

## 컴퓨터네트워크 FinalExam

소프트웨어전공 20181658 이성진

### Programming 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

```
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

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.

#### < TransportLayer >

CmnVar.queue.take() 로

Queue 넣어놔던 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 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 ! ! ! 끝");
    }
}
```

```
3
4 class TransportLayer extends Thread {
5     private String t;
6     String message;
7
8     public TransportLayer(String t) {
9         this.t = t;
10    }
11
12    public void run() {
13        try {
14            message = CmnVar.queue.take();
15            System.out.println("TransportLayer -> " + message + " 전송 받음");
16        } catch (InterruptedException e1) {
17            // TODO Auto-generated catch block
18            e1.printStackTrace();
19        }
20        System.out.println("TransportLayer -> " + message + " 전송");
21        try {
22            CmnVar.queue.put(message);
23            System.out.println("stop.. wait! ");
24        } catch (InterruptedException e1) {
25            // TODO Auto-generated catch block
26            e1.printStackTrace();
27        }
28        System.out.println("-----");
29        NetworkLayer t33 = new NetworkLayer("");
30        t33.start();
31        try {
32            t33.join();
33        } catch (InterruptedException e) {
34            // TODO Auto-generated catch block
35            e.printStackTrace();
36        }
37    }
38}
```

### < NetworkLayer >

전송 받고 다음 레이어로  
재전송합니다.

DatalinkLayer 쓰레드를

시작하고

Join을 걸어줍니다.

```
3 class NetworkLayer extends Thread {
1   private String t;
2   String message;
3
4   public NetworkLayer(String t) {
5       this.t = t;
6   }
7
8   public void run() {
9       try {
10          message = CmVar.queue.take();
11          System.out.println("NetworkLayer -> " + message + " 전송 받을 ");
12      } catch (InterruptedException e1) {
13          // TODO Auto-generated catch block
14          e1.printStackTrace();
15      }
16      System.out.println("NetworkLayer -> " + message + " 전송");
17      try {
18          CmVar.queue.put(message);
19      } catch (InterruptedException e1) {
20          // TODO Auto-generated catch block
21          e1.printStackTrace();
22      }
23      System.out.println("-----");
24      DatalinkLayer t3 = new DataLinkLayer("");
25      t3.start();
26      try {
27          t3.join();
28      } catch (InterruptedException e) {
29          // TODO Auto-generated catch block
30          e.printStackTrace();
31      }
32  }
```

### < DataLinkLayer >

코드가 길어 짧아서 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.

```
3 class DataLinkLayer extends Thread {
1   private String p;
2   String message;
3
4   public DataLinkLayer(String p) {
5       this.p = p;
6   }
7
8   public void run() {
9       try {
10          message = CmVar.queue.take();
11          System.out.println("DataLinkLayer -> " + message + " 전송 받을 ");
12      } catch (InterruptedException e1) {
13          // TODO Auto-generated catch block
14          e1.printStackTrace();
15      }
16      p = message;
17      String bitw = p;
18      String bitsuffer = bitw.replace("1111", "11110");
19      System.out.println("DataLinkLayer -> bitstuffing: " + bitsuffer);
20      System.out.println("Simple protocol로 Mac Layer에게로 전송");
21  }
```

위 코드는 DLC , 아래코드는 MAC 부분입니다.

```
System.out.println("Simple protocol로 Mac Layer에게로 전송");
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 transmission 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

Transmit and receive를 합니다.

위에서 collision 여부에 따라

증가시킨 Success와 k>Kmax에 따라

1. Abort ! ( 과제가 No free error) 이기때문

에 주석하고 k—감소시키고 다시 진행

하였습니다.

Success인 경우 PhysicalLayer로 보내기

위해 queue에 담아준 뒤

PhysicalLayer 스레드를 시작하고

Join()을 통해서 기다립니다.

(join)아래는 다시 돌아올 때 설명)

```
    }
    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();
}
}
```

### <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)

