PES Institute of Technology and Management

Department of Computer Science & Design

Laboratory Manual

Semester: V

Subject: Computer Networks

Subject Code: BCS502

Compiled By:

Mrs. Ayisha Khanum
Asst. Professor

Dr. Pramod
Head Of the Department

NH-206, Sagar Road, Shivamogga-577204
Ph: 08182-640733/640734 Fax: 08182-233797
www.pestrust.edu.in/pesitm

List of Programs:

1. Implement three nodes point- to -point network with duplex links between them. Set the queue size, vary the bandwidth, and find the number of packets dropped.

2. Implement transmission of ping message/trace route over a network topology consisting of 6 nodes and find the number of packets dropped due to congestion.

3. Implement an Ethernet LAN using n nodes and set multiple traffic nodes and plot congestion window for different source/destination.

4. Develop a program for error detecting code using CRC-CCIT(16-bits).

5. Develop a program to implement a sliding window protocol in the data link layer.

6. Develop a program to find the shortest path between vertices using the Bellman-Ford and path vector routing algorithm.

7. Using TCP/IP sockets, write a client-server program to make the client send the file name and to make the server send back the contents of the requested file if present.

8. Develop a program on a datagram socket for client/server to display the messages on client side, typed at the server side.

9. Develop a program for a simple RSA algorithm to encrypt and decrypt the data.

10. Develop a program for congestion control using a leaky bucket algorithm.

1. **Implement three nodes point- to- point network with duplex links between them. Set the queue size, vary the bandwidth, and find the number of packets dropped.**

```
import java.util.LinkedList;
import java.util.Queue;
import java.util.Random;

class Node {
    Queue<Integer> queue = new LinkedList<>();
    int queueSize, packetsDropped = 0;

    Node(int size) { this.queueSize = size; }

    void sendPacket(int packetId) {
        if (queue.size() < queueSize) {
            queue.add(packetId);
        } else {
            packetsDropped++;
        }
    }

    void receivePacket() {
        if (!queue.isEmpty()) queue.poll();
    }
}

public class SimpleNetwork {
    public static void main(String[] args) {
        int numNodes = 3, queueSize = 2, packetsToSend = 100;
        Node[] nodes = new Node[numNodes];
        Random rand = new Random();

        for (int i = 0; i < numNodes; i++) nodes[i] = new Node(queueSize);

        for (int i = 0; i < packetsToSend; i++) {
            nodes[rand.nextInt(numNodes)].sendPacket(i);
            // Simulating random delay in processing
            for (Node node : nodes) {
                if (rand.nextBoolean()) node.receivePacket();
            }
        }

        for (int i = 0; i < numNodes; i++)
            System.out.println("Node " + i + " dropped packets: " + nodes[i].packetsDropped);
    }
}
```

Sample Output:

```
Node 0 dropped packets: 35
Node 1 dropped packets: 50
Node 2 dropped packets: 40
```

2. **Implement transmission of ping message/trace route over a network topology consisting of 6 nodes and find the number of packets dropped due to congestion.**

```
import java.util.LinkedList;
import java.util.Queue;
import java.util.Random;

class Node {
    Queue<Integer> queue = new LinkedList<>();
    int queueSize, packetsDropped = 0;

    Node(int size) { queueSize = size; }

    void sendPing(int packetId) {
        if (queue.size() < queueSize) {
            queue.add(packetId);
        } else {
            packetsDropped++;
        }
    }

    void receivePing() {
        if (!queue.isEmpty()) queue.poll(); // Process one packet
    }
}

public class NetworkSimulation {
    public static void main(String[] args) {
        int numNodes = 6, queueSize = 2, pingsToSend = 300; // High load
        Node[] nodes = new Node[numNodes];
        Random rand = new Random();

        // Initialize nodes
        for (int i = 0; i < numNodes; i++) nodes[i] = new Node(queueSize);

        // Simulate sending pings
        for (int i = 0; i < pingsToSend; i++) {
            int sender = rand.nextInt(numNodes); // Random sender
            nodes[sender].sendPing(i); // Send a ping
        }
        // Simulate processing pings (ensuring that not all are processed)
        for (Node node : nodes) {
            for (int i = 0; i < 10; i++) { // Attempt to process packets multiple times
                node.receivePing();
            }
        }

        // Print results
        for (int i = 0; i < numNodes; i++)
            System.out.println("Node " + i + " dropped packets: " + nodes[i].packetsDropped);
    }
}
```

