목차

5강
TCP 소켓 프로그래밍 IV

Chatting(서버/클라이언트)

0. 시현

1. 요구사항

1. 단체 채팅방 (1개의 방에서 다수의 사용자가 채팅을 한다.)

2. 닉네임을 등록해야 한다.

3. 다른 사용자가 입장하면 “OOO님이 입장 하였습니다” 메시지가 출력

4. 다른 사용자의 메시지는 키보드로 입력도중에 전달되어 화면에 출력된다.

5. 메시지의 전송은 엔터를 쳤을 때 전송되고 방안의 모든 사용자에게 전달된다.

6. 방을 나올 때는 “quit” 또는 프로그램 종료로 할 수 있다.

7. 방을 나가면 다른 사용자에게 “OOO님이 퇴장 하였습니다” 메시지가 출력된다.

2. Chatting 서버

❑ 서버 기능 정의

1. 서버는 여러 클라이언트가 접속할 수 있어야 한다. **(다중 처리 가능, 멀티스레드 프로그래밍)**

2. 서버는 여러 클라이언트에게 동시에 메시지를 보낼 수 있는 브로드캐스팅**(broadcasting)** 기능이 있어야 한다.

3. EchoServer의 각 스레드는 자신의 **IO Stream** 객체만 사용하면 되었지만 , **Chat Server**에서는 다른 스레드의 **IO Stream**을 사용해야 한다. **(printWriter** 객체 **)**

4. 닉네임을 등록하기 위한 요청, 메시지를 전달하기 위한 요청, 방을 나가기 위한 요청 등 클라이언트의 요청이을 구별하기 위해 프로토콜(채팅 프로토콜)을 설계해야 한다.

예)
JOIN:홍길동**\r\n**
MESSAGE:방가 **^^;\r\n**
QUIT\r\n

2. Chatting 서버

❑ MainThread

```
// 1. 서버 소겟 생성
serverSocket = new ServerSocket();

// 2. 바인딩
String hostAddress = InetAddress.getLocalHost().getHostAddress();
serverSocket.bind( new InetSocketAddress( hostAddress, PORT ) );
log( "연결 기다림 " + hostAddress + ":" + PORT );

// 3. 요청 대기
while( true ) {
   Socket socket = serverSocket.accept();
   new ChatServerTread( socket ).start();
}
```

- **Main Thread**의 주요코드
- 클라이언트로 부터 연결 요청을 기다린다.
- 클라이언트와 연결된 후, 클라이언트와 채팅 데이터 통신은 **ChatServerTread**가 한다.

2. Chatting 서버

❑ ChatServerThread

1. 스레드의 인스턴스 변수

- 통신을 위한 스트림을 얻어 오기 위해 **Socket** 객체를 저장해야 한다.
- 연결된 클라이언트의 닉네임을 저장하고 있어야 한다.

```
public class ChatServerTread extends Thread {

	private String nickname;
	private Socket socket;

	public ChatServerTread( Socket socket ) {
		this.socket = socket;
	}
}
```

2. Chatting 서버

❑ ChatServerThread

2. 요청 처리를 위한 Loop 작성

- **run** 메소드 오버라이딩
- **main thread**로 부터 전달받은 **socket**를 통해 **IO Stream**을 받아오는데 문자 단위 처리와 라인 단위 읽기를 위해 보조 스트림 객체를 생성해서 사용한다.

```
//1. Remote Host Information

//2. 스트림 얻기
bufferedReader =
     new BufferedReader( new InputStreamReader( socket.getInputStream(), StandardCharsets.UTF_8 ) );
printWriter =
     new PrintWriter( new OutputStreamWriter( socket.getOutputStream(), StandardCharsets.UTF_8 ), true );


//3. 요청 처리
while( true ) {
   String request = bufferedReader.readLine();
   if( request == null ) {
      log( "클라이언트로 부터 연결 끊김" );
      break;
   }

   // 4. 프로토콜 분석

}
```

2. Chatting 서버

❑ ChatServerThread

3. 프로토콜 분석

```
String[] tokens = request.split( ":" );

if( "join".equals( tokens[0] ) ) {

   doJoin( tokens[1], printWriter );

} else if( "message".equals( tokens[0] ) ) {

   doMessage( tokens[1] );

} else if( "quit".equals( tokens[0] ) ) {

   doQuit();

} else {

   ChatServer.log( "에러:알수 없는 요청(" + tokes[0] + ")" );
}
```

- chat 프로토콜 형식

요청명령:파라미터1:파라미터2: … \r\n

- 각 요청을 구분하는 경계가 되는 것은 \r\n 이다.
- 요청은 “:” 기준으로 요청명령과 파라미터로 분리한다.
- 각 각의 요청명령을 처리하는 메서드를 구현하고 호출한다.

