컴퓨터네트워크 FinalExam

소프트웨어전공 20181658 이성진

Priogramming language – JAVA

File명

- FinalSender_20181658이성진.java
- FinalReceiver_20181658이성진.java

How to run (과제 이클립스 idle 이용해서 코딩)

- **FinalReceiver_20181658이성진.java를 먼저 실행 시킨 후, FinalSender_20181658이성진.java 실행**

실행시 입력 data

기본으로 "11111111" 8자리로 맞춰 실행되게 해놨습니다.
채점하실 때 입력을 하는 기준이라면,

- FinalSender_20181658이성진.java

파일의 38 ~ 42 주석 해제, 43 주석 하시고
실행시켜 8자리 숫자 입력해주시면 감사합니다!

```
class ApplicationLyaer extends Thread {
    private String t;

    public ApplicationLyaer(String t) {
        this.t = t;
    }

    public void run() {
        /*
        Scanner s = new Scanner(System.in);
        System.out.println("전송하고 싶은 8자리 비트 입력: ");
        String message = s.nextLine();
        */
        String message = "11111111";
        System.out.println("ApplicationLayer -> " + message + " 전송");
        System.out.println("-------------------------------------------------");
```

Code block의 구조 및 기능 설명

① Layer 들은 Thread를 사용해서 생성하였습니다.

② Layer간의 IPC는 message queue (blockingqueue)를 이용해서 구현하였습니다.

③ PhysicalLayer Sender-Receiver 는 socket 을 이용해서 구현하였습니다.

Sender 파일의 import 모듈들입니다.
Socket 통신을 할 때 사용한 1~6 ,
CSMA/CD 구현할때 사용한 7
입력 받을 때 사용한 8
IPC 구현할때 사용한 blockingqueue 9~10 입니다

```
1  import java.io.DataInputStream;
2  import java.io.DataOutputStream;
3  import java.io.IOException;
4  import java.io.InputStream;
5  import java.io.OutputStream;
6  import java.net.Socket;
7  import java.util.Random;
8  import java.util.Scanner;
9  import java.util.concurrent.ArrayBlockingQueue;
10 import java.util.concurrent.BlockingQueue;
11
```

.

IPC 통신 – messagequeue 를 하기위한

Blockingqueue 클래스를 만들어
그 안에 queue를 선언해주었습니다.
FinalSender_20181658이성진 클래스
아래에 메인을 만들고
Application thread를 실행시켰습니다

```
3  class CmnVar {
4      public static BlockingQueue<String> queue = new ArrayBlockingQueue<String>(1);
5  }
6
7  class FinalSender_20181658이성진 {
8
9      public static void main(String args[]) {
0
1          Scanner s = new Scanner(System.in);
2          String to = "";
3          ApplicationLyaer t1 = new ApplicationLyaer(to);
4          t1.start();
5      }
6
7  }
```

Sender쪽부터 시작하겠습니다.

<Applicaion>

쓰레드로 구현하였습니다.

보내고자 할 패킷을

"11111111"로 임시로 지정했습니다

(초록색 주석 풀고 입력 받고

다시 실행시킬 수 있습니다)

< Receiver -> Sender 순으로 실행>

위에서 만들어놓은 queue에

보내고자 할 패킷을 넣어놓습니다

CmnVar.queue.put()

TransportLayer를 실행하기위해

쓰레드를 시작합니다.

TransportLayer가 끝난 후 Application

이 실행되어야하기 때문에 join()을

사용했습니다. Join() 아래는 무사히

다시 돌아오면 출력메시지입니다.

```
class ApplicationLyaer extends Thread {
    private String t;

    public ApplicationLyaer(String t) {
        this.t = t;
    }

    public void run() {
        /*
        Scanner s = new Scanner(System.in);
        System.out.println("전송하고 싶은 8자리 비트 입력: ");
        String message = s.nextLine();
        */
        String message = "11111111";
        System.out.println("ApplicationLayer -> " + message + " 전송");
        System.out.println("-----------------------------------------------------");
        try {
            CmnVar.queue.put(message);
        } catch (InterruptedException e1) {
            // TODO Auto-generated catch block
            e1.printStackTrace();
        }
        TransportLayer t2 = new TransportLayer("");
        t2.start();
        try {
            t2.join();
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        try {
            t  = CmnVar.queue.take();
        } catch (InterruptedException e1) {
            // TODO Auto-generated catch block
            e1.printStackTrace();
        }
        System.out.println("ApplicaionLayer ->" + t + " 메시지를 받았다. 잘 도착했다고 한다 ! ! !  끗 ");
```

< TransportLayer >

CmnVar.queue.take() 로

Queue 넣어놧던 메시지를 꺼내

전송받음을 확인합니다.

NetworkLayer로 전송하기위해

출력메시지를 보내고 queue에

다시 담아 보내며 보냈으니 잠시

Stop message를 띄워줍니다

NetworkLayer 쓰레드를 시작하고

Join()을 걸어줍니다.