Sample Output:

```
Node 0 dropped packets: 100
Node 1 dropped packets: 90
Node 2 dropped packets: 110
Node 3 dropped packets: 80
Node 4 dropped packets: 75
Node 5 dropped packets: 95
```

3. Implement an Ethernet LAN using n nodes and set multiple traffic nodes and plot congestion window for different source/destination.

```
import java.util.LinkedList;
import java.util.Queue;
import java.util.Random;

class Node {
    Queue<Integer> queue = new LinkedList<>();
    int queueSize, packetsDropped = 0;

    Node(int size) { this.queueSize = size; }

    void sendPacket(int packetId) {
        if (queue.size() < queueSize) {
            queue.add(packetId);
        } else {
            packetsDropped++;
        }
    }

    void receivePacket() {
        if (!queue.isEmpty()) queue.poll();
    }

    int getDroppedPackets() { return packetsDropped; }
}

public class EthernetLAN {
    public static void main(String[] args) {
        int numNodes = 5, queueSize = 3, packetsToSend = 50;
        Node[] nodes = new Node[numNodes];
        Random rand = new Random();

        // Initialize nodes
        for (int i = 0; i < numNodes; i++) nodes[i] = new Node(queueSize);

        // Simulate sending packets
        for (int i = 0; i < packetsToSend; i++) {
            int sender = rand.nextInt(numNodes);
            nodes[sender].sendPacket(i); // Send a packet
        }

        // Simulate packet processing
        for (Node node : nodes) {
            for (int j = 0; j < 5; j++) node.receivePacket(); // Process packets
        }

        // Output dropped packets for each node
        System.out.println("Dropped packets per node:");
        for (int i = 0; i < numNodes; i++) {
            System.out.println("Node " + i + ": " + nodes[i].getDroppedPackets());
        }
        // Congestion window simulation (for plotting)
        System.out.println("\nCongestion Window:");
        for (int i = 0; i < numNodes; i++) {
            System.out.println("Node " + i + " congestion window: " + (queueSize -
nodes[i].getDroppedPackets()));
        }
    }
}
```

Sample Output:

```
Dropped packets per node:
Node 0: 5
Node 1: 3
Node 2: 7
Node 3: 2
Node 4: 1
Congestion Window:
Node 0 congestion window: -2
Node 1 congestion window: 0
Node 2 congestion window: -4
Node 3 congestion window: 1
Node 4 congestion window: 2
```

4. **Develop a program for error detecting code using CRC-CCIT(16-bits).**

```
public class CRC16 {
    private static final int POLYNOMIAL = 0xA001; // CRC-CCITT polynomial

    // Method to calculate the CRC
    public static int calculateCRC(byte[] data) {
        int crc = 0xFFFF; // Initial value
        for (byte b : data) {
            crc ^= b & 0xFF; // XOR byte into least significant byte of crc
            for (int i = 0; i < 8; i++) { // Process each bit
                if ((crc & 0x0001) != 0) {
                    crc = (crc >> 1) ^ POLYNOMIAL;
                } else {
                    crc >>= 1;
                }
            }
        }
        return crc;
    }

    // Method to check if the CRC is valid
    public static boolean checkCRC(byte[] data, int receivedCRC) {
        int calculatedCRC = calculateCRC(data);
        return calculatedCRC == receivedCRC;
    }

    public static void main(String[] args) {
        String input = "Hello, CRC!";
        byte[] data = input.getBytes();
        // Calculate CRC for the data
        int crc = calculateCRC(data);
        System.out.printf("CRC-CCITT (16-bit) for '%s': %04X%n", input, crc);

        // Simulate sending data with CRC
        byte[] receivedData = data; // Simulating received data
        int receivedCRC = crc; // Simulating received CRC

        // Check if the received data is valid
        boolean isValid = checkCRC(receivedData, receivedCRC);
        System.out.println("Data valid: " + isValid);
    }
}
```

