**Title:** Implementation of mutual exclusion algorithm

1) Token Ring

   Code:

   TokenServer.java

```
package lab_04;

import java.net.*;

public class TokenServer {
        public static void main(String agrs[]) throws Exception {
                int port = 8000;
                System.out.println("Server Started on " + port);
                while (true) {
                        Server sr = new Server();
                        sr.recPort(port);
                        sr.recData();
                }
        }
}

class Server {

        boolean hasToken = false;
        boolean sendData = false;
        int recport;

        void recPort(int recport) {
                this.recport = recport;
        }

        void recData() throws Exception {
                byte buff[] = new byte[256];
                DatagramSocket ds;
                DatagramPacket dp;
                String str;

                ds = new DatagramSocket(recport);
                dp = new DatagramPacket(buff, buff.length);
                ds.receive(dp);
                ds.close();

                str = new String(dp.getData(), 0, dp.getLength());
                System.out.println("The message is " + str);
        }
}
```

TokenClient.java

```
package lab_04;

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

public class TokenClient {
        public static void main(String arg[]) throws Exception {
                InetAddress localhost;
                BufferedReader br;
                String str = "";
                TokenClientInside tokenClient, tokenServer;

                while (true) {
                        localhost = InetAddress.getLocalHost();
                        tokenClient = new TokenClientInside(localhost);
                        tokenServer = new TokenClientInside(localhost);
                        tokenClient.setSendPort(9004);
                        tokenClient.setRecPort(8002);
                        localhost = InetAddress.getLocalHost();
                        tokenServer.setSendPort(9000);

                        if (tokenClient.hasToken == true) {
                                System.out.println("Do you want to enter the Data –>
Yes/No?");
                                br = new BufferedReader(new
InputStreamReader(System.in));
                                str = br.readLine();
                                if (str.equalsIgnoreCase("yes")) {
```

```
					System.out.println("Ready to send");
					tokenServer.setSendData = true;
					tokenServer.sendData();
					tokenServer.setSendData = false;
				} else if (str.equalsIgnoreCase("no")) {
					System.out.println("Token Waiting...");
					tokenClient.sendData();
					tokenClient.recData();
				}
			} else {
				System.out.println("ENTERING RECEIVING
MODE...");
				tokenClient.recData();
			}
		}
	}

}

class TokenClientInside {
	InetAddress localhost;
	int sendPort, recPort;
	boolean hasToken = true;
	boolean setSendData = false;

	TokenClientInside(InetAddress localhost) {
		this.localhost = localhost;
	}

	void setSendPort(int sendPort) {
		this.sendPort = sendPort;
	}

	void setRecPort(int recPort) {
		this.recPort = recPort;
	}

	void sendData() throws Exception {
		BufferedReader br;
```