Join 아래 코드는 다시 되돌아올 때

사용하는 코드니 밑에서 설명하겠

습니다.

```
class TransportLayer extends Thread {
    private String t;
    String message;

    public TransportLayer(String t) {
        this.t = t;
    }

    public void run() {
        try {
            message = CmnVar.queue.take();
            System.out.println("TransportLayer -> " + message + " 전송 받음 ");
        } catch (InterruptedException e1) {
            // TODO Auto-generated catch block
            e1.printStackTrace();
        }
        System.out.println("TransportLayer -> " + message + " 전송");
        try {
            CmnVar.queue.put(message);
            System.out.println("stop.. wait! ");
        } catch (InterruptedException e1) {
            // TODO Auto-generated catch block
            e1.printStackTrace();
        }
        System.out.println("-----------------------------------------------------");
        NetworkLayer t33 = new NetworkLayer("");
        t33.start();
        try {
            t33.join();
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
```

< **NetworkLayer** >

전송 받고 다음 레이어로
재전송합니다.

DatalinkLayer 쓰레드를
시작하고
Join을 걸어줍니다.

```
class NetworkLayer extends Thread {
    private String t;
    String message;

    public NetworkLayer(String t) {
        this.t = t;
    }

    public void run() {
        try {
            message = CmnVar.queue.take();
            System.out.println("NetworkLayer -> " + message + " 전송 받음 ");
        } catch (InterruptedException e1) {
            // TODO Auto-generated catch block
            e1.printStackTrace();
        }
        System.out.println("NetworkLayer -> " + message + " 전송");
        try {
            CmnVar.queue.put(message);
        } catch (InterruptedException e1) {
            // TODO Auto-generated catch block
            e1.printStackTrace();
        }
        System.out.println("-------------------------------------------------");
        DataLinkLayer t3 = new DataLinkLayer("");
        t3.start();
        try {
            t3.join();
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
```

< DataLinkLayer >

코드가 길어 짤라서 첨부하겠습니다.
NetworkLayer에서 전송하기 위해
Queue에 넣어놓은 message를 꺼냅니다.
꺼낸 message를 bitstuffing 하기위해 p
에 -> bitw 로 넣어줍니다
(같은 일을 두번하네요.....)
Bitstuffing 한 결과를 출력해주고
Simpleprotocol 로 MAC에게 전달해줍니다.

```
class DataLinkLayer extends Thread {
    private String p;
    String message;

    public DataLinkLayer(String p) {
        this.p = p;
    }

    public void run() {
        try {
            message = CmnVar.queue.take();
            System.out.println("DataLinkLayer -> " + message + " 전송 받음 ");
        } catch (InterruptedException e1) {
            // TODO Auto-generated catch block
            e1.printStackTrace();
        }
        p = message;
        String bitw = p;
        String bitsuffer = bitw.replace("11111", "111110");
        System.out.println("DataLinkLayer -> bitstuffing: " + bitsuffer);
        System.out.println("Simple protocol로 Mac Layer에게로 전송");
```

위 코드는 DLC , 아래코드는 MAC 부분입니다.

```
System.out.println("MacLayer -> csma/cd start: ");

Scanner s = new Scanner(System.in);
Random rd = new Random();
boolean a = true;
boolean b = true;
int rdnum;
int perrdnum;
int rdcollision;
int R;
int Success = 0;
int Kmax = 1;
int k = 0;
```

CSMA/CD에 사용된
변수들이며 MAC sublayer입니다.

While(b) {
Persistence가 랜덤을 이용해서 홀수면 idel
짝수면 busy 로 true, false를 이용해 idel이
나올때까지 반복합니다.}
While(a) {
Rdnum % 2 == 1을 이용해 홀수면
transmission done, rdcollision 랜덤으로 발생
홀수면 Not collision, 짝수면 collision이라

```
while(k<=Kmax) {
    while(b) {
        perrdnum = rd.nextInt(10)+1;
        if(perrdnum % 2 == 1) {
            System.out.println("persistence is idel");
            b = false;
        }
        else {
            System.out.println("persistence is busy, wait, Try again");
            b = true;
        }
    }
    while(a) {
        System.out.println("Transmission done?");
        for(int i=0;i<1;i++) {
            rdnum = rd.nextInt(10)+1;
            rdcollision = rd.nextInt(10)+1;
            if(rdnum % 2 == 1) { //홀수면 transmsiion done...
                System.out.println("Yes !!! done !! ");
                if(rdcollision % 2 == 1) { //홀수면
                    System.out.println("Not Collision");
                    Success = 1;
                    k = Kmax + 1;
                }
                else {
                    System.out.println("Collision");
                    System.out.println("Send a jamming signal");
                    Success = 0;
                    k = k +1;
                }
                a = false;
            }
            else {
                System.out.println("NO !!!! Transmit and receive");
                i=0;
            }
        }
    }
```

Jamming signal을 보냅니다.

Rdnum % 2 == 1을 이용해 짝수면

```
        }
        else {
            System.out.println("NO !!!! Transmit and receive");
            i=0;
        }
    }
}
if(k>Kmax && Success == 0) {
    //System.out.println("Abort !");
    k--;
}
else if(k>Kmax && Success == 1){
    System.out.println("Success!!");
    System.out.println("MacLayer -> csma/cd end: ");
    System.out.println("-------------------------------------------------");
}

try {
    CmnVar.queue.put(bitsuffer);
} catch (InterruptedException e1) {
    // TODO Auto-generated catch block
    e1.printStackTrace();
}
PhysicalLayer t4 = new PhysicalLayer("");
t4.start();
try {
    t4.join();
} catch (InterruptedException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
```

Transmit and receivet를 합니다.