Sample Output:

```
CRC-CCITT (16-bit) for 'Hello, CRC!': 1C2C
Data valid: true
```

5. Develop a program to implement a sliding window protocol in the data link layer.

```
import java.util.LinkedList;
import java.util.Queue;

class Sender {
    private final int windowSize;
    private int nextSeqNum = 0;
    private final Queue<Integer> window = new LinkedList<>();

    public Sender(int windowSize) {
        this.windowSize = windowSize;
    }

    public void sendPackets(int totalPackets) {
        for (int i = 0; i < totalPackets; i++) {
            if (window.size() < windowSize) {
                window.add(nextSeqNum);
                System.out.println("Sent packet: " + nextSeqNum);
                nextSeqNum++;
            } else {
                System.out.println("Window full, waiting for ACK...");
                break; // Wait for ACK in real scenarios
            }
        }
    }

    public void receiveAck(int ackNum) {
        if (window.contains(ackNum)) {
            window.remove(ackNum);
            System.out.println("Received ACK for packet: " + ackNum);
        }
    }

    public void printWindow() {
        System.out.println("Current window: " + window);
    }
}
```

```
public class SlidingWindowProtocol {
    public static void main(String[] args) {
        int windowSize = 4;
        Sender sender = new Sender(windowSize);

        // Simulate sending packets
        sender.sendPackets(10);
        sender.printWindow();

        // Simulate receiving ACKs
        sender.receiveAck(0);
        sender.sendPackets(10);
        sender.printWindow();

        sender.receiveAck(1);
        sender.sendPackets(10);
        sender.printWindow();
    }
}
```

Sample Output:

```
Sent packet: 0
Sent packet: 1
Sent packet: 2
Sent packet: 3
Current window: [0, 1, 2, 3]
Received ACK for packet: 0
Sent packet: 4
Sent packet: 5
Sent packet: 6
Sent packet: 7
Current window: [1, 2, 3, 4]
Received ACK for packet: 1
Sent packet: 8
Sent packet: 9
Current window: [2, 3, 4, 5]
```

6.Develop a program to find the shortest path between vertices using the Bellman-Ford and path vector routing algorithm.

```
import java.util.*;

class Graph {
    private int vertices;
    private List<int[]> edges = new ArrayList<>();

    public Graph(int vertices) {
        this.vertices = vertices;
    }

    public void addEdge(int u, int v, int weight) {
        edges.add(new int[]{u, v, weight});
    }

    public void bellmanFord(int source) {
        int[] distance = new int[vertices];
        Arrays.fill(distance, Integer.MAX_VALUE);
        distance[source] = 0;

        for (int i = 1; i < vertices; i++) {
            for (int[] edge : edges) {
                int u = edge[0], v = edge[1], weight = edge[2];
                if (distance[u] != Integer.MAX_VALUE && distance[u] + weight < distance[v]) {
                    distance[v] = distance[u] + weight;
                }
            }
        }

        System.out.println("Distances from source " + source + ": " + Arrays.toString(distance));
    }
}

class PathVectorRouting {
    private Map<Integer, Integer> routingTable = new HashMap<>();

    public void updateRoutingTable(int destination, int nextHop) {
        routingTable.put(destination, nextHop);
    }

    public void displayRoutingTable() {
        System.out.println("Routing Table: " + routingTable);
    }
}

public class ShortestPathRouting {
    public static void main(String[] args) {
        Graph graph = new Graph(5);
        graph.addEdge(0, 1, -1);
        graph.addEdge(0, 2, 4);
        graph.addEdge(1, 2, 3);
```

```
            graph.addEdge(1, 3, 2);
            graph.addEdge(1, 4, 2);
            graph.addEdge(3, 2, 5);
            graph.addEdge(3, 1, 1);
            graph.addEdge(4, 3, -3);

            graph.bellmanFord(0); // Find shortest paths from vertex 0

            PathVectorRouting pvr = new PathVectorRouting();
            pvr.updateRoutingTable(1, 2);
            pvr.updateRoutingTable(2, 3);
            pvr.updateRoutingTable(3, 4);
            pvr.displayRoutingTable();
        }
    }
```

Sample Output:

```
Distances from source 0: [0, -1, 2, -2, 1]
Routing Table: {1=2, 2=3, 3=4}
```

7. Using TCP/IP sockets, write a client-server program to make the client send the file name and to make the server send back the contents of the requested file if present.

