1. ChatClient 소스 line별 분석하여 각 line 뒤에 주석문으로 설명할 것

```
import java.net.*;
import java.io.*;
public class ChatClient {
	public static void main(String[] args) {
		// argument 길이 확인. 2개가 아니면 에러메시지 출력 후 프로그램 종료 //
		if(args.length != 2){
			System.out.println("Usage : java ChatClient <username> <server-ip>");
			System.exit(1);
		}
		Socket sock = null; // 소켓 변수 선언.
		BufferedReader br = null; // bufferedReader 선언.
		PrintWriter pw = null; // PrintWriter 선언
		boolean endflag = false; // 프로그램을 종료할지 계속할지 결정하는 flag 변수 선언
		try{
			sock = new Socket(args[1], 10001); //argument로 들어온 server-ip 주소와 포트
                                                로 새로운 소켓을 생성한다.
			pw = new PrintWriter(new OutputStreamWriter(sock.getOutputStream())); //
PrintWriter의 출력 대상을 소켓으로 지정
			br = new BufferedReader(new
InputStreamReader(sock.getInputStream())); // BufferedReader의 입력 대상을 소켓으로 지정
			BufferedReader keyboard = new BufferedReader(new
InputStreamReader(System.in)); // 사용자의 입력을 기억하는 keyboard 변수 선언
			// 사용자가 argument로 전달한 유저이름을 소켓에 입력한다 //
			pw.println(args[0]);
			pw.flush();
			InputThread it = new InputThread(sock, br); // 소켓으로 새로운 쓰레드 생성
			it.start(); // 쓰레드 시작. run()메소드가 호출된다.
			String line = null;
			while((line = keyboard.readLine()) != null){
				pw.println(line); // 사용자가 입력한 정보를 line변수에 저장하고 소켓에 입력
한다.
				pw.flush();
				// /quit이라는 명령어가 입력되면 endflag을 true로 만들어주고 반복문 종료
>> 프로그램 종료 시그널 //
				if(line.equals("/quit")){
					endflag = true;
					break;
				}
			}
			System.out.println("Connection closed.");
		}catch(Exception ex){
			if(!endflag) // 프로그램이 비정상적으로 종료됐을 경우 어떤 에러인지 출력
				System.out.println(ex);
		}finally{
			// 에러의 여부와 관계없이 무조건 실행. 소켓과 입출력 스트림을 모두 닫아준다 //
			try{
				if(pw != null)
					pw.close();
			}catch(Exception ex){}
			try{
				if(br != null)
					br.close();
			}catch(Exception ex){}
			try{
				if(sock != null)
```

```
                                        sock.close();
                                }catch(Exception ex){}
                        } // finally
                } // main
} // class

class InputThread extends Thread{
        private Socket sock = null; // 소켓 변수 선언
        private BufferedReader br = null; // 입력스트림 선언
        public InputThread(Socket sock, BufferedReader br){
                this.sock = sock; // 인자로 전달된 소켓과 입력스트림을 저장한다.
