HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

COMPUTER NETWORK
ASSIGNMENT 01-SOCKET PROGRAMMING

Chat Application Report

Lecturer: Quan Thanh Tho
Author: Vuong Le Huy
Nguyen La Thong
Tran Vu Hong Thien

4/2020

Table 1: GROUP'S MEMBERS

No.	Name	ID
1	Vuong Le Huy	1652252
2	Nguyen La Thong	1752522
3	Tran Vu Hong Thien	1752506

Table 2: ACTIVITY LOG

Day	Changes	Member
7/05/2020	Create project; implement Server.java; Add methods to client.java.	Tran Vu Hoang Thien
10/05/2020	Design Chat window; Implement Chat manager.	Vuong Le Huy
19/05/2020	Add Chat manager;initialize file's event handler.	Tran Vu Hong Thien
20/04/2020	Check and fix chat listener issue; Add file transfer. Record fisrt demonstration.	Vuong Le Huy
23/05/2020	Check overall project; Finish up file transfer functions; Fix button listener issue; Compile application specifications into report.	Nguyen La Thong

Contents

1 Introduction

In this project, we wish to develop an application that enables two users to transmit and receive text and files via WiFi through socket programming. The primary programming language used in developing the application is Java. The application allows two users on two different machines to chat with each other and a person can chat with different people at the same time. The chat client is built on the hybrid model between client-server and P2P model and the system has a central server for user registration and online user management, clients chat directly to each other.

2 Server Programming

First let's take a glance at Server's Functionalities.

```
11      // Vector to store active clients
12      static Vector<ClientHandler> ar = new Vector<>();
13      static HashMap<String, String[]> list = new HashMap<>();
```

Active client's data are stored in a HashMap variable called "list" (The HashMap class is roughly equivalent to Hash table, except that it is unsynchronized and permits nulls.)

```java
    public static void main(String[] args) throws IOException
    {
        // server is listening on port 1234
        ServerSocket ss = new ServerSocket(1234);

        // running infinite loop for getting
        // client request
        while (true)
        {
            // Accept the incoming request
            Socket s = ss.accept();

            System.out.println("New client request received : " + s);

            System.out.println("Creating a new handler for this client...");

            // Create a new handler object for handling this request.
            ClientHandler mtch = new ClientHandler(s);

            // Create a new Thread with this object.
            Thread t = new Thread(mtch);

            System.out.println("Adding this client to active client list");

            // add this client to active clients list
            ar.add(mtch);

            // start the thread.
            t.start();
        }
    }
}
```

Server listens to requests from a predetermined port and call a handler which processes those requests. Once request process completed, a thread is initialized and started after clients is added to active clients list.

```java
// ClientHandler class
class ClientHandler implements Runnable
{
    private Socket s;

    // constructor
    public ClientHandler(Socket s) {
        this.s = s;
    }

    public Socket getSocket() {
        return s;
    }

    @Override
    public void run() {
            try
            {
                ObjectOutputStream dos = new ObjectOutputStream(s.getOutputStream());
                dos.writeObject(Server.list);
                DataInputStream dis = new DataInputStream(s.getInputStream());
                String key = dis.readUTF();
                String[] value = new String[2];
                value[0] = s.getInetAddress().getHostAddress();
                value[1] = Integer.toString(s.getPort());
                Server.list.put(key, value);
                for (ClientHandler mc : Server.ar) {
                    dos = new ObjectOutputStream(mc.getSocket().getOutputStream());
                    dos.writeObject(Server.list);
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
    }
}
```

Client handler primarily write client's address and port to server's list.