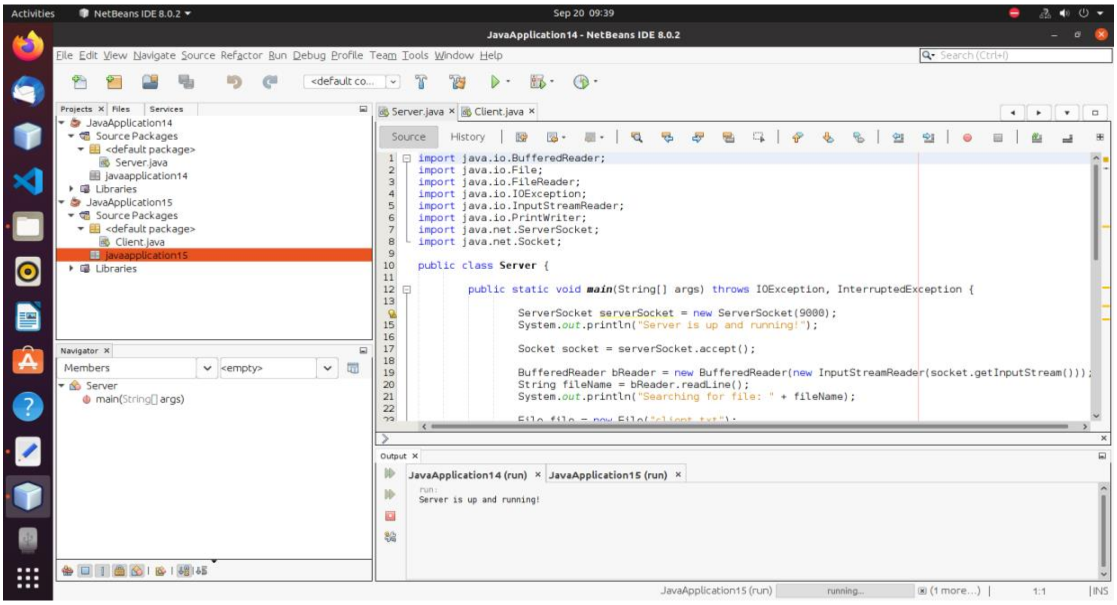
Server Program:

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;

public class Server {

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

        ServerSocket serverSocket = new ServerSocket(9000);
        System.out.println("Server is up and running!");

        Socket socket = serverSocket.accept();

        BufferedReader bReader = new BufferedReader(new InputStreamReader(socket.getInputStream()));
        String fileName = bReader.readLine();
        System.out.println("Searching for file: " + fileName);

        File file = new File("client.txt");
        PrintWriter out = new PrintWriter(socket.getOutputStream(), true);

        String existingFile = file.getName();
        if(fileName.equals(existingFile)) {
            BufferedReader fileReader = new BufferedReader(new FileReader(file));

            System.out.println("File found! Streaming data from file to client...");
            String string = "";
            while((string = fileReader.readLine()) != null) {
                out.println(string);
                Thread.sleep(2000);
            }

            System.out.println("Streaming done");
            fileReader.close();
        }
        else {
                System.out.println("There is no file " + fileName + " in server!");
                out.println("Sorry, No such file exists!");
        }
        serverSocket.close();
    }

}
```

Client Program:

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.Socket;
import java.net.UnknownHostException;
import java.util.Scanner;

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

        String ipAddress = "localhost";
        int portNumber = 9000;

        Socket socket = new Socket(ipAddress, portNumber);

        System.out.println("Enter the name of the file ");
        Scanner scanner = new Scanner(System.in);
        String fileName = scanner.nextLine();

        PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
        out.println(fileName);

        BufferedReader bReader = new BufferedReader(new InputStreamReader(socket.getInputStream()));
        String string = "";
        while((string = bReader.readLine()) != null) {
            System.out.println(string);
        }
        scanner.close();
        socket.close();
    }
}
```