                this.br = br;
        }
        // 쓰레드 객체가 start 가 되면 이 메소드가 호출된다 //
        public void run(){
                try{
                        String line = null;
                        // 사용자가 입력한 문자열을 받아서 한줄씩 끝까지 출력한다. //
                        while((line = br.readLine()) != null){
                                System.out.println(line);
                        }
                }catch(Exception ex){
                }finally{
                        try{
                                if(br != null) // 입력스트림이 비어있으면 종료
                                        br.close();
                        }catch(Exception ex){}
                        try{
                                if(sock != null)
                                        sock.close();
                        }catch(Exception ex){}
                }
        } // InputThread
}
```

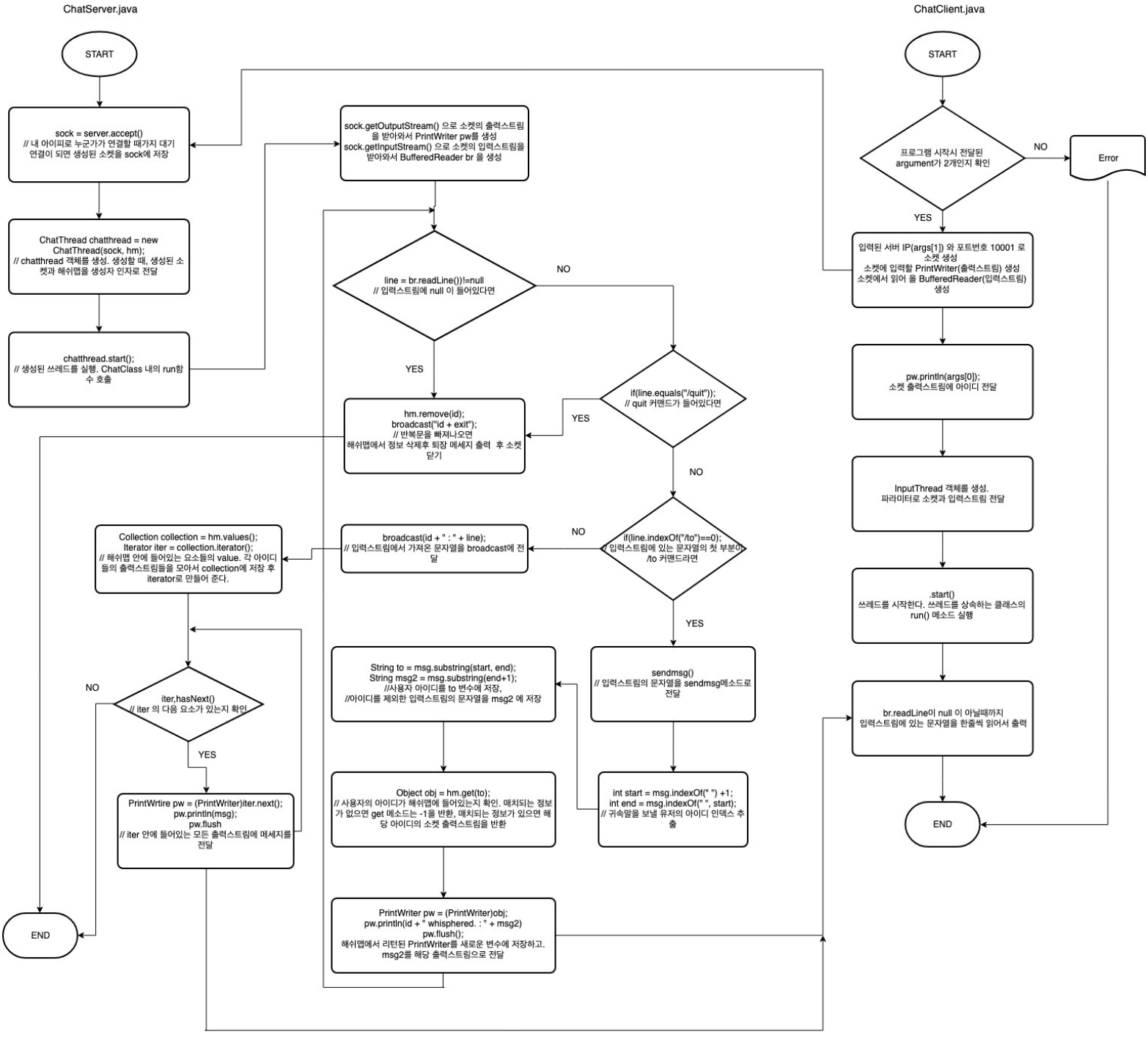
2. ChatServer 소스 line별 분석하여 각 line 뒤에 주석문으로 설명할 것

```
import java.net.*;
import java.io.*;
import java.util.*;

public class ChatServer {

        public static void main(String[] args) {
                try{
                        ServerSocket server = new ServerSocket(10001); // 소켓 만들기
                        System.out.println("Waiting connection...");
                        HashMap <String, PrintWriter> hm = new HashMap(); // key와 value의 자료형
                                                                                만들어두기
                        while(true){
                                Socket sock = server.accept(); // 내 아이피로 누군가가 연결하면 소켓이
만들어짐
                                ChatThread chatthread = new ChatThread(sock, hm);
                                chatthread.start(); // 스레드를 스타트 하면 밑에 Run 메소드가 실행됨
                        }
                }catch(Exception e){
                        System.out.println(e);
```

```
                }
        } // main
}

class ChatThread extends Thread{
        private Socket sock;
        private String id;
        private BufferedReader br;
        private HashMap hm;
        private boolean initFlag = false;
        public ChatThread(Socket sock, HashMap hm){
                this.sock = sock;
                this.hm = hm;
                try{
                        PrintWriter pw = new PrintWriter(new
OutputStreamWriter(sock.getOutputStream())); // 소켓에 출력하는 PrintWriter 생성
                        br = new BufferedReader(new
InputStreamReader(sock.getInputStream())); // 소켓에서 입력받는 BufferedReader 생성
                        id = br.readLine(); // 소켓에서 입력된 문자열 중 가장 첫줄을 읽어서 ID에 저장.