3 Client Programming

With server successfully initialized, clients can now gain access to it.

```
11      private HashMap <String,String[]> list;
12      private ArrayList <ChatManager> cm = new ArrayList<ChatManager>();
13      private final int ServerPort = 1234;
14      private String destinationPath = "C:\\Users\\Public\\File\\";
15      private String username;
```

Clients also have a list to stored information of other clients who connect to a common port.

```
17      public static void main(String args[]) throws UnknownHostException, IOException
18      {
19          Client client = new Client();
20          client.createConnection();
21      }
22
23      public ArrayList<ChatManager> getChatManagerList() {
24          return cm;
25      }
26
27      public void addChatManagerList(ChatManager c) {
28          cm.add(c);
29      }
30
31      public String getUsername() {
32          return username;
33      }
34
35      public void createConnection()
36      {
```

Client's methods

```
35      public void createConnection()
36      {
37          try {
38          Scanner scn = new Scanner(System.in);
39          // getting localhost ip
40          System.out.println("Enter server IP address: ");
41          String addr = scn.nextLine();
42          InetAddress ip = InetAddress.getByName(addr);
43
44          // establish the connection
45          Socket s = new Socket(ip, ServerPort);
46          (new Thread(){public void run() {
47              try {
48                  while (true) {
49                      ObjectInputStream ois = new ObjectInputStream(
50                              s.getInputStream());
51                      list = (HashMap<String, String[]>) ois.readObject();
52                  }
53              } catch (SocketException e) {
54                  e.printStackTrace();
55              } catch (IOException e) {
56                  e.printStackTrace();
57              } catch (ClassNotFoundException e) {
58                  e.printStackTrace();
59              }
60          }}).start();
61          DataOutputStream dos = new DataOutputStream(s.getOutputStream());
62          System.out.println("Enter username: ");
63          username = scn.nextLine();
64          dos.writeUTF(username);
65          DatagramSocket ds = new DatagramSocket(s.getLocalPort());
66          ChatListener r = new ChatListener(ds,this);
67          Thread t = new Thread(r);
68          t.start();
69          String peer;
70          while (true) {
71              System.out.println("Who do you want to talk to ?");
72              peer = scn.nextLine();
73              String[] p = list.get(peer);
74              if (p == null) {
75                  System.out.println("Wrong username");
76                  continue;
77              }
78              String ipaddress = p[0];
79              String port = p[1];
80              String sa = "/"+ipaddress+":"+port;
81              boolean breakFlag = false;
82
```

```
83              for (ChatManager x : cm) {
84                  if (x.getSocketAddress().equals(sa)) {
85                      breakFlag = true;
86                      break;
87                  }
88              }
89
90              if (!breakFlag) {
91              ChatManager c = new ChatManager(peer,ipaddress,port,ds,username);
92              cm.add(c);
93              c.start();
94              }
95          }
96          } catch (UnknownHostException e) {
97              e.printStackTrace();
98          } catch (SocketException e) {
99              e.printStackTrace();
100         } catch (IOException e) {
101             e.printStackTrace();
102         }
103     }
104 }
105
```

By entering server's IP address and username, clients gain access to the server and be eligible to kick off a conversation with other active clients.

4 Chat Manager

Chat manager is added to oversee chat forum and handle files transfer.

```
26      private Thread t;
27      private ChatWindow cw;
28      private String ipaddress;
29      private String port;
30      private DatagramSocket ds;
31      private String destinationPath = "C:\\Users\\Public\\File\\";
32      private String sender;
33      private String receiver;
```

Chat manager retains IP address, port of senders and receivers as well as destination path of the message.

```
44      public String getSocketAddress() {
45          return "/"+ipaddress+":"+port;
46      }
47
48      @Override
49      public void run() {
50          cw.getButton().addActionListener(new SendButtonListener());
51      }
52
53      public void createFrame() {
54          cw = new ChatWindow(receiver);
55      }
56
57      public void addTextFromOtherUser(String string) {
58          cw.addTextFromOtherUser(string);
59      }
60
61      public void start() throws IOException {
62          createFrame();
63
64          if (t==null) {
65              t = new Thread(this);
66              t.start();
67          }
68      }
```