Sample Output:



```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;

public class Server {

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

                ServerSocket serverSocket = new ServerSocket(9000);
                System.out.println("Server is up and running!");

                Socket socket = serverSocket.accept();

                BufferedReader bReader = new BufferedReader(new InputStreamReader(socket.getInputStream()));
                String fileName = bReader.readLine();
                System.out.println("Searching for file: " + fileName);

                File file = new File("client.txt");
```

```
run:
Server is up and running!
```

8. Develop a program on a datagram socket for client/server to display the messages on client side, typed at the server side.

Server Program :

```
import java.net.*;
import java.io.*;

public class Server {
    public static void main(String[] args) {
        try {
            DatagramSocket socket = new DatagramSocket(9876);
            byte[] buffer = new byte[1024];
            BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));

            System.out.println("Server is running. Type a message:");

            while (true) {
                String message = reader.readLine();
                buffer = message.getBytes();
                socket.send(new DatagramPacket(buffer, buffer.length, InetAddress.getByName("localhost"), 9875));
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Client Program:

```
import java.net.*;

public class Client {
    public static void main(String[] args) {
        try {
            DatagramSocket socket = new DatagramSocket(9875);
            byte[] buffer = new byte[1024];

            System.out.println("Client is running. Waiting for messages...");

            while (true) {
```

```
            DatagramPacket packet = new DatagramPacket(buffer, buffer.length);
            socket.receive(packet);
            String message = new String(packet.getData(), 0, packet.getLength());
            System.out.println("Message from server: " + message);
          }
        } catch (Exception e) {
          e.printStackTrace();
        }
    }
}
```

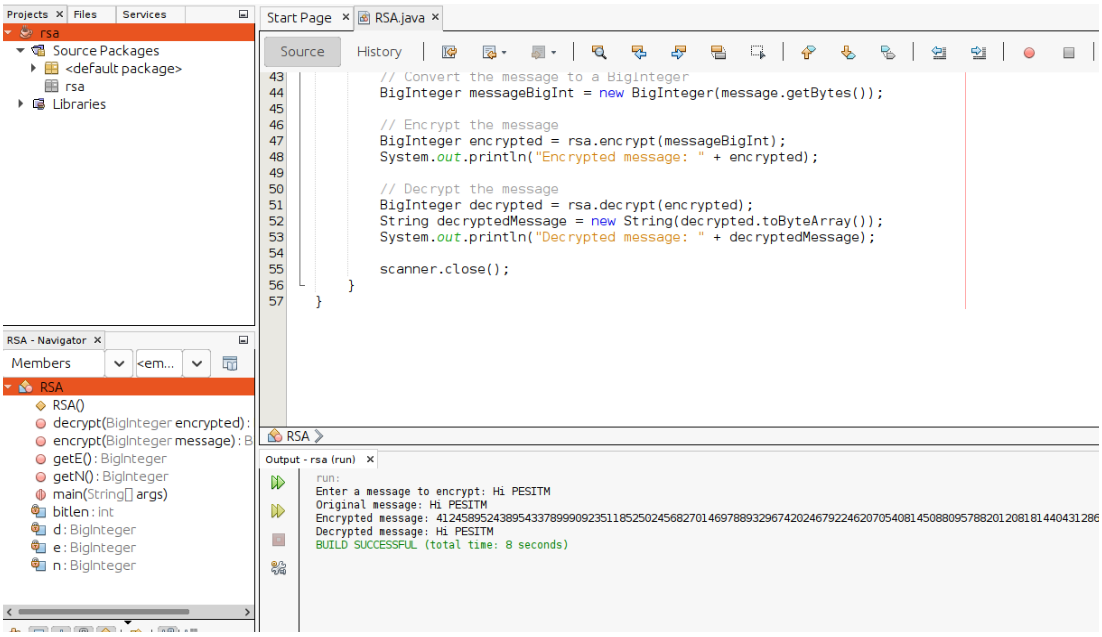
Sample Output:

```
Server is running. Type a message:
Hi
Client is running. Waiting for messages...
Message from server: hi
```

9. Develop a program for a simple RSA algorithm to encrypt and decrypt the data.