```
4
3 class PhysicalLayer extends Thread {
4     private String t;
5     private Socket socket = null;
6     private OutputStream outputStream;
7     private DataOutputStream dataOutputStream;
8     private InputStream inputStream;
9     private DataInputStream dataInputStream;
10    static String ackmessage = "";
11    String message;
12
13    public PhysicalLayer(String t) {
14        this.t = t;
15    }
16    public void run() {
17        try {
18            message = CmnVar.queue.take();
19            System.out.println("PhysicalLayer -> " + message + " 전송 받음");
20        } catch (InterruptedException e1) {
21            // TODO Auto-generated catch block
22            e1.printStackTrace();
23        }
24        //System.out.println("PhysicalLayer -> " + t + " 입력받음");
25        t = message;
26        String mlt[] = t.split("");
27        String after[] = new String[mlt.length];
28        String h = "";
29        String nonzero = "-";
30
31        nonzero = "+";
32        h = "0";
33    }
34    else {
35        after[i] = "+";
36        nonzero = "+";
37        h = "+";
38    }
39    }
40    else if( (h.equals("-")) {
41        if(mlt[i].equals("1")) {
42            after[i] = "0";
43            nonzero = "-";
44            h = "0";
45        }
46        else {
47            after[i] = "-";
48            nonzero = "-";
49            h = "-";
50        }
51    }
52    }
53    System.out.print("MLT 처리 결과: ");
54    for(int i = 0; i<after.length; i++) {
55        System.out.print(after[i]);
56    }
57    System.out.println();
58    StringBuffer result = new StringBuffer();
59    for (int i = 0; i < after.length; i++) {
60        result.append( after[i] );
61    }
62    String sendmessage = result.toString();
63}
```

```
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 = "+";
            }
            else {
                after[i] = "-";
                nonzero = "-";
                h = "-";
            }
        }
    }
}
```

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 >

```

1 import java.io.DataInputStream;
2 import java.io.DataOutputStream;
3 import java.io.InputStream;
4 import java.io.OutputStream;
5 import java.net.ServerSocket;
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
12 class FinalReceiver_20181658이성진 {
13     public static void main(String[] args) {
14         String ok = "";
15         RecPhysicalLayer t6 = new RecPhysicalLayer(ok);
16         t6.start();
17     }
18 }
19
20 class CmnVarRec {
21     public static BlockingQueue<String> queue = new ArrayBlockingQueue<String>(1);
22 }

```

Sender와 동일, ServerSocket만 추가

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.  
받은 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("Receiver@PhysicalLayer");
        System.out.println("sender@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

l와 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 들어갑니다  
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  
transport에서 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(bitsuffer);
    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.

<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)

mlt처리결과를 sendMessage2 에 넣어  
소켓으로 Receiver의 Physical Layer에서  
Sender의 Physical Layer로  
dataOutputStream.writeUTF(send  
message2) // 전송  
dataOutputStream.flush()  
// 비우기  
해서 보내준뒤 sender에게 전송  
했다고 메시지를 출력합니다.

Receiver의 할일은 끝났습니다  
그럼 다시 Sender로 가겠습니다.

```

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

try {
    t7.join();
} catch (InterruptedException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
try {
    t = CmVarRec.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(mlt2[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] = "+";
                nonzero = "-";
            }
        }
    }
}

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();
    }
}
}

```

< 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 링크 " + 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 + " Physical로부터 전송받은 ");
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

Transportlayer가 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.

### <Application Layer >

join()으로 기다리고 있던

applicationlayer가 실행됩니다.

queue에서 수신확인 메시지를 꺼내  
출력합니다

```
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();
}

class ApplicationLayer extends Thread {
    private String t;

    public ApplicationLayer(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("ApplicationLayer -> " + t + " 메시지를 받았다. 잘 도착했다고 한다 !!! 끝 ");
    }
}
```

## <실행 화면 Sender, Receiver>

- ① FinalReceiver\_20181658이성진.java를 먼저 실행시켜줍니다

```
FinalReceiver_20181658이성진 (2) [Java Application] C:\Program Files\Java\j
Sender로부터 데이터 전송받을 준비 완료
```

- 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: 100000110000111001011
Data Link로 100000110000111001011전송
-----
DataLinkLayer -> 100000110000111001011 Physical로 부터 전송받음
unstuffing: 100000110000111001011
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 ! ! !  끝
```

### <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 -> 100000110000111001011 지나가는중
-----
DataLinkLayer -> 100000110000111001011전송 받음
DataLinkLayer -> bitstuffing: 100000110000111001011
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
-----
```