위에서 collision 여부에 따라

증가시킨 Success와 k>Kmax에 따라

1. Abort ! (과제가 No free error) 이기때문에 주석하고 k—감소시키고 다시 진행하였습니다.

Success인 경우 PhysicalLayer로 보내기 위해 queue에 담아준 뒤

PhysicalLayer 쓰레드를 시작하고

Join()을 통해서 기다립니다.

(join()아래는 다시 돌아올 때 설명)

<PhysicalLayer >

CmnVar.queue.take()를 이용해

전송 받은 message를 꺼내주고

Message를 t에 넣어 mlt를 할 준비를합니다.

String t 를 String mit[] 에 넣어줍니다.

After[] 은 mlt 적용 후 저장하는 곳입니다.

H 는 현재위치, nonzero는 부호 변화 판단 변수 입니다. Mlt 규칙을 통해 변환합니다.

변환하고 sendmessage에 저장합니다.

(소켓을 통해 sendmessage 리시버로 전송 예정)

```
class PhysicalLayer extends Thread {
    private String t;
    private Socket socket = null;
    private OutputStream outputStream;
    private DataOutputStream dataOutputStream;
    private InputStream inputStream;
    private DataInputStream dataInputStream;
    static String ackmessage = "";
    String message;

    public PhysicalLayer(String t) {
        this.t = t;
    }
    public void run() {
        try {
            message = CmnVar.queue.take();
            System.out.println("PhysicalLayer -> " + message + " 전송 받음 ");
        } catch (InterruptedException e1) {
            // TODO Auto-generated catch block
            e1.printStackTrace();
        }
        //System.out.println("PhysicalLayer -> " + t + " 입력받음");
        t = message;
        String mlt[] = t.split("");
        String after[] = new String[t.length()];
        String h = " ";
        String nonzero = " ";
```

```
for(int i =0; i<mlt.length; i++){
    if(i == 0) {
        if(mlt[i].equals("0")) {
            after[i] = mlt[i];
            h = mlt[i];
            nonzero = "-";
        }
        else {
            after[i] = "+";
            h = "+";
            nonzero = "-";
        }
    }
    else if(i > 0) {
        if((h.equals("0")) && (mlt[i].equals("1")))  {
            if(nonzero.equals("-")){
                after[i] = "+";
                nonzero = "+";
                h = "+";
            }
            else {
                after[i] = "-";
                nonzero = "-";
                h = "-";
            }
        }
        else if((h.equals("0")) && (mlt[i].equals("0"))) {
            after[i] = "0";
            h = "0";
        }
        else if( (h.equals("+")) ) {
            if(mlt[i].equals("1")) {
                after[i] = "0";
                nonzero = "+";
                h = "0";
            }
            else {
                after[i] = "+";
                nonzero = "+";
                h = "+";
            }
        }
        else if( (h.equals("-"))) {
            if(mlt[i].equals("1")) {
                after[i] = "0";
                nonzero = "-";
                h = "0";
            }
            else {
                after[i] = "-";
                nonzero = "-";
                h = "-";
            }
        }
    }
}
System.out.print("MLT 처리 결과: ");
for(int i = 0; i<after.length; i++) {
    System.out.print(after[i]);
}
System.out.println();
StringBuffer result = new StringBuffer();
for (int i = 0; i < after.length; i++) {
   result.append( after[i] );
}
String sendmessage = result.toString();
```

Socket을 이용해
Sender의 PhysicalLayer에서
Receiver의 PhysicalLayer로
전송해줍니다.

```
for (int i = 0; i < after.length; i++) {
   result.append( after[i] );
}
String sendmessage = result.toString();

try {
    socket = new Socket("localhost", 8000);
    outputStream = socket.getOutputStream();
    dataOutputStream = new DataOutputStream(outputStream);

    inputStream = socket.getInputStream();
    dataInputStream = new DataInputStream(inputStream);
    while(true) {
        dataOutputStream.writeUTF(sendmessage);
        dataOutputStream.flush();
        System.out.println(sendmessage + "receiver PhysicalLayer로 전송");
        break;
    }
} catch(IOException e) {
    e.printStackTrace();
}finally {
    try {
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

< Receiver >

```
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.Random;
import java.util.Scanner;
import java.util.concurrent.ArrayBlockingQueue;
import java.util.concurrent.BlockingQueue;
```

Sender와 동일, ServerSocket만 추가

```
class FinalReceiver_20181658이성진 {
    public static void main(String[] args) {
        String ok = "";
        RecPhysicalLayer t6 = new RecPhysicalLayer(ok);
        t6.start();
    }
}

class CmnVarRec {
    public static BlockingQueue<String> queue = new ArrayBlockingQueue<String>(1);
}
```

Receiver main에서 Physical Thread 실행
IPC 통신 - messagequeue CmnVarRec 큐 제작

<Receiver – PhysicalLayer >

Sender PhysicalLayer에서 소켓통신
으로 전송한 메시지를
Receiver RecPhysicalLayer에서
String clientMessage = dataInputStream.
readUTF() 로 받아옵니다.
받아온 clientmessage를 t 에 넣습니다.
그리고 mlt reverse를 하기 위해 mlt[],
After[] 배열을 만들어줍니다.