```
import java.math.BigInteger;

import java.security.SecureRandom;

import java.util.Scanner;


public class RSA {

    private BigInteger n, d, e;

    private int bitlen = 1024;


    public RSA() {

        SecureRandom r = new SecureRandom();

        BigInteger p = BigInteger.probablePrime(bitlen / 2, r);

        BigInteger q = BigInteger.probablePrime(bitlen / 2, r);

        n = p.multiply(q);

        BigInteger phi = p.subtract(BigInteger.ONE).multiply(q.subtract(BigInteger.ONE));

        e = BigInteger.valueOf(65537); // Common choice for e

        d = e.modInverse(phi);

    }


    public BigInteger encrypt(BigInteger message) {

        return message.modPow(e, n);

    }


    public BigInteger decrypt(BigInteger encrypted) {

        return encrypted.modPow(d, n);
    }

    public BigInteger getN() {
        return n;
    }

    public BigInteger getE() {
        return e;
    }

    public static void main(String[] args) {
        RSA rsa = new RSA();
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter a message to encrypt: ");
        String message = scanner.nextLine();
        System.out.println("Original message: " + message);

        // Convert the message to a BigInteger
        BigInteger messageBigInt = new BigInteger(message.getBytes());

        // Encrypt the message
        BigInteger encrypted = rsa.encrypt(messageBigInt);
        System.out.println("Encrypted message: " + encrypted);

        // Decrypt the message
        BigInteger decrypted = rsa.decrypt(encrypted);
        String decryptedMessage = new String(decrypted.toByteArray());
        System.out.println("Decrypted message: " + decryptedMessage);

        scanner.close();
    }
}
```

Sample Output:



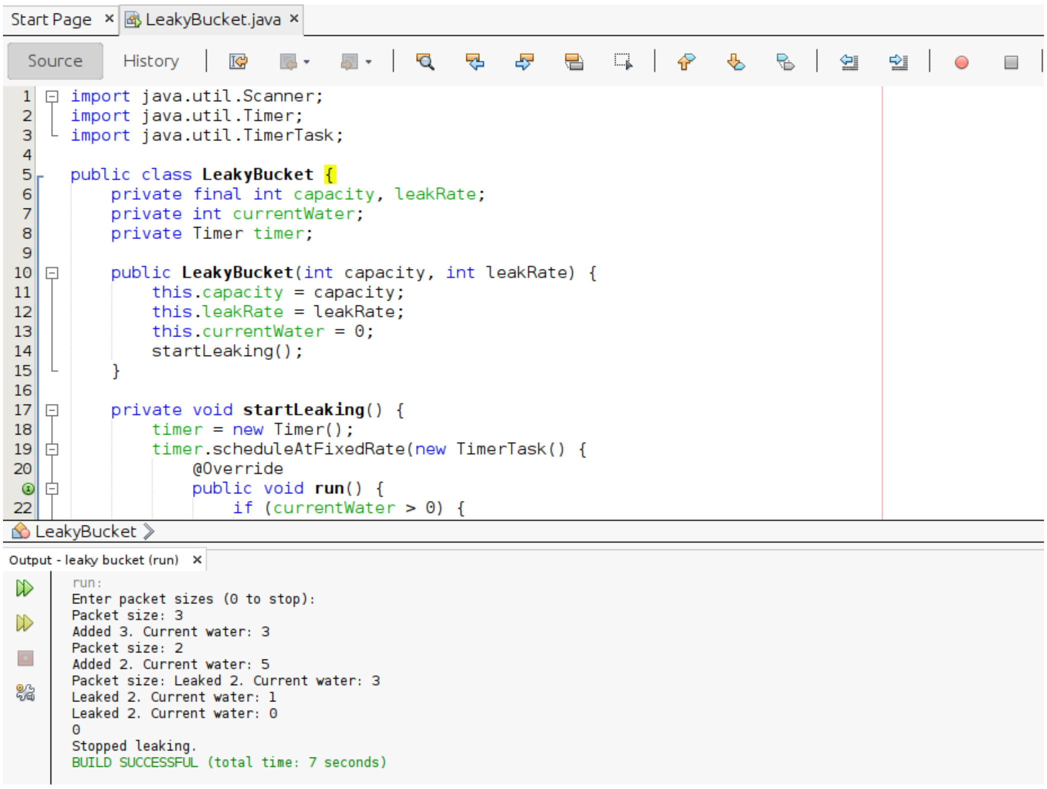
```
            // Convert the message to a BigInteger
            BigInteger messageBigInt = new BigInteger(message.getBytes());