                        broadcast(id + " entered."); // broadcast 함수의 인자로 위에서 읽은 id를 인자로
                                                        넘겨준다.
                        System.out.println("[Server] User (" + id + ") entered.");
                        synchronized(hm){ // 여러 쓰레드에서 동시에 정보를 넣는 것을 방지하기 위해
                                                sychronized block 사용
                                hm.put(this.id, pw); // 해쉬맵에 아이디와 출력스트림을 저장한다.
                        }
                        initFlag = true;
                }catch(Exception ex){
                        System.out.println(ex);
                }
        } // constructor
        public void run(){
                try{
                        String line = null;
                        while((line = br.readLine()) != null){ // 소켓에 입력스트림의 정보를 한줄씩 읽는다.
                                if(line.equals("/quit"))  // /quit이 입력되면 반복 종료
                                        break;
                                if(line.indexOf("/to ") == 0){ // 사용자의 입력의 첫 부분에 /to 가 포함되어
있으면 sendmsg 입력된 문자열을 인자로 보내주며 호출
                                        sendmsg(line);
                                }else
                                        broadcast(id + " : " + line); // 아무 명령어 없이 그냥 문자열만 입력
됐다면 broadcast를 호출
                        }
                }catch(Exception ex){
                        System.out.println(ex);
                }finally{
                        synchronized(hm){
                                hm.remove(id); // 입력이 끝나면 해쉬맵에서 정보를 삭제
                        }
                        broadcast(id + " exited.");
                        try{
                                if(sock != null)
                                        sock.close();
                        }catch(Exception ex){}
                }
        } // run
        public void sendmsg(String msg){
                // sendmsg가 호출될 때는 사용자의 입력 가장 앞에 /to커맨드 + username 이 있다.
```

```
                int start = msg.indexOf(" ") +1; // 사용자가 입력한 정보의 첫 띄어쓰기가 있는 인덱스를 찾아서 start 변수에 저장.
                int end = msg.indexOf(" ", start); // 위에서 찾은 채팅 문자열부터 다음 띄어쓰기가 있는 인덱스를 찾아서 end에 저장. >> 사용자의 아이디가 있는 문자열 구간을 찾는 과정
                if(end != -1){
                        String to = msg.substring(start, end); // to변수에 사용자의 아이디를 문자열에서 잘라서 저장한다.
                        String msg2 = msg.substring(end+1); // 사용자의 아이디 이후에는 채팅 메세지가 있기 때문에 end+1 부터 끝까지 자르면 사용자의 채팅메세지를 저장할 수 있다.
                        Object obj = hm.get(to); // 해쉬맵에 사용자의 아이디가 있는지 확인해서 오브젝트 변수에 들어있는 출력스트림을 저장한다.