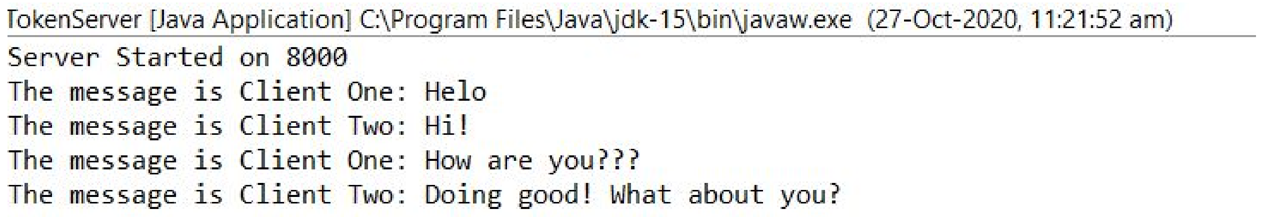
```
                String str = "Token";
                DatagramSocket ds;
                DatagramPacket dp;

                if (setSendData == true) {
                        System.out.println("Enter the Data:");
                        br = new BufferedReader(new
InputStreamReader(System.in));
                        str = "Client One: " + br.readLine();
                        System.out.println("Now sending...");

                }
                ds = new DatagramSocket(sendPort);
                dp = new DatagramPacket(str.getBytes(), str.length(), localhost,
sendPort - 1000);
                ds.send(dp);
                ds.close();
                setSendData = false;
                hasToken = false;
        }

        void recData() throws Exception {
                String msgstr;
                byte buffer[] = new byte[256];
                DatagramSocket ds;
                DatagramPacket dp;

                ds = new DatagramSocket(recPort);
                dp = new DatagramPacket(buffer, buffer.length);
                ds.receive(dp);
                ds.close();
                msgstr = new String(dp.getData(), 0, dp.getLength());
                System.out.println("The data is " + msgstr);

                if (msgstr.equals("Token")) {
                        hasToken = true;
                }
        }
```

```
}

TokenClient2.java

package lab_04;

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

public class TokenClient2 {
        static boolean setSendData;
        static boolean hasToken;

        public static void main(String arg[]) throws Exception {
                InetAddress localhost;
                BufferedReader br;
                String str1;
                TokenClientInside2 tokenClient;
                TokenClientInside2 Server;
                while (true) {
                        localhost = InetAddress.getLocalHost();
                        tokenClient = new TokenClientInside2(localhost);
                        tokenClient.setRecPort(8004);
                        tokenClient.setSendPort(9002);
                        localhost = InetAddress.getLocalHost();
                        Server = new TokenClientInside2(localhost);
                        Server.setSendPort(9000);
                        if (hasToken == true) {

                                System.out.println("Do you want to enter the Data –>
YES/NO");
                                br = new BufferedReader(new
InputStreamReader(System.in));
                                str1 = br.readLine();
                                if (str1.equalsIgnoreCase("yes")) {
                                        System.out.println("Ready to send");
                                        Server.setSendData = true;
                                        Server.sendData();
                                } else if (str1.equalsIgnoreCase("no")) {
                                        System.out.println("Token Waiting...");
                                        tokenClient.sendData();
                                        hasToken = false;
                                }
                        } else {
                                System.out.println("ENTERING RECIEVING
MODE...");
                                tokenClient.recData();
                                hasToken = true;
                        }
                }
        }
}

class TokenClientInside2 {
        InetAddress localhost;
        int sendPort, recPort;
        boolean setSendData = false;
        boolean hasToken = false;

        TokenClientInside2(InetAddress localhost) {
                this.localhost = localhost;
        }

        void setSendPort(int sendPort) {
                this.sendPort = sendPort;
        }

        void setRecPort(int recPort) {
                this.recPort = recPort;
        }

        void sendData() throws Exception {
                BufferedReader br;
                String str = "Token";
                DatagramSocket ds;
                DatagramPacket dp;

                if (setSendData == true) {
```

```
                System.out.println("Enter the Data");
                br = new BufferedReader(new
InputStreamReader(System.in));
                str = "Client Two: " + br.readLine();
                System.out.println("Now sending...");
            }
            ds = new DatagramSocket(sendPort);
            dp = new DatagramPacket(str.getBytes(), str.length(), localhost,
sendPort - 1000);
            ds.send(dp);
            ds.close();
            System.out.println("Data sent");
            setSendData = false;
            hasToken = false;

        }

        void recData() throws Exception {
            String msgstr;
            byte buffer[] = new byte[256];
            DatagramSocket ds;
            DatagramPacket dp;
            ds = new DatagramSocket(recPort);
            dp = new DatagramPacket(buffer, buffer.length);
            ds.receive(dp);
            ds.close();
            msgstr = new String(dp.getData(), 0, dp.getLength());
            System.out.println("The data is " + msgstr);
            if (msgstr.equals("Token")) {
                hasToken = true;
            }
        }

    }
```

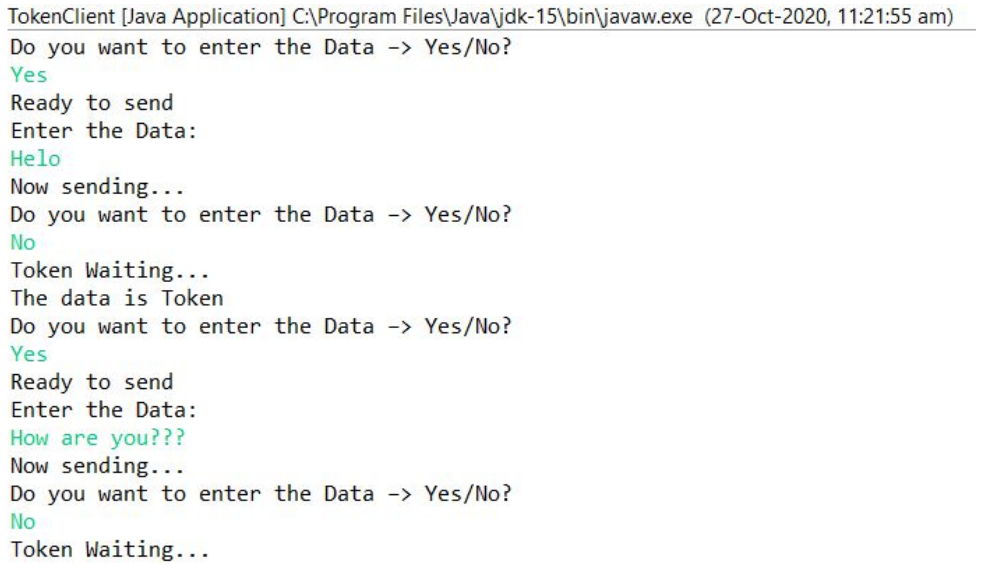
Screenshots:

Token Server:



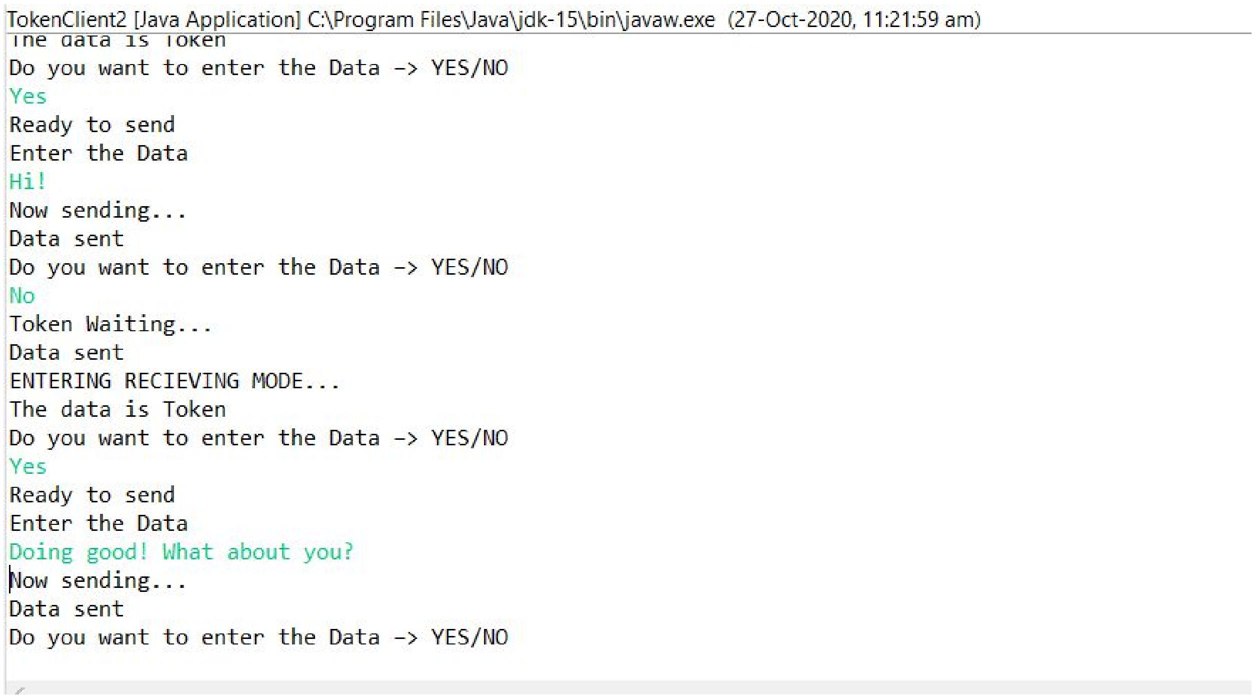
```
TokenServer [Java Application] C:\Program Files\Java\jdk-15\bin\javaw.exe  (27-Oct-2020, 11:21:52 am)
Server Started on 8000
The message is Client One: Helo
The message is Client Two: Hi!
The message is Client One: How are you???
The message is Client Two: Doing good! What about you?
```

TokenClient1:



```
TokenClient [Java Application] C:\Program Files\Java\jdk-15\bin\javaw.exe  (27-Oct-2020, 11:21:55 am)
Do you want to enter the Data -> Yes/No?
Yes
Ready to send
Enter the Data:
Helo
Now sending...
Do you want to enter the Data -> Yes/No?
No
Token Waiting...
The data is Token
Do you want to enter the Data -> Yes/No?
Yes
Ready to send
Enter the Data:
How are you???
Now sending...
Do you want to enter the Data -> Yes/No?
No
Token Waiting...
```