Essentially, chat manager get socket address, create frame for chat forum and add text to it.

```
70  class SendButtonListener implements ActionListener {
71
72      @Override
73      public void actionPerformed(ActionEvent e){
74          try {
75          String a= cw.getText();
76          String msg =sender+":"+a;
77          DatagramPacket DpSend = new DatagramPacket(msg.getBytes(),msg.getBytes().length,InetAddress.getByName(ipaddress),Integer.parseInt(port));
78          ds.send(DpSend);
79          cw.addTextFromThisUser(msg);
80          if (msg.contains("\\")) {
81              FileEvent event = getFileEvent(a);
82              ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
83              ObjectOutputStream os = new ObjectOutputStream(outputStream);
84              os.writeObject(event);
85              byte[] data = outputStream.toByteArray();
86              DpSend = new DatagramPacket(data, data.length,InetAddress.getByName(ipaddress),Integer.parseInt(port));
87              ds.send(DpSend);
88              System.out.println("File sent from client");
89          }
90          } catch (UnknownHostException er) {
91              er.printStackTrace();
92          } catch (SocketException er) {
93              er.printStackTrace();
94          } catch (IOException er) {
95              er.printStackTrace();
96          }
97      }
98  }
```

Necessarily, a button listener is added to alert the chat manager to the sending event.

5 Chat Window

```
15      JFrame mainFrame;
16      JTextField textBox;
17      JTextArea msgArea;
18      JPanel controlPanel;
19      JButton sendButton;
20      String usernames;
```

Chat window is responsible for retaining frame and user interface of the chat box.

```
22      ChatWindow(String _usernames) {
23          mainFrame = new JFrame("");
24          textBox = new JTextField();
25          msgArea = new JTextArea();
26          controlPanel = new JPanel();
27          sendButton = new JButton("Send");
28          sendButton.setPreferredSize(new Dimension(40, 60));
29          usernames = _usernames;
30
31          addComponents();
32      }
```

Chat window configuration.

```
34      public void addComponents() {
35          mainFrame.setLayout(new GridLayout(0,1));
36          controlPanel.setLayout(new BorderLayout());
37
38          mainFrame.add(msgArea);
39          msgArea.append("STARTING CHAT WITH " + usernames);
40          mainFrame.add(textBox);
41          mainFrame.add(controlPanel);
42          controlPanel.add(sendButton, BorderLayout.SOUTH);
43
44          mainFrame.setVisible(true);
45          mainFrame.setSize(600, 600);
46          mainFrame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
47      }
48
49      public JButton getButton() {
50          return sendButton;
51      }
52
53      public String getText() {
54          return textBox.getText();
55      }
56
57      public void addTextFromThisUser(String string) {
58          msgArea.append("\n" + string);
59          textBox.setText("");
60          textBox.requestFocus();
61      }
62
63      public void addTextFromOtherUser(String string) {
64          msgArea.append("\n" + string);
65      }
66  }
```

Methods for monitoring user's view.

6 Chat Listener

```
25      public void run() {
26          try {
27              byte[] receive = new byte[65535];
28              DatagramPacket DpReceive = null;
29              while (true) {
30                  DpReceive = new DatagramPacket(receive, receive.length);
31                  ds.receive(DpReceive);
32                  String message = new String (DpReceive.getData(), 0, DpReceive.getLength());
33                  String[] peer = message.split(":");
34                  boolean breakFlag = false;
35                  for (ChatManager x : c.getChatManagerList()) {
36                      if (x.getSocketAddress().equals(DpReceive.getSocketAddress().toString())) {
37                          x.addTextFromOtherUser(message);
38                          breakFlag = true;
39                          break;
40                      }
41                  }
42
43                  if (!breakFlag) {
44                      String[] b = DpReceive.getSocketAddress().toString().substring(1).split(":");
45                      String ipaddress = b[0];
46                      String port = b[1];
47                      ChatManager cm = new ChatManager(peer[0],ipaddress,port,ds,c.getUsername());
48                      c.addChatManagerList(cm);
49                      cm.start();
50                      cm.addTextFromOtherUser(message);
51                  }
52                  if (message.contains("\\")) {
53                  byte[] incomingData = new byte[1024 * 1000 * 50];
54                      DatagramPacket incomingPacket = new DatagramPacket(incomingData, incomingData.length);
55                      ds.receive(incomingPacket);
56                      byte[] data = incomingPacket.getData();
57                      ByteArrayInputStream in = new ByteArrayInputStream(data);
58                      ObjectInputStream is = new ObjectInputStream(in);
59                      fileEvent = (FileEvent) is.readObject();
60                      if (fileEvent.getStatus().equalsIgnoreCase("Error")) {
61                          System.out.println("Some issue happened while packing the data @ client side");
62                          continue;
63                      }
64                      createAndWriteFile();
65                  receive = new byte[65535];
66                  }
67              }
68          } catch (IOException e) {
69              e.printStackTrace();
70          }
71          catch (ClassNotFoundException e) {
72              e.printStackTrace();
73          }
```