2. Chatting 서버

❑ ChatServerThread

3-1. join 프로토콜 구현

```
private void doJoin( String nickName, Writer writer ) {
   this.nickname = nickName;
   /* writer pool에  저장 */
}
```

- 프로토콜

  “join:nickname\r\n”

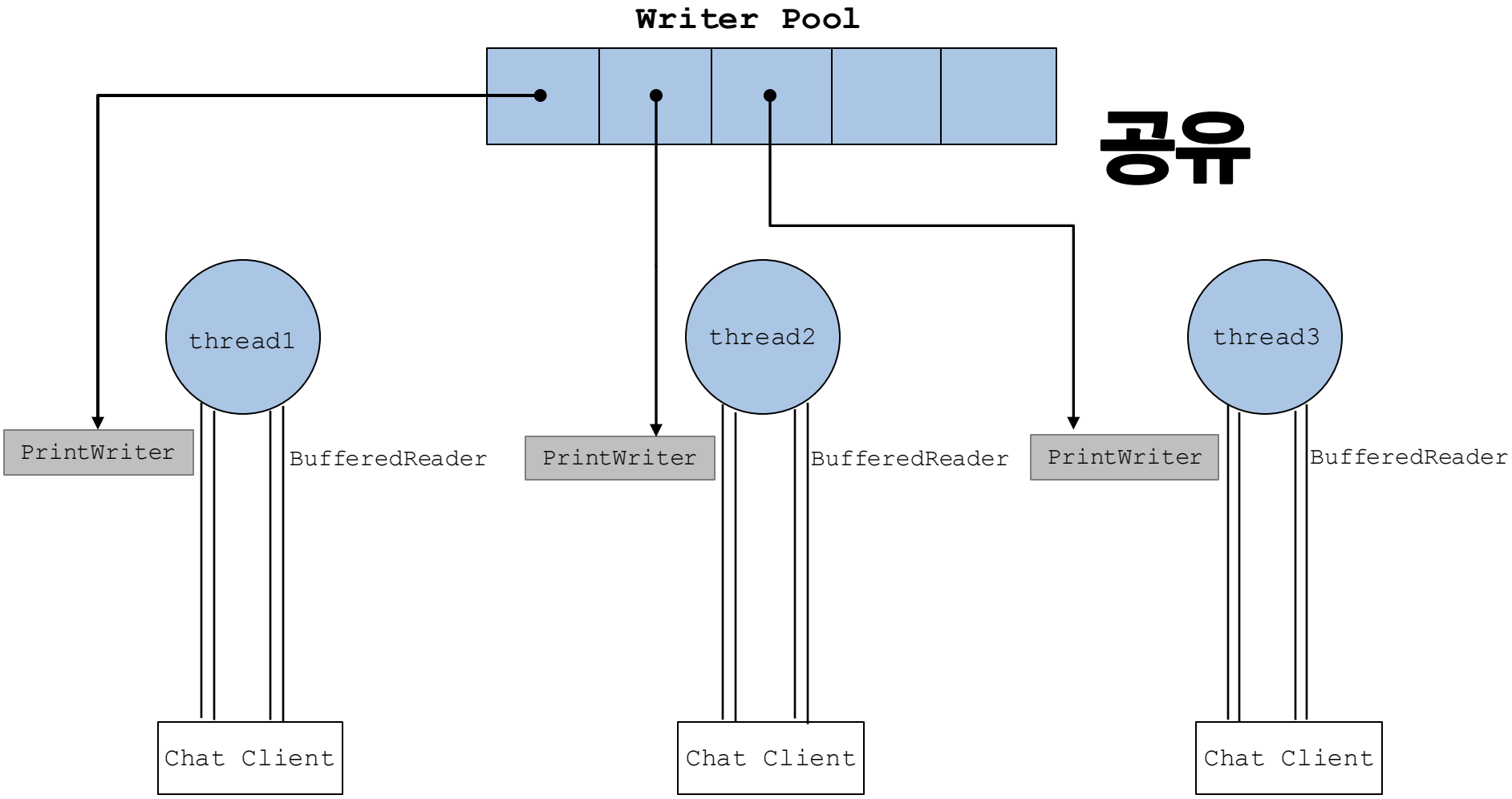
- 1번째 파라미터 nickname을 thread 객체 변수로 저장한다.

- writer pool에 현재 스레드의 writer 인 printWriter를 저장해야 한다.