TokenClient2:



```
TokenClient2 [Java Application] C:\Program Files\Java\jdk-15\bin\javaw.exe  (27-Oct-2020, 11:21:59 am)
The data is Token
Do you want to enter the Data -> YES/NO
Yes
Ready to send
Enter the Data
Hi!
Now sending...
Data sent
Do you want to enter the Data -> YES/NO
No
Token Waiting...
Data sent
ENTERING RECIEVING MODE...
The data is Token
Do you want to enter the Data -> YES/NO
Yes
Ready to send
Enter the Data
Doing good! What about you?
Now sending...
Data sent
Do you want to enter the Data -> YES/NO
```

2) Ricart Agrawala:

Code:

```
package lab_04.RicartAgrawala;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.LinkedList;

public class RicartAgrawala extends Thread {
        int[] ports = { 8081, 8082, 8083, 8084 };
        boolean waiting = false, accessing = false;
        private int hi;
        LinkedList<Processus> Ri = new LinkedList<Processus>(), waitingList =
new LinkedList<Processus>(),
                        differedList = new LinkedList<Processus>();
```

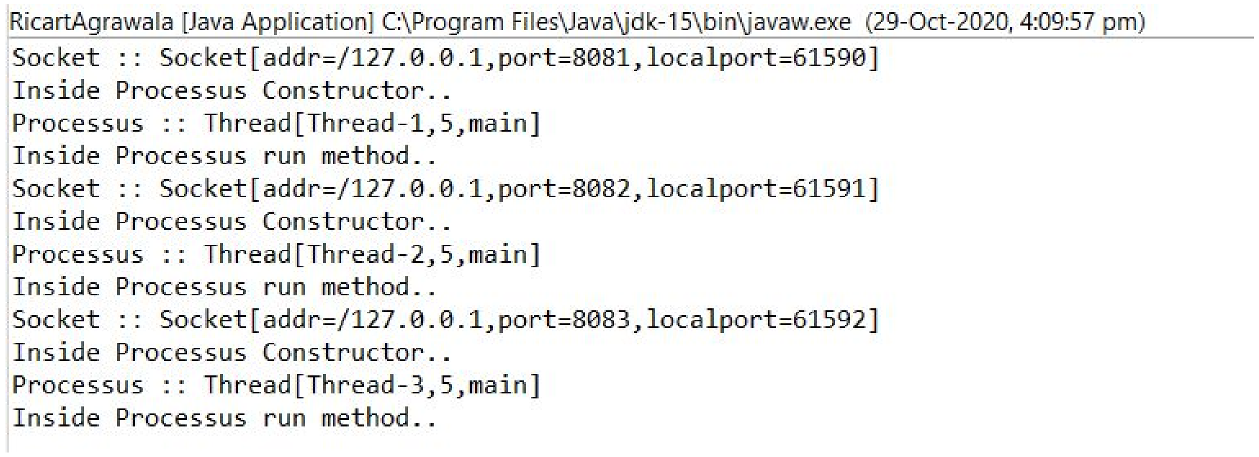
```
	LinkedList<Processus> RiL = new LinkedList<Processus>();
	int ID;
	int port;

	public static void main(String[] args) {
		int port = Integer.parseInt(args[0]);
		RicartAgrawala p = new RicartAgrawala(port);
		p.start();
		p.createServer();

	}

	public RicartAgrawala(int port) {
		this.port = port;
		this.ID = port;
	}

	public void run() {
		PrintWriter pw;
		try {
			sleep(5000);
		} catch (Exception e) {
			e.printStackTrace();
		}
		for (int port : ports) {
			try {
				if (port != ID) {
					Socket s = new Socket("127.0.0.1", port);
					System.out.println("Socket :: " + s.toString());
					Processus p = new Processus(s, port);
					System.out.println("Processus :: " +
p.toString());
					pw = new PrintWriter(s.getOutputStream(),
true);
					RiL.add(p);
					pw.println(ID);
					p.start();
				}
```

```
                } catch (Exception e) {
                        System.out.println(e.toString());
                        System.out.println("Error Connecting with Other
processes");
                }
        }
        while (true) {
                try {
                        System.in.read();
                        hi++;
                        waiting = true;
                        for (Processus p : Ri)
                                waitingList.add(p);
                        sendtoRi(String.valueOf(hi));

                        System.out.println("Asking Processes... ");
                        while (true) {
                                sleep(1500);
                                if (waitingList.size() == 0) {
                                        accessing = true;
                                        System.out.println("Accessing Critical
Section...");
                                        sleep(5000);
                                        System.out.println("Done Working on
Critical Section!");
                                        sendtoDiffere("OK");
                                        differedList.clear();
                                        waiting = false;
                                        accessing = false;
                                        break;
                                }

                        }
                } catch (Exception e) {
                }
        }

    }
```

```
	public void createServer() {
		try (ServerSocket server = new ServerSocket(port)) {
			Processus p;
			while (true) {
				Socket s = server.accept();
				BufferedReader input = new BufferedReader(new
InputStreamReader(s.getInputStream()));
				int id = Integer.parseInt(input.readLine());
				p = new Processus(s, id);
				Ri.add(p);
				System.out.println(id + " is Successfully Connected.");
				sleep(500);
				p.start();
			}
		} catch (Exception e) {
			e.printStackTrace();
		}
	}

	public void sendtoRi(String message) {
		for (Processus p : Ri)
