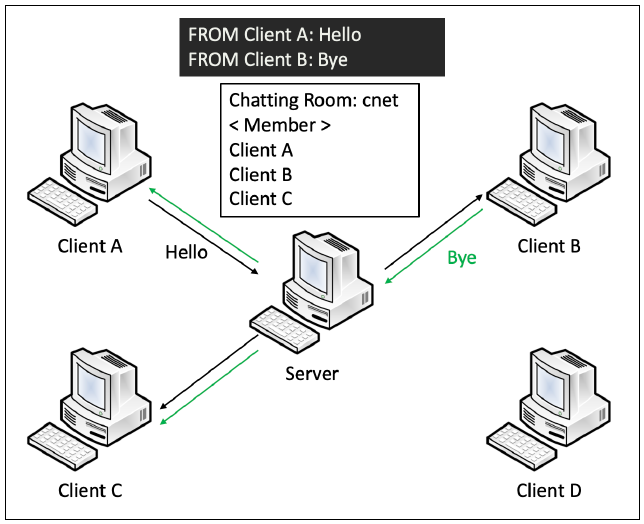
**Assignment1: TCP-based Client and Server Open Chatting Program**

2016025078 강덕영

**-개요**

Java에서 제공하는 TCP socket API를 사용하여 Client-Server 구조의 오픈 채팅 프로그램을 구현하였습니다.



위 그림과 같이 클라이언트-서버 방식의 채팅 프로그램을 구현하였습니다. 서버는 클라이언트의 요청에 따라 채팅방을 만들고, 채팅방에 참가한 클라이언트의 채팅을 중계 전달하는 역할을 합니다. 서버는 로컬 호스트 IP(127.0.0.1)를 사용하며, 서버와 채팅을 주고 받기 위한 특정 포트를 사용합니다. 클라이언트는 #CREATE <채팅방 이름> <사용자 이름> 명령어를 사용하여 채팅방을 개설하여 채팅방에 참여하거나, #JOIN <채팅방 이름> <사용자 이름> 명령어를 통해 기존에 존재하는 채팅방에 참여할 수 있습니다. 물론, 동일한 이름의 채팅방이 존재할 경우 #CREATE 명령어는 동작하지 않으며, 특정 이름의 채팅방이 존재하지 않을 경우 #JOIN 명령어는 동작하지 않습니다. 채팅방에 참가한 후 채팅을 입력하면 동일한 채팅방에 있는 모든 사용자에게 채팅이 전달됩니다(발화자 제외). 이때, 과제 1에서 구현한 것과 동일하게, 채팅을 입력하는 것과 다른 사용자의 채팅을 받는 것을 동시에 할 수 있도록 스레드를 이용하여 구현하였습니다. #STATUS 명령어를 입력하면 현재 채팅방의 이름과, 현재 이 채팅방에 참여하고 있는 모든 사용자의 이름을 출력해줍니다. #EXIT 명령어를 입력하면 현재 채팅방에서 나올 수 있으며, 채팅방에 입장하지 않은 상태에서 #TERMINATE 명령어를 사용하면 프로그램이 종료됩니다. 마지막으로 서버는 여러개의 채팅방을 유지할 수 있습니다. (파일 송수신은 구현하지 않았습니다.)

**-구현 세부사항**

사용 언어는 Java이며, 이 프로그램은 Server.java와 Client.java 파일을 컴파일 및 실행함으로써 구동됩니다.

**1. Client.java**

서버 프로그램이 동작하고 있을 때, java Client 127.0.0.1 2020 2021 명령어를 cmd에 입력하면 클라이언트 프로그램이 실행되며, 서버 소켓과 연결됩니다. 클라이언트는 다음 명령어들을 수행할 수 있습니다.

- #CREATE <chatting room name> <user name>

- #JOIN <chatting rorom name> <user name>

#CREATE, #JOIN 명령어를 수행할 때, 서버로 메시지를 보내고 또 서버로부터 메시지를 받기 위하여 BufferedReader, PrintWriter 객체를 생성하여 진행합니다. #CREATE, #JOIN 명령어가 성공적으로 수행되면, 클라이언트의 화면에 각각 "#CREATE success", "#JOIN success"가 출력됩니다. 그렇지 않으면 "#CREATE fail", "#JOIN fail"이 출력됩니다.