2. Chatting 서버

❑ ChatServerThread

3-1. join 프로토콜 구현



2. Chatting 서버

❑ ChatServerThread

3-1. join 프로토콜 구현

- main thread에서 PrintWriter를 담을 수 있는 List를 생성한다.

```
List<Writer> listWriters = new ArrayList<Writer>();
```

- 데이터 통신 스레드들에서 이 List를 공유 해야 하기 때문에 스레드에 List 객체를 참조하는 변수를 추가한다.

```
List<Writer> listWriters;
```

- 요청이 수락하고 스레드를 생성할 때, List객체를 스레드의 생성자를 통해 전달한다.

```
new ChatServerTread( socket, listWriters ).start();
```

```
public ChatServerTread( Socket socket, List<Writer> listWriters ) {
    this.socket = socket;
    this.listWriters = listWriters;
}
```

2. Chatting 서버

❑ ChatServerThread

3-1. join 프로토콜 구현

```
private void doJoin( String nickName, Writer writer ) {
   this.nickname = nickName;

   /* writer pool에  저장 */
   addWriter( writer );
}

private void addWriter( Writer writer ) {
   synchronized( listWriters ) {
      listWriters.add( writer );
   }
}
```

- addWriter 메소드의 구현 예시
- List인 Writer Pool 에 파라미터로 받은 Writer를 추가한다.
- synchronized 키워드는 여러 스레드가 하나의 공유 객체에 접근할 때, 동기화를 보장 해준다.

2. Chatting 서버

❑ ChatServerThread

3-1. join 프로토콜 구현

```
private void broadcast( String data ) {

   synchronized( listWriters ) {

      for( Writer writer : listWriters ) {
           PrintWriter printWriter = (PrintWriter)writer;
           printWriter.println( data );
           printWriter.flush();
      }

   }

}
```

- 서버에 연결된 모든 클라이언트에 메시지를 보내는(브로드캐스트) 메소드
- 스레드간 공유 객체인 **listWriters** 에 접근 하기 때문에 동기화 처리를 해 주어야 한다.
- **PrintWriter**의 메서드를 사용해야 하기 때문에 다운 캐스팅을 명시적으로 해주었다.

2. Chatting 서버

❑ ChatServerThread

3-1. join 프로토콜 구현

```
private void doJoin( String nickName, Writer writer ) {
   this.nickname = nickName;

   String data = nickName + "님이 참여하였습니다.";
   broadcast( data );

   /* writer pool에  저장 */
   addWriter( writer );

   // ack
   printWriter.println( "join:ok" );
   printWriter.flush();
}
```

- **doJoin**은 한 사용자가 채팅 방에 참여 했을 때, 다른 사용자들에게 **"OOO**님이 입장하셨습니다.**"** 라는 메세지를 브로드캐스팅해야 한다.

- **ack**를 보내 방 참여가 성공했다는 것을 클라이언트에게 알려 줘야 한다.

2. Chatting 서버

❑ ChatServerThread

3-2. message 프로토콜 구현

```
private void doMessage( String message ) {

	/* 잘 구현 해 보기 */

}
```

- 프로토콜

“message:하이 ^^;\r\n”

2. Chatting 서버

❑ ChatServerThread

3-3. quit 프로토콜 구현

```
private void doQuit(  Writer writer ) {
   removeWriter( writer );

   String data = nickname + "님이 퇴장 하였습니다.";
   broadcast( data );
}

private void removeWriter( Writer writer ) {

   /* 잘 구현 해보기 */
}
```

- 프로토콜

  “quit”

- “OOO님이 퇴장 하였습니다” 메시지가 브로드캐스팅 되어야 한다. 현재 스레드의 writer를 Writer Pool에서 제거한 후, 브로드캐스팅 한다.

2. Chatting 서버

❑ ChatServerThread

3-4. 에러 처리 (클라이언트가 “quit” 보내지 않고 소켓을 닫은 경우)

```
if( request == null ) {
    ChatServer.log( "클라이언트로 부터 연결 끊김" );
    doQuit( printWriter );
    break;
}
```

3. Chatting 클라이언트

❑ 요구사항

1. 키보드로 입력이 가능

2. 입력 중에 메시지를 수신 할 수 있다**.**

3. 즉**,** 키보드 입력을 받는 작업은 **main thread**에서

5. 데이터 수신과 프로토콜 처리 작업은 데이터 수신 **Thread**에서 처리하도록 작성한다**.**

6. 다음 구현 예시를 참고해서 구현한다**.**

3. Chatting 클라이언트

❑ 구현 예시 - main thread

```
Scanner scanner = null;
Socket socket = null;
try {
   //1. 키보드 연결

   //2. socket 생성

   //3. 연결

   //4. reader/writer 생성

   //5. join 프로토콜
   System.out.print("닉네임>>" );
   String nickname = scanner.nextLine();
   printWriter.println( "join:" + nickname );
   printWriter.flush();

   //6. ChatClientReceiveThread 시작

   //7. 키보드 입력 처리
   while( true ) {
      System.out.print( ">>" );
      String input = scanner.nextLine();

      if( "quit".equals( input ) == true ) {
          // 8. quit 프로토콜 처리
          break;
      } else {
          // 9. 메시지 처리
      }
   }

} catch( IOException ex ) {
       log( "error:" + ex );
} finally {
      //10. 자원정리
}
```

3. Chatting 클라이언트

❑ 구현 예시 - ChatClientReceiveThread

```
private BufferedReader bufferedReader;

@Override
public void run() {

    /* reader를 통해 읽은 데이터 콘솔에 출력하기 (message 처리) */

}
```