```
class RecPhysicalLayer extends Thread {
    private String t;
    private ServerSocket serverSocket = null;
    private Socket socket = null;
    private OutputStream outputStream = null;
    private DataOutputStream dataOutputStream = null;
    private InputStream inputStream = null;
    private DataInputStream dataInputStream = null;

    public RecPhysicalLayer(String t) {
        this.t = t;
    }
    public void run() {
        try {
            serverSocket = new ServerSocket(8000);
            System.out.println("Sender로부터 데이터 전송받을 준비 완료");

            socket = serverSocket.accept();
            System.out.println("Sender 연결 완료");
            System.out.println("socket : " + socket);

            inputStream = socket.getInputStream();
            dataInputStream = new DataInputStream(inputStream);

            outputStream = socket.getOutputStream();
            dataOutputStream = new DataOutputStream(outputStream);

            while (true) {
                    String clientMessage = dataInputStream.readUTF();
                    t = clientMessage;
                    break;
              }
           } catch (Exception e) {
              e.printStackTrace();
           } finally {
              try {
              } catch (Exception e) {
                  e.printStackTrace();
              }
           }
           System.out.println("-------------------------------------------------");
           System.out.println("Receiver의 PhysicalLayer");
           System.out.println("sender의 PhysicalLayer로부터 " + t + " 도착");
           String mlt[] = t.split("");
           String after[] = new String[t.length()];
           String eql = " ";
```

MLT reverse를 합니다.
i 가 0과같으면 0,
+ 와 같으면 1
I와 i-1을 비교해봤을 때 같으면 0
다르면 1 로 구현했습니다.
Reverse 한 결과를 String unstuff에 넣고 CmnVarRec queue에 담습니다.
DataLinkLayer 쓰레드를 시작하고 Join()을 걸어 기다립니다.

```
for(int i =0; i<mlt.length; i++){
    if(i == 0) {
        if(mlt[i].equals("0")) {
            after[i] = "0";
            eql = mlt[i];
        }
        else if(mlt[i].equals("+")){
            after[i] = "1";
            eql = mlt[i];
        }
    }
    else {
        if(mlt[i].equals(eql)) {
            after[i] = "0";
            eql = mlt[i];
        }
        else {
            after[i] = "1";
            eql = mlt[i];
        }
    }
}
System.out.print("MLT -> bit stream: ");
for(int i = 0; i<after.length; i++) {
    System.out.print(after[i]);
}

System.out.println();
String unstuff = String.join("", after);
System.out.println("Data Link로 " + unstuff + "전송");
try {
    CmnVarRec.queue.put(unstuff);   // queue에 넣어서 datalink로 전송
} catch (InterruptedException e1) {
    // TODO Auto-generated catch block
    e1.printStackTrace();
}
System.out.println("-----------------------------------------------------");
RecDataLinkLayer t7 = new RecDataLinkLayer("");
t7.start();
try {
    t7.join();
} catch (InterruptedException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
```

< Receiver DataLinkLayer >
PhysicalLayer에서 queue에 넣어놓은 걸 Queue.take() 해서 꺼내 p에 저장합니다.
P를 bitw에 넣어 bitunstuffing을 한 후 Bitunsuffer에 넣어줍니다. 넣어준 값을 NetworkLayer로 보내기위해 queue에 다시 넣고 NetworkLayer 쓰레드를 시작하고 Join()을 기다립니다.

```
class RecDataLinkLayer extends Thread {
    private String p;

    public RecDataLinkLayer(String p) {
        this.p = p;
    }

    public void run() {
        try {
            p = CmnVarRec.queue.take();
            System.out.println("DataLinkLayer -> " + p + " 전송 받음 ");
        } catch (InterruptedException e1) {
            // TODO Auto-generated catch block
            e1.printStackTrace();
        }
        String bitw = p;
        String bitunsuffer = bitw.replace("111110", "11111");
        System.out.println("unstuffing: " + bitunsuffer);
        System.out.println("NetworkLayer로 " + bitunsuffer + "전송");
        try {
            CmnVarRec.queue.put(bitunsuffer);   // queue에 넣어서 NetworkLayer 로 전송
        } catch (InterruptedException e1) {
            // TODO Auto-generated catch block
            e1.printStackTrace();
        }
        System.out.println("-----------------------------------------------------");
        RecNetworkLayer t8 = new RecNetworkLayer("");
        t8.start();
        try {
            t8.join();
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
```

<NetworkLayer >

Datalink에서 온걸 take 한뒤 확인하고 다시 put 해서 Transport 쓰레드를 시작하고 Join을 걸어줍니다.