클라이언트가 채팅방에 참여하면, 메시지 송/수신을 동시에 하기 위해 스레드 두 개를 생성 후 실행합니다. 메시지 송신을 위해 MessageSenderTCPThread 클래스를, 메시지 수신을 위해 MessageReceiverTCPThread 클래스를 사용합니다. 두 스레드를 각각 start 함수를 이용해 실행시킴으로써 메시지 송/수신을 동시에 하며, join 함수를 이용해 클라이언트의 채팅이 끝날 때까지 기다리게 됩니다. 클라이언트는 #EXIT 명령어를 통해 채팅방에서 나올 수 있습니다. 채팅방에서 나온 후 #CREATE, #JOIN 명령어를 사용해 다시 채팅방에 참가하여 채팅을 할 수 있습니다.

- #TERMINATE

채팅방에서 나온 클라이언트는 #TERMINATE 명령어를 통해 프로그램을 완전 종료할 수 있습니다.

**2. Server.java**

클라이언트의 채팅을 중계하기 위해, 서버 프로그램은 클라이언트 프로그램보다 먼저 실행되어야 합니다. java Server 2020 2021 명령어를 cmd에 입력함으로써 서버 프로그램을 실행할 수 있습니다. 서버는 다수의 클라이언트의 채팅방 요청을 처리하기 위해, 클라이언트의 요청을 받았다면 해당 클라이언트를 위한 스레드를 생성 후 실행시킵니다. 서버는 ctrl+c 명령어를 받으면 종료됩니다. 서버는 현재 채팅방 정보(채팅방 이름, 채팅방에 포함된 유저 이름)를 유지하기 위해, 클라이언트를 위한 스레드 클래스에 Map 자료구조를 static 변수로 선언합니다. 구체적으론, '채팅방 이름' -> '이 채팅방에 참여한 클라이언트 이름'을 위한 Map과, '채팅방 이름' -> '이 채팅방에 참여한 클라이언트의 소켓'을 위한 Map이 존재합니다. 서버는 클라이언트로부터 명령어(#CREATE, #JOIN, #EXIT, #STATUS, #TERMINATE)를 받으면 적절히 처리하고 response 메시지를 보냅니다. 그리고 채팅을 받으면 동일 채팅방에 존재하는 모든 소켓에게 채팅을 보냄으로써 채팅을 중계합니다(발화자 제외). 만약 #TERMINATE 명령어를 받으면 소켓을 닫고 스레드를 종료합니다.

**-프로그램 실행 방법**

운영체제: Windows 10

Assignment2\_2016025078\_강덕영.zip 파일을 압축 해제(ex, ComputerNetwork\_TCP 이름으로) 후 cmd를 통해 ComputerNetwork\_TCP\src 경로로 이동 후 진행하면 됩니다.

컴파일: javac Server.java

javac Client.java

프로그램 실행:

1. 서버 프로그램 실행: java Server 2020 2021

우선 서버 프로그램을 실행시킵니다.

2. 클라이언트 프로그램 실행: java Client 127.0.0.1 2020 2021

클라이언트 수만큼 cmd를 띄우고 이 명령어를 실행합니다.

클라이언트 프로그램 실행 후 명령어:

1. #CREATE <채팅방 이름> <사용자 이름>: 입력한 이름으로 채팅방 생성

2. #JOIN <채팅방 이름> <사용자 이름>: 입력한 이름으로 특정 채팅방에 들어가기

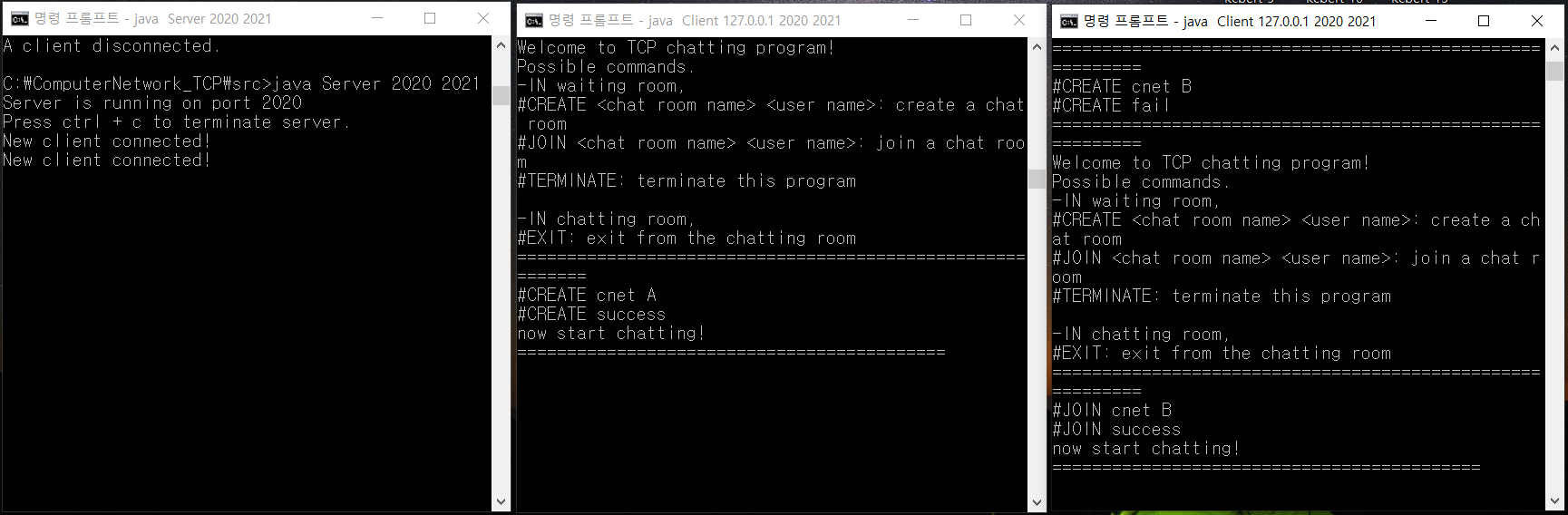
3. #EXIT: 채팅방에서 나오기

4. #TERMINATE: 프로그램 종료하기

참고로 #TERMINATE는 클라이언트 프로그램을 실행한 직후 혹은 채팅방에서 나왔을 때 사용 가능한 명령어입니다.

