActionEvent e) {

        if (red.isSelected()) {
            message.setText("STOP");
            message.setForeground(Color.RED);
        }
        else if (yellow.isSelected()) {
            message.setText("READY");
            message.setForeground(Color.ORANGE);
        }
        else if (green.isSelected()) {
            message.setText("GO");
            message.setForeground(Color.GREEN);
        }
    }

    public static void main(String[] args) {
        new TrafficLightSimulation();
    }
}
```

b) Write a program to create a thread for moving a ball inside a panel vertically. The ball should be created when the user clicks on the start button.

```
import javax.swing.*;
import java.awt.*;
```

```
import java.awt.event.*;

public class BallMovement extends JFrame implements ActionListener {

    private JButton startButton;
    private BallPanel panel;

    public BallMovement() {
        setTitle("Vertical Ball Movement");
        setSize(400, 400);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        panel = new BallPanel();
        startButton = new JButton("Start");

        startButton.addActionListener(this);

        add(panel, BorderLayout.CENTER);
        add(startButton, BorderLayout.SOUTH);

        setVisible(true);
    }

    public void actionPerformed(ActionEvent e) {
        panel.startBallMovement();
    }

    public static void main(String[] args) {
        new BallMovement();
    }
}

// Panel where ball will move
class BallPanel extends JPanel {

    private int y = -20;    // Starting position above panel
    private int x = 180;    // Horizontal center
    private int dy = 5;     // Movement speed
    private boolean running = false;
```

```
    public void startBallMovement() {
        if (!running) {
            running = true;
            Thread t = new Thread(() -> moveBall());
            t.start();
        }
    }

    private void moveBall() {
        while (true) {
            y += dy;

            // Reverse direction on hitting top or bottom
            if (y <= 0 || y >= getHeight() - 20)
                dy = -dy;

            repaint();

            try {
                Thread.sleep(30);  // controls speed (30 ms)
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }

    public void paintComponent(Graphics g) {
        super.paintComponent(g);

        g.setColor(Color.BLUE);
        g.fillOval(x, y, 20, 20);  // draw ball (20x20)
    }
}
```

c) Using the concepts of thread synchronization create two threads as sender and receiver. Sender thread will set a message to the receiver thread that will display the message on console. The sender thread accepts the input message from console. Continue this process until sender sets the message as “Good Bye Corona”.

```
import java.util.Scanner;

class MessageBox {
    private String message;
    private boolean hasMessage = false;

    // sender sets message
    public synchronized void sendMessage(String msg) {
        while (hasMessage) {          // wait if previous message not consumed
            try { wait(); } catch (InterruptedException e) {}
        }
        message = msg;
        hasMessage = true;
        notify();                     // notify receiver
    }

    // receiver reads message
    public synchronized String receiveMessage() {
        while (!hasMessage) {         // wait if no message
            try { wait(); } catch (InterruptedException e) {}
        }
        hasMessage = false;
        notify();                     // notify sender for next message
        return message;
    }
}

class Sender extends Thread {
    private MessageBox box;
    private Scanner sc = new Scanner(System.in);

    public Sender(MessageBox box) {
        this.box = box;
    }

    public void run() {
        String msg = "";

        while (true) {
            System.out.print("Enter message: ");
```

```
            msg = sc.nextLine();

            box.sendMessage(msg);

            if (msg.equals("Good Bye Corona"))
                break;
        }
    }
}

class Receiver extends Thread {
    private MessageBox box;

    public Receiver(MessageBox box) {
        this.box = box;
    }

    public void run() {
        while (true) {
            String msg = box.receiveMessage();
            System.out.println("Receiver got: " + msg);

            if (msg.equals("Good Bye Corona"))
                break;
        }
    }
}

public class SenderReceiverSync {
    public static void main(String[] args) {
        MessageBox box = new MessageBox();

        Sender sender = new Sender(box);
        Receiver receiver = new Receiver(box);

        sender.start();
        receiver.start();
    }
}
```

******** write a java program to add elements in fixed size array ********

```
import java.util.Arrays;

public class FixedSizeArrayAddition {

	public static void main(String[] args) {
	// Original fixed-size array
	int[] originalArray = {10, 20, 30, 40};
	System.out.println("Original Array: " + Arrays.toString(originalArray));
	// Element to add
	int newElement = 50;

	// Create a new array with one additional space
	int[] newArray = new int[originalArray.length + 1];

	// Copy elements from the original array to the new array
	for (int i = 0; i < originalArray.length; i++) {
	newArray[i] = originalArray[i];
	}

	// Add the new element to the last position of the new array
	newArray[newArray.length - 1] = newElement;

	System.out.println("Array after adding element: " +
Arrays.toString(newArray));
	}
}
```