```
class RecNetworkLayer extends Thread {
    private String p;

    public RecNetworkLayer(String p) {
        this.p = p;
    }

    public void run() {
        try {
            p = CmnVarRec.queue.take();
            System.out.println("NetworkLayer -> " + p + " 전송 받음 ");
        } catch (InterruptedException e1) {
            // TODO Auto-generated catch block
            e1.printStackTrace();
        }
        System.out.println("NetworkLayer -> " + p + "전송중");
        try {
            CmnVarRec.queue.put(p);
        } catch (InterruptedException e1) {
            // TODO Auto-generated catch block
            e1.printStackTrace();
        }
        System.out.println("-----------------------------------------------------");
        RecTransportLayer t9 = new RecTransportLayer("");
        t9.start();
        try {
            t9.join();
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
```

<TransportLayer >

Network에서 넣은 queue를 꺼내서 ACK를 생성한 뒤 Applicaion으로 전송합니다. Applicaion 쓰레드를 시작하고 Join()을 걸어줍니다.

```
class RecTransportLayer extends Thread {
    private String p;
    static String ackmessage = "";
    String str;

    public RecTransportLayer(String p) {
        this.p = p;
    }

    public void run() {
        try {
            p = CmnVarRec.queue.take();
        } catch (InterruptedException e1) {
            // TODO Auto-generated catch block
            e1.printStackTrace();
        }
        System.out.println("Transport -> " + p + "도착");
        System.out.println("ACK 생성");
        System.out.println("Application으로 " + p + "전송");
        System.out.println("-----------------------------------------------------");
        RecApplicaionLayer t10 = new RecApplicaionLayer("");
        try {
            CmnVarRec.queue.put(p);
        } catch (InterruptedException e1) {
            // TODO Auto-generated catch block
            e1.printStackTrace();
        }
        t10.start();
        try {
            t10.join();
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
```

<Applicaion >

Transport에서 넣은 queue값을 꺼내 확인한 후 수신확인이라고 print합니다. Application void run()이 종료되면 Join() 하고있던 Transport가 실행됩니다.

```
class RecApplicaionLayer extends Thread {
    private String p;

    public RecApplicaionLayer(String p) {
        this.p = p;
    }

    public void run() {
        try {
            p = CmnVarRec.queue.take();
        } catch (InterruptedException e1) {
            // TODO Auto-generated catch block
            e1.printStackTrace();
        }
        System.out.println("Application -> " + p + "도착");
        System.out.println(p + "수신 확인");
        System.out.println("-----------------------------------------------------");


    }
}
```

<TransportLayer>

Join() 으로 기다리고 있던

그 이후부터 ACK 메시지를

Text 에 넣어

A, C, K 를 char 로 바꾼 후

10진수로 -> 2진수로 차례대로 바꿔

준 후 ackmessage에 최종적으로

ACK 를 2진수로 바꾼 값이 들어갑니디

Queue.put 에 ackmessage를 넣습니다

Run void() 가 끝나면

join()으로 기다리고 있던

NetworkLayer가 실행됩니다.

```
        }
        t10.start();
        try {
            t10.join();
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        System.out.println("TransportLayer -> ACK to ASCII");
        String text = "ACK";
        for(int i=0; i< text.length(); i++) {
            System.out.print(text.charAt(i) + "-> ");
            int tmp = ((int)text.charAt(i));
            System.out.print(tmp + "-> ");
            str = Integer.toBinaryString(tmp);
            System.out.println(str);
            ackmessage += str;
        }
        System.out.println("Ack to ASCII-> " + ackmessage);
        System.out.println("NetworkLayer로 " + ackmessage + "전송");
        try {
            CmnVarRec.queue.put(ackmessage);
        } catch (InterruptedException e1) {
            // TODO Auto-generated catch block
            e1.printStackTrace();
        }
        System.out.println("-----------------------------------------------------");
    }
```

<NetworkLayer >

join()으로 t9 (TransportLayer) 가

끝날 때 까지 기다린 후

tranport에서 넣은 queue안에 값을

꺼내고 지나가는 중이라는 메시지를

출력하고 다시 queue에 넣어줍니다.

```
        t9.start();
        try {
            t9.join();
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        try {
            p = CmnVarRec.queue.take();
        } catch (InterruptedException e1) {
            // TODO Auto-generated catch block
            e1.printStackTrace();
        }
        System.out.println("NetworkLayer -> " + p + " 지나가는중");
        System.out.println("-----------------------------------------------------");
        try {
            CmnVarRec.queue.put(p);
        } catch (InterruptedException e1) {
            // TODO Auto-generated catch block
            e1.printStackTrace();
        }

    }
```

<DataLinkLayer>

join으로 networklayer가 끝나기를

기다림이 끝난 이후 코드이다.

networklayer에서 queue에다 넣은

값을 빼서 bitstuffing을 한 후 SimpleProto

Col로 queue에 넣고

Mac sublayer로 전송해준다.

```
        System.out.println("-----------------------------------------------------");
        RecNetworkLayer t8 = new RecNetworkLayer("");
        t8.start();
        try {
            t8.join();
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

        try {
            p = CmnVarRec.queue.take();
        } catch (InterruptedException e1) {
            // TODO Auto-generated catch block
            e1.printStackTrace();
        }
        System.out.println("DataLinkLayer -> " + p + "전송 받음");
        String bitsuffer = p.replace("11111", "111110");
        System.out.println("DataLinkLayer -> bitstuffing: " + bitsuffer);
        try {
            CmnVarRec.queue.put(bitunsuffer);
            System.out.println("Simple Protocol로 MAC에게 전송");// queue에 넣어서 MacSublayer
        } catch (InterruptedException e1) {
            // TODO Auto-generated catch block
            e1.printStackTrace();
        }
        try {
            CmnVarRec.queue.take(); // Mac sublayer에서 꺼냄
        } catch (InterruptedException e1) {
            // TODO Auto-generated catch block
            e1.printStackTrace();
        }
```

Csma/cd 과정은 sender에서
보여준 과정이랑 같기 때문에 생략하고
Csma/cd 가 끝난후 queue에 넣습니다.