                        if(obj != null){
                                PrintWriter pw = (PrintWriter)obj; // 새로운 PrintWrite를 하나 만들어서 입력받은 사용자의 정보가 들어있는 출력스트림을 저장
                                pw.println(id + " whisphered. : " + msg2); // 이 출력 스트림에 채팅메세지를 보내준다.
                                pw.flush();
                        } // if
                }
        } // sendmsg
        public void broadcast(String msg){
                synchronized(hm){ // 쓰레드가 한번에 하나만 hm에 접근할 수 있게 한다.
                        Collection collection = hm.values(); // 해쉬맵의 모든출력스트림을 가져온다. >> 모든 유저에게 정보를 보내줘야하기 때문에
                        Iterator iter = collection.iterator(); // Iterator로 위에서 뽑은 출력스트림에 모두 접근
                        while(iter.hasNext()){
                                PrintWriter pw = (PrintWriter)iter.next(); // 출력스트림에 한번씩 접근하면서 이 메소드를 호출하면서 인자로 전달된 메세지를 보내준다.
                                pw.println(msg);
                                pw.flush();
                        }
                }
        } // broadcast
}
```

3. 소스를 분석하여 Flowchart 로 그릴 것



4. ChatClient.java 에서 InputThread를 만들어 사용하는 이유와 사용방식을 100자 이상 설명하라

InputThread 는 Thread 클래스를 상속하는 클래스이기 때문에 다수의 사용자가 클라이언트를 사용할 때 효율적으로 관리할 수 있다. InputThread 는 파라미터로 소켓과 입력스트림을 받는다. 인자가 전달되면서 InputThread 클래스 안에 있는 Socket 변수와 BufferedReader 변수에 현재 소켓과 입력스트림을 저장한다. InputThread 는 main 클래스에서 .start() 메소드가 실행되면 쓰레드가 작업을 하는데 필요한 공간을 만들고 run method를 호출한다. InputThread 안에는 run 메소드가 재정의 되어 있기 때문에 오버라이딩된 메소드가 실행된다. Run 메소드안에는 쓰레드가 시작되며 전달된 입력스트림에 있는 정보를 한 줄 씩 읽어 모두 출력한다.

5. ChatServer.java 에서 broadcast() 와 sendmsg() 에서 hm 사용법을 각각 100자 이상 설명하라

1. broadcast() : broadcast는 현재 서버에 접속중인 모든 사용자에게 메세지는 보내는 역할을 한다. broadcast는 collection 변수를 선언하고 그 안에 해쉬맵(hm)에 들어있는 value인 각 아이디와 매치되는 출력스트림을 가져와 저장한다. 각 정보에 저장된 데이터를 하나씩 접근하기 위해서 만들어진 collection을 Iterator에 다시 저장한다. 이후에 while문을 반복하며 hasNext() 메소드로 iterator안에 남아있는 정보가 있는지 확인하고 남은 정보가 없을 때까지 iterator 안에 들어있는 PrintWriter 들을 새로운 변수에 하나씩 저장하고 broadcast 메소드를 호출할때 함께 전달된 문자열 msg를 출력해준다. 이렇게 하면 현재 서버에 저장된 모든 아이디의 출력스트림에 메세지가 전달된다. 해쉬맵이 사용된 이유는 해쉬맵은 동시에 여러 데이터를 짝지어서 관리할 수 있기 때문에 아이디와 아이디에 맞는 소켓 출력스트림을 손쉽게 함께 관리할 수 기 때문이다.
2. sendmsg() : sendmsg 는 사용자가 /to [유저아이디] [메세지] 를 입력했을 때 호출된다. 메소드가 호출되면 가장 먼저 유저 아이디를 뽑아내는 작업을 수행한다. 그 이후에 입력된 메세지를 입려된 유저아이디에게만 보내게 된다. 해쉬맵 hm 안에는 아이디와 출력스트림이 짝지어서 관리되고 있다. 따라서 유저의 아이디만 알면 해쉬맵 자료구조에서 지원하는 메소드인 get() 로 쉽게 해당 아이디가 사용하는 출력스트림을 가져올 수 있다.