Listen to messages from other clients in forum.

```
76      public void createAndWriteFile() {
77      String outputFile = fileEvent.getDestinationDirectory() + fileEvent.getFilename();
78      if (!new File(fileEvent.getDestinationDirectory()).exists()) {
79          new File(fileEvent.getDestinationDirectory()).mkdirs();
80      }
81
82      File dstFile = new File(outputFile);
83      FileOutputStream fileOutputStream = null;
84
85      try {
86          fileOutputStream = new FileOutputStream(dstFile);
87          fileOutputStream.write(fileEvent.getFileData());
88          fileOutputStream.flush();
89          fileOutputStream.close();
90          System.out.println("Output file : " + outputFile + " is successfully saved ");
91      } catch (FileNotFoundException e) {
92          e.printStackTrace();
93      } catch (IOException e) {
94          e.printStackTrace();
95      }
96
97      }
98
99  }
```

File writing method

7 File's Events Handler

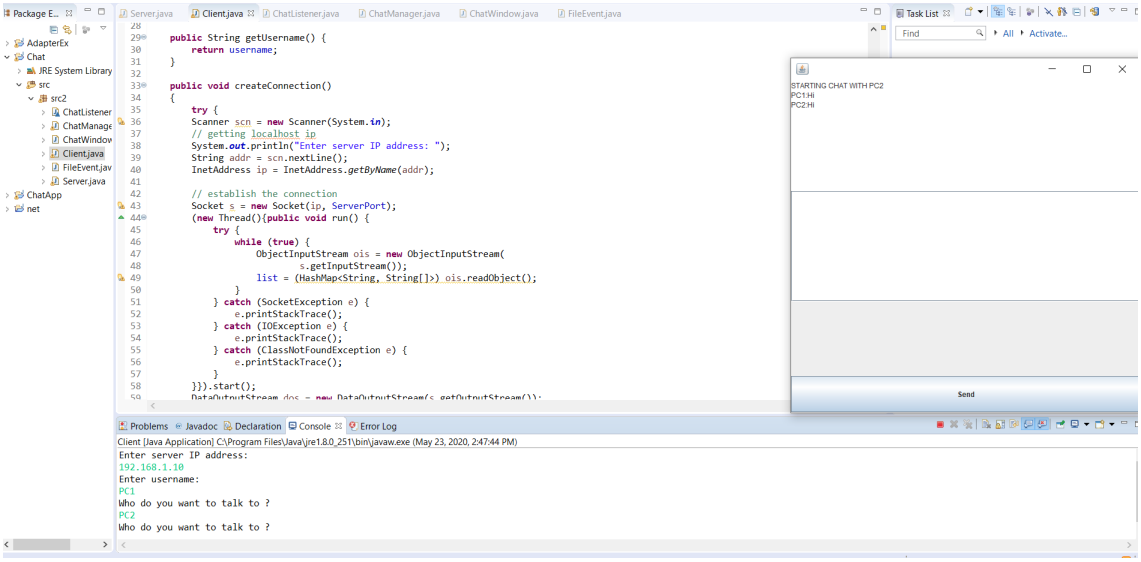
```
100     public FileEvent getFileEvent(String message) {
101         FileEvent fileEvent = new FileEvent();
102         String fileName = message.substring(message.lastIndexOf("\\") + 1, message.length());
103         //String sourceFilePath = message.substring(0, message.lastIndexOf("\\") + 1);
104         System.out.println(message);
105         fileEvent.setDestinationDirectory(destinationPath);
106         fileEvent.setFilename(fileName);
107         fileEvent.setSourceDirectory(message);
108         File file = new File(message);
109         if (file.isFile()) {
110             try {
111             DataInputStream diStream = new DataInputStream(new FileInputStream(file));
112             long len = (int) file.length();
113             byte[] fileBytes = new byte[(int) len];
114             int read = 0;
115             int numRead = 0;
116             while (read < fileBytes.length && (numRead = diStream.read(fileBytes, read, fileBytes.length - read)) >= 0) {
117                 read = read + numRead;
118             }
119             fileEvent.setFileSize(len);
120             fileEvent.setFileData(fileBytes);
121             fileEvent.setStatus("Success");
122             } catch (Exception e) {
123                 e.printStackTrace();
124                 fileEvent.setStatus("Error");
125             }
126         } else {
127             System.out.println("path specified is not pointing to a file");
128             fileEvent.setStatus("Error");
129         }
130         return fileEvent;
131     }
132 }
```