			p.sendMessage(ID + ":" + message);
	}

	public void sendTo(int x, String message) {
		for (Processus p : Ri)
			if (p.getIdP() == x)
				p.sendMessage(ID + ":" + message);
	}

	public void sendtoAttendu(String message) {
		for (Processus p : waitingList)
			p.sendMessage(ID + ":" + message);
	}

	public void sendtoDiffere(String message) {

		for (Processus p : differedList)
			p.sendMessage(ID + ":" + message);
	}

	class Processus extends Thread {
		BufferedReader input;
		PrintWriter output;
		String msg;
		int id;

		public Processus(Socket client, int id) {
			this.id = id;
			System.out.println("Inside Processus Constructor..");
			try {
				output = new PrintWriter(client.getOutputStream(),
true);
				input = new BufferedReader(new
InputStreamReader(client.getInputStream()));
			} catch (IOException e) {
				e.printStackTrace();
			}
		}

		public int getIdP() {
			return id;
		}

		public void sendMessage(String str) {
			output.println(str);
		}

		public void run() {
			System.out.println("Inside Processus run method..");
			while (true) {
				try {
					String msg = input.readLine();
					System.out.println("Message Received : " +
msg);
					String msgT[] = msg.split(":");
					int rId = Integer.parseInt(msgT[0]);
					if (msgT[1].equals("OK")) {
```

```
                                        for (Processus p : Ri)
                                            if (p.getIdP() == rId)
                                                waitingList.remove(p);
                                    } else {
                                        int rHi = Integer.valueOf(msgT[1]);
                                        if (accessing || (waiting && hi > rHi)) {
                                            for (Processus p : Ri)
                                                if (p.getIdP() == rId)
                                                    differedList.add(p);
                                        } else {
                                            if (hi < rHi)
                                                hi = rHi;
                                            sendTo(rId, "OK");
                                        }
                                    }
                                } catch (IOException e) {
                                    for (Processus p : Ri)
                                        if (p.getIdP() == id)
                                            Ri.remove(p);
                                    for (Processus p : RiL)
                                        if (p.getIdP() == id)
                                            RiL.remove(p);
                                    for (Processus p : waitingList)
                                        if (p.getIdP() == id)
                                            RiL.remove(p);
                                    for (Processus p : differedList)
                                        if (p.getIdP() == id)
                                            RiL.remove(p);
                                    System.out.println(id + "Error! Socket will be
  closed immediatly");
                                    break;
                                }

                            }
                        }

                    }
  }
```

Screenshot:



RicartAgrawala [Java Application] C:\Program Files\Java\jdk-15\bin\javaw.exe (29-Oct-2020, 4:09:57 pm)

```
Socket :: Socket[addr=/127.0.0.1,port=8081,localport=61590]
Inside Processus Constructor..
Processus :: Thread[Thread-1,5,main]
Inside Processus run method..
Socket :: Socket[addr=/127.0.0.1,port=8082,localport=61591]
Inside Processus Constructor..
Processus :: Thread[Thread-2,5,main]
Inside Processus run method..
Socket :: Socket[addr=/127.0.0.1,port=8083,localport=61592]
Inside Processus Constructor..
Processus :: Thread[Thread-3,5,main]
Inside Processus run method..
```