```
            k--;
        }
        else if(k>Kmax && Success == 1){
            System.out.println("Success!!");
            System.out.println("MacLayer -> csma/cd end: ");
            System.out.println("----------------------------------------
            try {
                CmnVarRec.queue.put(bitsuffer);
            } catch (InterruptedException e1) {
                // TODO Auto-generated catch block
                e1.printStackTrace();
            }
```

<PhysicalLayer>

DataLinklayer가 끝나면 join()으로
기다리고 있던 physicallayer가
시작됩니다.
Queue에 있는걸 꺼내와 t 에 넣은
후 mlt2 를 해줍니다.
(sender에서 한 코드와 동일하여
생략하고 MLT 처리 결과 나온 부분부터
작성하겠습니다)

```
try {
    t7.join();
} catch (InterruptedException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
try {
    t = CmnVarRec.queue.take();
} catch (InterruptedException e1) {
    // TODO Auto-generated catch block
    e1.printStackTrace();
}
System.out.println("PhysicalLayer -> " + t + "전송 받음");
String mlt2[] = t.split("");
String after2[] = new String[t.length()];
String h = " ";
String nonzero = " ";

for(int j =0; j<mlt2.length; j++){
    if(j == 0) {
        if(mlt[j].equals("0")) {
            after2[j] = mlt2[j];
            h = mlt2[j];
            nonzero = "-";
        }
        else {
            after2[j] = "+";
            h = "+";
            nonzero = "-";
        }
    }
    else if(j > 0) {
        if((h.equals("0")) && (mlt2[j].equals("1")))  {
            if(nonzero.equals("-")){
                after2[j] = "+";
```

mlt처리결과를 sendmessage2 에 넣어
소켓으로 Receiver의 Physical Layer에서
Sender의 Physical Layer로
dataOutputStread.writeUTF(send
message2) // 전송
dataOutputStream.flush()
// 비우기
해서 보내준뒤 sedner에게 전송
했다고 메시지를 출력합니다.

```
System.out.print("MLT 처리 결과: ");
for(int j = 0; j<after2.length; j++) {
    System.out.print(after2[j]);
}
System.out.println();
StringBuffer result = new StringBuffer();
for (int j = 0; j < after2.length; j++) {
   result.append( after2[j] );
}
String sendmessage2 = result.toString();
try {

    inputStream = socket.getInputStream();
    dataInputStream = new DataInputStream(inputStream);

    outputStream = socket.getOutputStream();
    dataOutputStream = new DataOutputStream(outputStream);

    while (true) {
            dataOutputStream.writeUTF(sendmessage2);
            dataOutputStream.flush();
            System.out.println(sendmessage2 + "sender에게 전송");
            System.out.println("----------------------------------------------------");
            break;
    }

} catch (Exception e) {
    e.printStackTrace();
} finally {
    try {
        if (dataOutputStream != null) dataOutputStream.close();
        if (outputStream != null) outputStream.close();
        if (dataInputStream != null) dataInputStream.close();
        if (inputStream != null) inputStream.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Receiver의 할일은 끝났습니다
그럼 다시 Sender로 가겠습니다.

< Sender Physical >

Receiver가 보낸 message를
t 에다 넣어준 후
mlt.reverse를 돌려줍니다.

mlt reverse 결과를 출력하고
queue에 담습니다.
(Sender에서 맨처음 만들어놓은
CmnVar Queue)
PhysicalLayer가 할일이 끝났습니다.

```
try {
    while(true) {
        String ServerMessage = dataInputStream.readUTF();
        t = ServerMessage;
        break;
    }
} catch(IOException e) {
    e.printStackTrace();
}finally {
    try {
        if (dataOutputStream != null) dataOutputStream.close();
        if (outputStream != null) outputStream.close();
        if (dataInputStream != null) dataInputStream.close();
        if (inputStream != null) inputStream.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
ackmessage = t;
System.out.println("Receiver Physical Layer로부터 " + ackmessage + "받음");
String tt = ackmessage;
String mlt2[] = tt.split("");
String after2[] = new String[tt.length()];
String eql = " ";

for(int i =0; i<mlt2.length; i++){
    if(i == 0) {
        if(mlt2[i].equals("0")) {
            after2[i] = "0";
            eql = mlt2[i];
        }
        else if(mlt2[i].equals("+")
            after2[i] = "1";
            eql = mlt2[i];
        }
    }
                }
            }
            else {
                if(mlt2[i].equals(eql)) {
                    after2[i] = "0";
                    eql = mlt2[i];
                }
                else {
                    after2[i] = "1";
                    eql = mlt2[i];
                }
            }
        }
        System.out.print("MLT -> bit stream: ");
        for(int i = 0; i<after2.length; i++) {
            System.out.print(after2[i]);
        }

        System.out.println();
        String unstuff = String.join("", after2);
        System.out.println("Data Link로 " + unstuff + "전송");
        try {
            CmnVar.queue.put(unstuff);
        } catch (InterruptedException e1) {
            // TODO Auto-generated catch block
            e1.printStackTrace();
        }
        System.out.println("----------
    }
```