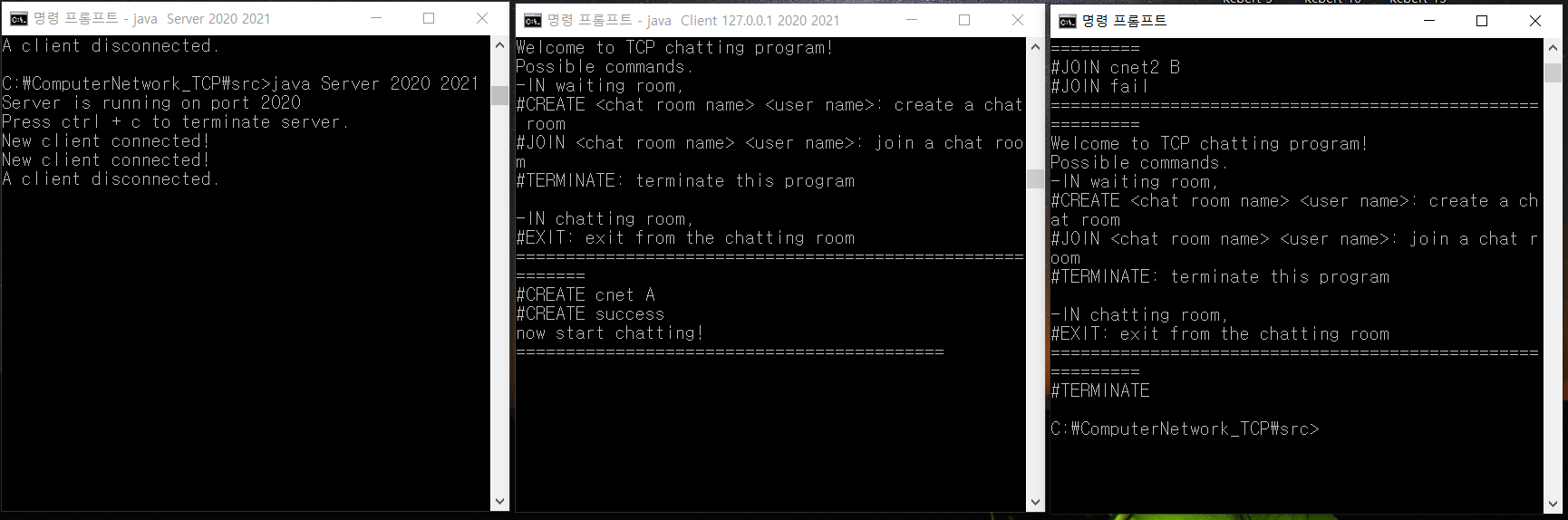
-**프로그램 실행 캡쳐**

1. 채팅방 생성 및 참여1



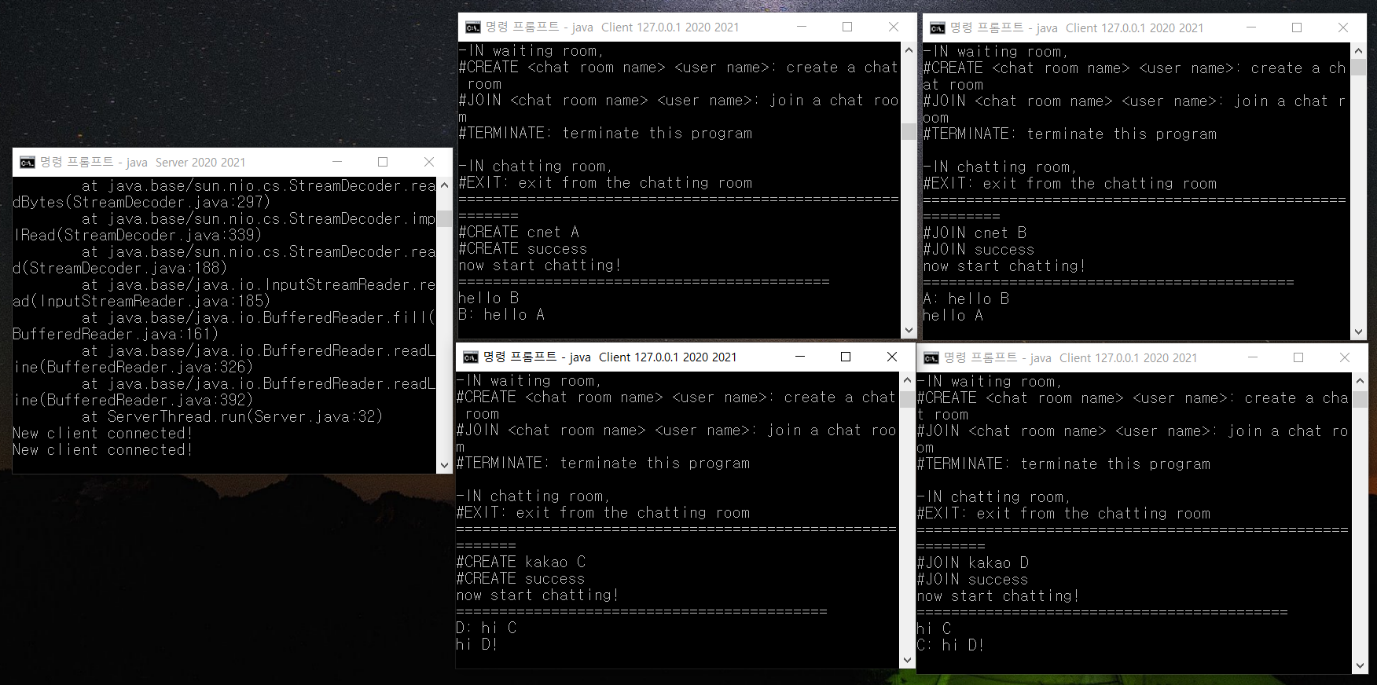
위 그림은 왼쪽부터 각각 서버, 클라이언트1, 클라이언트2의 프로그램을 실행시킨 것입니다. 우선 Client1이 'A'라는 이름으로 채팅방 'cnet'을 생성하고 성공 메시지(#CREATE success)를 받았습니다. 이후에 Client2는 'B'라는 이름으로 채팅방 'cnet'을 생성하려 했지만, 이미 동일 이름의 채팅방이 존재하므로 실패 메시지(#CREATE fail)를 받은 모습입니다. 이후에 Client2는 'B'라는 이름으로 채팅방 'cnet'에 참여 요청을 보냈고, 'cnet' 채팅방이 존재하므로 성공 메시지(#JOIN success)를 받은 모습입니다. 이제 A와 B는 서로 채팅을 할 수 있습니다.