File's events handler which directs files to its dedicated destination and report transfer status.

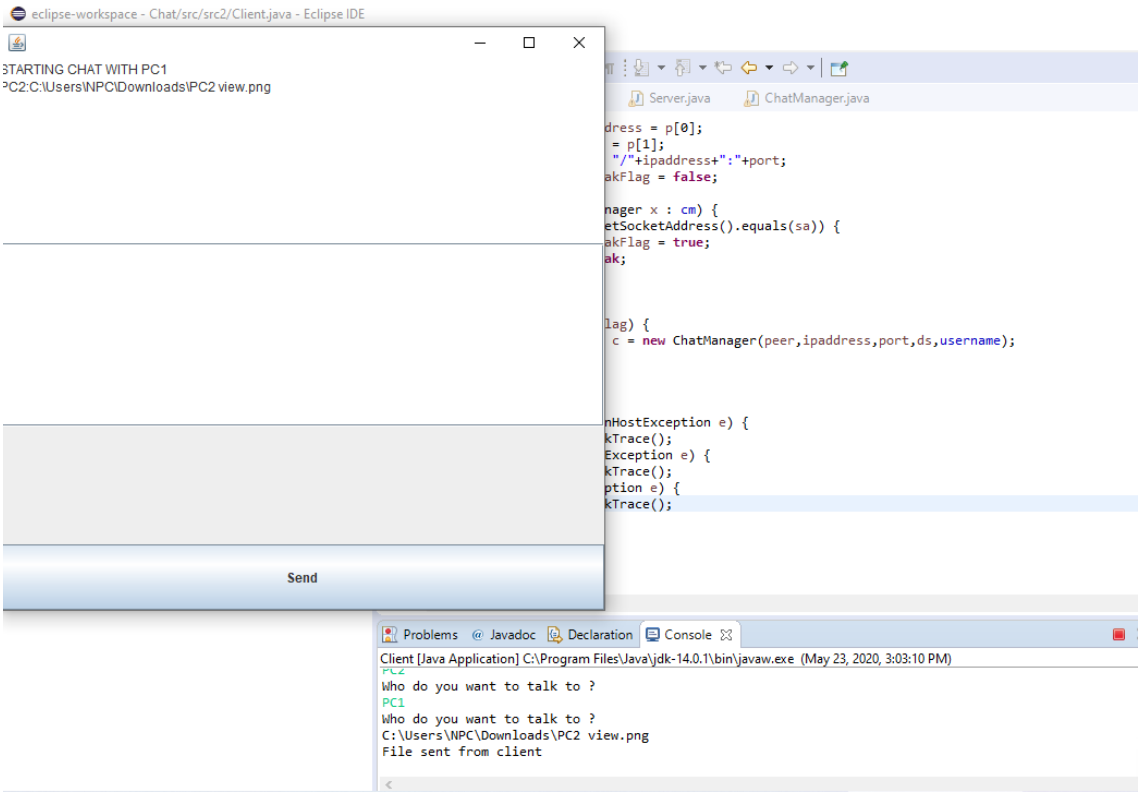
```
19    public String getDestinationDirectory() {
20    return destinationDirectory;
21    }
22
23    public void setDestinationDirectory(String destinationDirectory) {
24    this.destinationDirectory = destinationDirectory;
25    }
26
27    public String getSourceDirectory() {
28    return sourceDirectory;
29    }
30
31    public void setSourceDirectory(String sourceDirectory) {
32    this.sourceDirectory = sourceDirectory;
33    }
34
35    public String getFilename() {
36    return filename;
37    }
38
39    public void setFilename(String filename) {
40    this.filename = filename;
41    }
42
43    public long getFileSize() {
44    return fileSize;
45    }
46
47    public void setFileSize(long fileSize) {
48    this.fileSize = fileSize;
49    }
50
51    public String getStatus() {
52    return status;
53    }
54
55    public void setStatus(String status) {
56    this.status = status;
57    }
58
59    public byte[] getFileData() {
60    return fileData;
61    }
62
63    public void setFileData(byte[] fileData) {
64    this.fileData = fileData;
65    }
```