<DataLinkLayer>
Physical Layer의 run void가 할일
이 끝나고 join()으로 기다리고 있던
DatalinkLayer가 시작됩니다.
queue에다 넣은 값을 가져오고
bitunstuffing을 한후
Simpleprotocol 뒤
Queue에 넣어준 뒤
Datalinklayer의 할일이 끝납니다.

```
    try {
        t4.join();
    } catch (InterruptedException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    try {
        p = CmnVar.queue.take();
    } catch (InterruptedException e1) {
        // TODO Auto-generated catch block
        e1.printStackTrace();
    }
    //String bitw2 = PhysicalLayer.getackmessage();
    System.out.println("DataLinkLayer -> " + p + " Physical로 부터 전송받음" );
    String bitunsuffer = p.replace("111110", "11111");
    System.out.println("unstuffing: " + bitunsuffer);
    System.out.println("simple protocol로 " + bitunsuffer + "보냄");
    try {
        CmnVar.queue.put(bitunsuffer);
    } catch (InterruptedException e1) {
        // TODO Auto-generated catch block
        e1.printStackTrace();
    }
    System.out.println("----------------------------------------------------");
```

<NetworkLayer>
Join()으로 기다리고있던 networklayer
가 실행됩니다. Queue에 있는 값을 빼서
잘 왔는지 확인하고 다시 queue에
넣어줍니다.
NetworkLayer가 할일이 끝났습니다.

```
    DataLinkLayer t3 = new DataLinkLayer("");
    t3.start();
    try {
        t3.join();
    } catch (InterruptedException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    try {
        t = CmnVar.queue.take();
    } catch (InterruptedException e1) {
        // TODO Auto-generated catch block
        e1.printStackTrace();
    }
    System.out.println("NetworkLayer -> " + t + "전송");
    System.out.println("----------------------------------------------------");
    try {
        CmnVar.queue.put(t);
    } catch (InterruptedException e1) {
        // TODO Auto-generated catch block
        e1.printStackTrace();
    }

}
```

<TransportLayer>

Join()으로 기다리고 있던
Tranportlayer가 실행됩니다.
queue에서 값을 꺼내
아스키코드 2진수를 7비트씩 자른 뒤
10진수로 바꿔줍니다.
바꿔준 10진수를 다시 아스키코드로
바꿔줍니다. 각각 바꾼 아스키코드를
합쳐서 ACK 가 된다면
수신 확인 메시지를 출력하고
수신 확인 메시지를 큐에 담습니다.
Transport의 할일은 끝났습니다.

```
        t33.join();
    } catch (InterruptedException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    try {
        t = CmnVar.queue.take();
    } catch (InterruptedException e1) {
        // TODO Auto-generated catch block
        e1.printStackTrace();
    }
    String check = t;
    System.out.println("TransportLayer -> " );
    String a = check.substring(0,7);
    String b = check.substring(7,14);
    String c = check.substring(14,21);
    int binaryToDecimala = Integer.parseInt(a, 2);
    int binaryToDecimalb = Integer.parseInt(b, 2);
    int binaryToDecimalc = Integer.parseInt(c, 2);
    char g = ((char)binaryToDecimala);
    char f = ((char)binaryToDecimalb);
    char t = ((char)binaryToDecimalc);
    String gg = g + "";
    String ff = f + "";
    String tt = t + "";
    String R = gg + ff + tt;
    System.out.println(a + "->" + binaryToDecimala + "-> " + gg );
    System.out.println(b + "->" + binaryToDecimalb + "-> " + ff );
    System.out.println(c + "->" + binaryToDecimalc + "-> " + tt );
    System.out.println(R);
    if ( R.equals("ACK")) {
        System.out.println("수신 확인");
    }
    try {
        CmnVar.queue.put("수신확인");
    } catch (InterruptedException e1) {
        // TODO Auto-generated catch block
        e1.printStackTrace();
    }
```

<Applicaion Layer >
join()으로 기다리고 있던
applicaionlayer가 실행됩니다.
queue에서 수신확인 메시지를 꺼내
출력합니다