2. 채팅방 생성 및 참여2



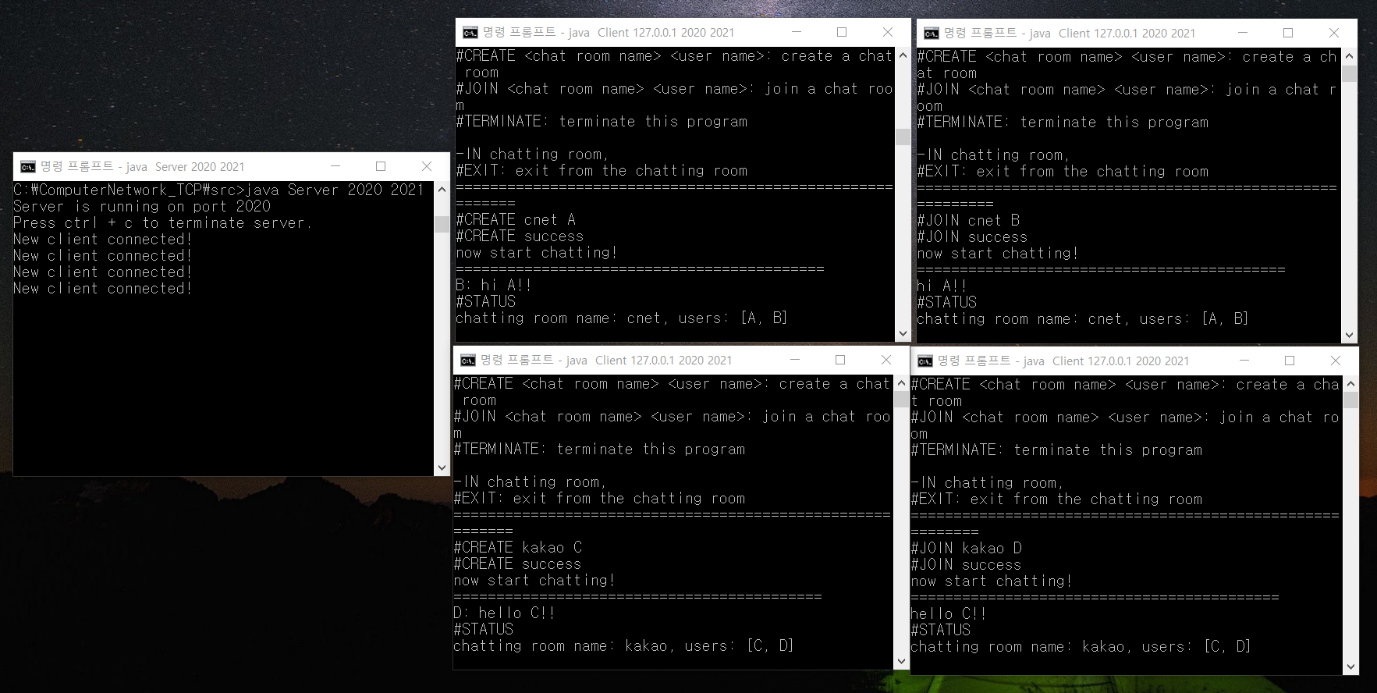
위 그림은 왼쪽부터 각각 서버, 클라이언트1, 클라이언트2의 프로그램을 실행시킨 것입니다. 우선 Client1이 'A'라는 이름으로 채팅방 'cnet'을 생성하였습니다. 이후에 Client2는 'B'라는 이름으로 채팅방 'cnet'2에 참가하려 했지만, 'cnet2' 채팅방은 존재하지 않으므로 실패 메시지(#JOIN fail)를 받은 모습입니다. 채팅방에 참가하지 못한 Client2는 #TERMINATE 명령어를 입력하여 프로그램을 종료하는 모습입니다.

