[과제 task 설명] : TCP 소켓 기반 다중 사용자 채팅 시스템

'server' : client들의 연결을 관리, msg 전달 / 'client' : user의 입력을 server로 전송, server에서 전달된 msg를 실시간으로 UI에 표시 / 'Tkinter GUI' : user 입력과 표시 담당

msg 수신은 별도 thread에서 처리하여 UI 멈춤을 방지(채팅 앱이 server에서 오는 메시지를 받는 동안에도, user가 입력창에 글을 쓰거나 버튼을 누를 수 있게 만든다)

[server 연결 및 실행 과정]

client는 실행되면 먼저 TCP 소켓을 생성하고, server에 연결

이때 최초로 username을 server에 전송하여 접속 상태를 알림

```
def connect(self):
    """Connect to server and start receiving thread"""
    # TODO 1: Create TCP socket
    self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    # TODO 2: Connect to (self.host, self.port)
    self.sock.connect((self.host, self.port))

    # TODO 3: Send username (self.ui.username) to server
    self.sock.send(self.ui.username.encode('utf-8'))
```

<AF_INET, SOCK_STREAM : TCP 통신 설정 / connect() : server 연결 요청 / 첫 데이터 전송 –> username : server가 사용자 식별 가능>

[msg 송신 과정]

user가 msg를 입력하면, send_message()가 호출됨

본 함수가 입력된 문자열을 "username: 내용" 형태로 변환하고 server에 전송

server가 반환하지 않는 본인 msg는 UI에 직접 출력(내가 보낸 메시지는 서버에서 다시 보내주지 않으므로, 클라이언트가 직접 UI에 출력해줘야 함)

```
def send_message(self, event=None):
    """Send message to the server"""
    # TODO 1: Get text from UI (self.ui.get_message())
    msg = self.ui.get_message()

    if not msg:
        return

    # TODO 2: Format: f"{self.ui.username}: {msg}"
    entire_msg = f"{self.ui.username}: {msg}"

    # TODO 3: Send via socket
    try:
        self.sock.send(entire_msg.encode('utf-8'))
    except Exception:
        self.ui.display_message("Fail to send msg.")
        return

    # TODO 4: Also display locally using self.ui.display_message()
    self.ui.display_message(entire_msg)
```

<self.ui.get_message() : 입력란 값 읽기 / f"{username}: {msg}" : user 표시 형식유지 / display_message() : server echo 없이 UI에 즉시표시>

[msg 수신 과정 : thread 처리]

Tkinter의 mainloop()와 socket.recv()는 둘다 blocking 함수 -> 하나의 thread에서 동시에 실행 불가

따라서 msg 수신은 별도의 thread에서 반복적으로 처리

```
# TODO 4: Start background receiving thread (self.receive_messages)
recv_thread = threading.Thread(target=self.receive_messages, daemon=True)
recv_thread.start()
```

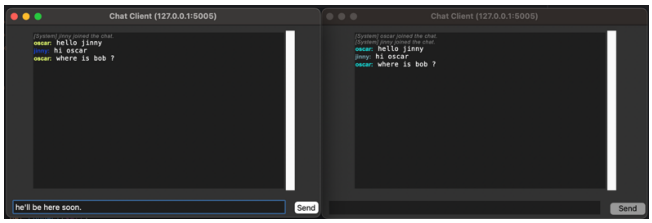
수신 thread는 다음 코드로 지속적으로 server 메세지를 기다림 :

```
while True:
    try:
        data = self.sock.recv(1024)
        if not data:
            self.ui.display_message("Server disconnected.")
            break
        msg = data.decode('utf-8')
        self.ui.display_message(msg)
    except Exception:
        break
```

<UI는 정상적으로 입력, 출력 처리 / server 메세지는 지연 없이 표시 / user가 typing 중이라도 채팅 내용이 실시간 갱신>

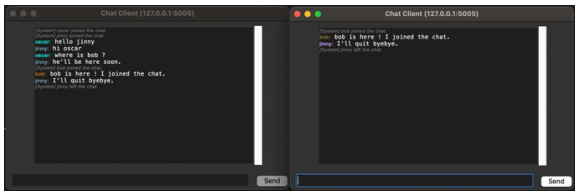
정리 : 소켓 통신과 thread 기반 병렬 처리를 활용하여 실시간 GUI 채팅 기능을 구현 / msg 수신을 별도 thread에서 처리함으로써 GUI가 멈추지 않고 user 입력과 server msg 표시가 동시에 가능하도록 설계

[동작 스크린샷] (username 입력 – 채팅방 참여 - 새로운 user 참여 – user입력으로 채팅 – user추가 참여 – 퇴장 : 순서대로 동작 캡처함)

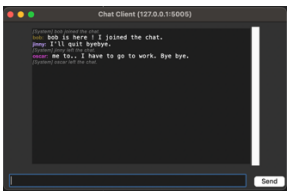


: oscar & jinny is chatting

: bob joined



: jinny quit



: oscar quit