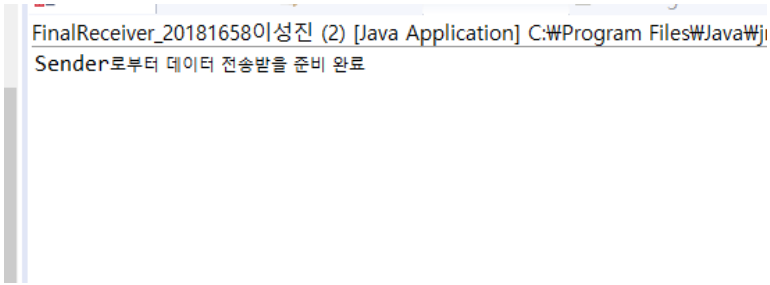
```
class ApplicationLyaer extends Thread {
    private String t;

    public ApplicationLyaer(String t) {
        this.t = t;
    }

    public void run() {
        /*
        Scanner s = new Scanner(System.in);
        System.out.println("전송하고 싶은 8자리 숫자 입력: ");
        String message = s.nextLine();
        */
        String message = "11111111";
        System.out.println("ApplicationLayer -> " + message + " 전송");
        System.out.println("-------------------------------------------------");
        try {
            CmnVar.queue.put(message);
        } catch (InterruptedException e1) {
            // TODO Auto-generated catch block
            e1.printStackTrace();
        }
        TransportLayer t2 = new TransportLayer("");
        t2.start();
        try {
            t2.join();
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        try {
            t  = CmnVar.queue.take();
        } catch (InterruptedException e1) {
            // TODO Auto-generated catch block
            e1.printStackTrace();
        }
        System.out.println("ApplicaionLayer ->" + t + " 메시지를 받았다. 잘 도착했다고 한다 ! ! !  끗 ");
```

<실행 화면 Sender, Receiver>

① FinalReceiver_20181658이성진.java를 먼저 실행시켜줍니다



② FinalSender_20181658이성진.java를 실행시켜줍니다.

<Sender>

```
ApplicationLayer -> 11111111 전송
-----------------------------------------------------
TransportLayer -> 11111111 전송 받음
TransportLayer -> 11111111 전송
stop.. wait!
-----------------------------------------------------
NetworkLayer -> 11111111 전송 받음
NetworkLayer -> 11111111 전송
-----------------------------------------------------
DataLinkLayer -> 11111111 전송 받음
DataLinkLayer -> bitstuffing: 111110111
Simple protocol로 Mac Layer에게로 전송
MacLayer -> csma/cd start:
persistence is idel
Transmission done?
NO !!!! Transmit and receive
Transmission done?
NO !!!! Transmit and receive
Transmission done?
Yes !!! done !!
Not Collision
Success!!
MacLayer -> csma/cd end:
PhysicalLayer로111110111보냄
-----------------------------------------------------
PhysicalLayer -> 111110111 전송 받음
MLT 처리 결과: +0-0++0-0
+0-0++0-0receiver PhysicalLayer로 전송
Receiver Physical Layer로부터 ++++++0-----0+000--0+받음
MLT -> bit stream: 10000110000111001011
Data Link로 100000110000111001011전송
-----------------------------------------------------
DataLinkLayer -> 10000110000111001011 Physical로 부터 전송받음
unstuffing: 10000110000111001011
simple protocol로 100000110000111001011보냄
-----------------------------------------------------
NetworkLayer -> 100000110000111001011전송
-----------------------------------------------------
TransportLayer -> 100000110000111001011전송 받음
1000001->65-> A
1000011->67-> C
1001011->75-> K
ACK
수신 확인
ApplicationLayer에게 수신확인 보냄
-----------------------------------------------------
ApplicaionLayer ->수신확인 메시지를 받았다. 잘 도착했다고 한다 ! ! !  끗
```

<Receiver>

```
Sender로부터 데이터 전송받을 준비 완료
Sender 연결 완료
socket : Socket[addr=/127.0.0.1,port=50456,localport=8000]
-----------------------------------------------------
Receiver의 PhysicalLayer
sender의 PhysicalLayer로부터 +0-0++0-0 도착
MLT -> bit stream: 111110111
Data Link로 111110111전송
-----------------------------------------------------
DataLinkLayer -> 111110111 전송 받음
unstuffing: 11111111
NetworkLayer로 11111111전송
-----------------------------------------------------
NetworkLayer -> 11111111 전송 받음
NetworkLayer -> 11111111전송중
-----------------------------------------------------
Transport -> 11111111도착
ACK 생성
Application으로 11111111전송
-----------------------------------------------------
Application -> 11111111도착
11111111수신 확인
-----------------------------------------------------
TransportLayer -> ACK to ASCII
A-> 65-> 1000001
C-> 67-> 1000011
K-> 75-> 1001011
Ack to ASCII-> 100000110000111001011
NetworkLayer로 100000110000111001011전송
-----------------------------------------------------
NetworkLayer -> 10000110000111001011 지나가는중
-----------------------------------------------------
DataLinkLayer -> 100000110000111001011전송 받음
DataLinkLayer -> bitstuffing: 10000110000111001011
Simple Protocol로 MAC에게 전송
MAC Layer -> CSMA/CD Start
persistence is busy, wait, Try again
persistence is busy, wait, Try again
persistence is busy, wait, Try again
persistence is idel
Transmission done?
Yes !!! done !!
Collision
Send a jamming signal
persistence is idel
Transmission done?
NO !!!! Transmit and receive
Transmission done?
Yes !!! done !!
Collision
Send a jamming signal
persistence is busy, wait, Try again
persistence is idel
Transmission done?
Yes !!! done !!
Not Collision
Success!!
MacLayer -> csma/cd end:
PhysicalLayer로100000110000111001011전송
-----------------------------------------------------
PhysicalLayer -> 100000110000111001011전송 받음
MLT 처리 결과: ++++++0-----0+000--0+
++++++0-----0+000--0+Sender PhysicalLayer에게 전송
-----------------------------------------------------
```