Primarily, File's event handler stores file's name, data, size, and delivery status.

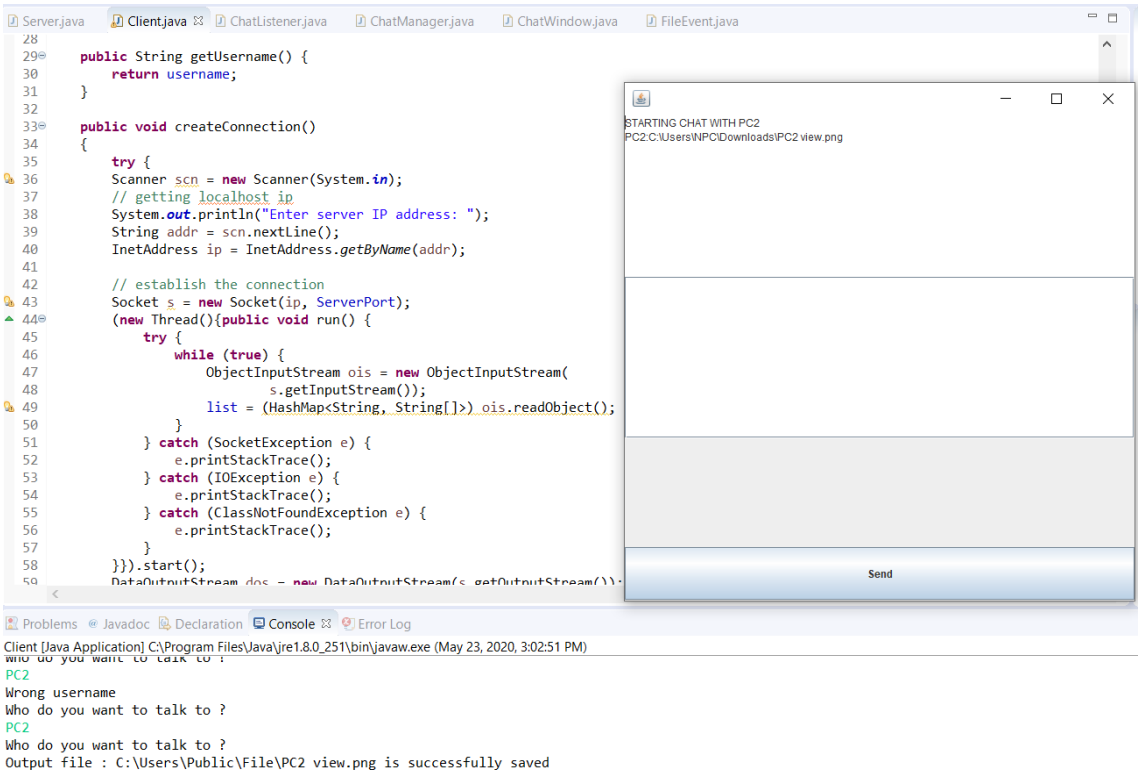
8 Application Demonstration



Chat Forum between two clients PC1 and PC2.



Sending files in application.



File transfer is completed and successfully saved.