            // Encrypt the message
            BigInteger encrypted = rsa.encrypt(messageBigInt);
            System.out.println("Encrypted message: " + encrypted);

            // Decrypt the message
            BigInteger decrypted = rsa.decrypt(encrypted);
            String decryptedMessage = new String(decrypted.toByteArray());
            System.out.println("Decrypted message: " + decryptedMessage);

            scanner.close();
        }
    }
```

```
run:
Enter a message to encrypt: Hi PESITM
Original message: Hi PESITM
Encrypted message: 412458952438954337899909235118525024568270146978893296742024679224620705408145088095788201208181440431286
Decrypted message: Hi PESITM
BUILD SUCCESSFUL (total time: 8 seconds)
```

10. Develop a program for congestion control using a leaky bucket algorithm.

```
import java.util.Scanner;
import java.util.Timer;
import java.util.TimerTask;

public class LeakyBucket {
    private final int capacity, leakRate;
    private int currentWater;
    private Timer timer;

    public LeakyBucket(int capacity, int leakRate) {
        this.capacity = capacity;
        this.leakRate = leakRate;
        this.currentWater = 0;
        startLeaking();
    }

    private void startLeaking() {
        timer = new Timer();
        timer.scheduleAtFixedRate(new TimerTask() {
            @Override
            public void run() {
                if (currentWater > 0) {
                    currentWater = Math.max(0, currentWater - leakRate);
                    System.out.println("Leaked " + leakRate + ". Current water: " + currentWater);
                }
            }
        }, 1000, 1000);
    }

    public void addPacket(int packetSize) {
        if (currentWater + packetSize <= capacity) {
            currentWater += packetSize;
            System.out.println("Added " + packetSize + ". Current water: " + currentWater);
        } else {
            System.out.println("Bucket overflow! Packet of size " + packetSize + " dropped.");
        }
    }

    public void stopLeaking() {
        if (timer != null) {
            timer.cancel();
            System.out.println("Stopped leaking.");
        }
    }

    public static void main(String[] args) {
        LeakyBucket bucket = new LeakyBucket(10, 2);
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter packet sizes (0 to stop):");

        while (true) {
            System.out.print("Packet size: ");
            if (scanner.hasNextInt()) {
                int packetSize = scanner.nextInt();
                if (packetSize == 0) break;
                bucket.addPacket(packetSize);
            } else {
                System.out.println("Invalid input. Please enter an integer.");
                scanner.next(); // Clear invalid input
            }
        }

        bucket.stopLeaking(); // Stop the leaking process
        scanner.close();
    }
}
```

Sample Output:



```
import java.util.Scanner;
import java.util.Timer;
import java.util.TimerTask;

public class LeakyBucket {
    private final int capacity, leakRate;
    private int currentWater;
    private Timer timer;

    public LeakyBucket(int capacity, int leakRate) {
        this.capacity = capacity;
        this.leakRate = leakRate;
        this.currentWater = 0;
        startLeaking();
    }

    private void startLeaking() {
        timer = new Timer();
        timer.scheduleAtFixedRate(new TimerTask() {
            @Override
            public void run() {
                if (currentWater > 0) {
```

```
run:
Enter packet sizes (0 to stop):
Packet size: 3
Added 3. Current water: 3
Packet size: 2
Added 2. Current water: 5
Packet size: Leaked 2. Current water: 3
Leaked 2. Current water: 1
Leaked 2. Current water: 0
0
Stopped leaking.
BUILD SUCCESSFUL (total time: 7 seconds)
```