3. 여러 채팅방 생성



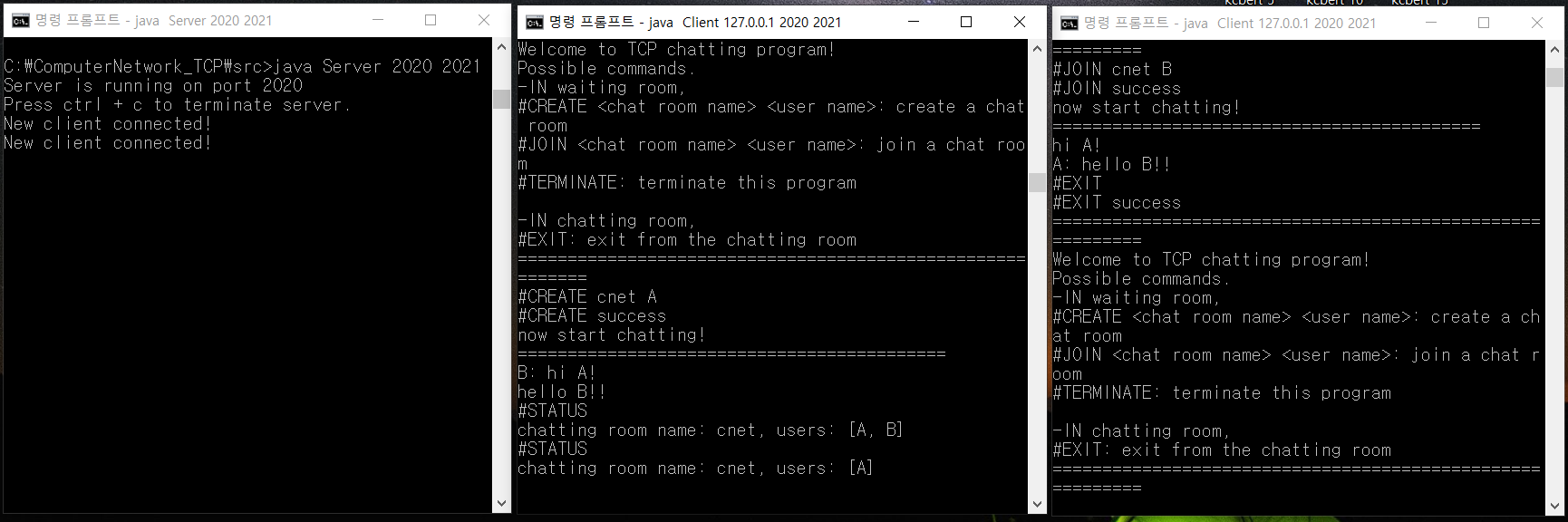
위 그림은 서버, 클라이언트1, 클라이언트2, 클라이언트3, 클라이언트4의 프로그램을 실행한 모습입니다. 가장 왼쪽이 서버, 우측 상단 두 화면이 각각 클라이언트1, 클라이언트2, 우측 하단 두 화면이 각각 클라이언트3, 클라이언트4입니다. 우선 서버 프로그램을 실행하였습니다. 클라이언트1(A)과 클라이언트2(B)는 'cnet' 채팅방에, 클라이언트3(C)과 클라이언트4(D)는 'kakao' 채팅방에 참여하여 채팅을 하고 있습니다. 서버가 여러 채팅방을 운용할 수 있음을 보여주고 있습니다.

4. #STATUS 명령어1



위 그림은 3번 그림과 동일한 환경에서, 각 채팅방에서 #STATUS 명령어를 입력한 모습입니다. 'cnet' 채팅방엔 'A'와 'B'가, 'kakao' 채팅방엔 'C'와 'D'가 잘 출력되는 것을 확인할 수 있습니다.

5. #STATUS 명령어2



위 그림은 왼쪽부터 서버, 클라이언트1, 클라이언트2의 프로그램을 실행시킨 것입니다. 클라이언트1(A), 클라이언트2(B)가 'cnet' 채팅방에 있을 때 #STATUS 명령어를 입력했습니다. 그리고 클라이언트2(B)가 #EXIT 명령어를 통해 채팅방을 나온 후 #STATUS 명령어를 입력했습니다. 두 번째 #STATUS 명령어에 대한 결과에 클라이언트2(B)가 제외된 것을 볼 수 있습니다.