Chat GUI

Készítsünk egy csoportos Chat vastag kliens alkalmazást. Alkalmazzuk a Java-RMI technológiát. Hozzunk létre két projektet az eclipse-ben a kliens és a szerver számára. Az alábbi interakciós diagramm bemutatja az elvi működést.

sequenceDiagram participant U as User participant UI as ChatUI participant CC as ChatClient participant CS as ChatServer participant IC as IChatClient participant IS as IChatServer U->>UI: Starts Application UI->>CC: Creates ChatClient CC->>IS: Tries to connect IS->>CS: Login request CS->>IC: Checks if client can connect IC->>CC: Connection successful CC->>UI: Update UI (Connected) UI->>U: Show connected status U->>UI: Enter message UI->>CC: Send message CC->>IS: Publish message IS->>CS: Publish message to all clients CS->>IC: Iterate over clients IC->>CC: Display message CC->>UI: Update UI with new message UI->>U: Show new message U->>UI: Disconnect UI->>CC: Disconnect from server CC->>IS: Logout request IS->>CS: Remove client from list CS->>IC: Update client list IC->>CC: Acknowledge disconnect CC->>UI: Update UI (Disconnected) UI->>U: Show disconnected status

Készíthetünk egy másik diagrammot is a működés bemutatásásra:

- **ChatUI** a felhasználói interfész, amely közvetlenül kommunikál a ChatClient-tel.
- **ChatClient** a kliens oldali logika, amely megvalósítja az IChatClient interfészt.
- **Java RMI Registry** a szolgáltatások nevének feloldására szolgál, amit a ChatClient használ a IChatServer interfész elérésére.
- **IChatServer** az interfész, amelyen keresztül a ChatClient kommunikál a ChatServer-rel.
- **ChatServer** a szerver oldali logika, amely megvalósítja az IChatServer interfészt.
- **Client List** a csatlakoztatott kliensek listája a szerveren.
- **User Interface** ábrázolja a felhasználói felület komponenseit, amelyek interakcióban állnak a felhasználóval.

E z a diagram bemutatja a rendszer összetevőinek kapcsolódási és kommunikációs struktúráját, beleértve a kliens és szerver közötti interakciókat.

graph TD A[ChatUI] --> B[ChatClient] B --> C{Java RMI Registry} B -->|lookup| D[IChatServer] D --> E[ChatServer] E --> F[IChatClient] F --> B D -.-> G[Client List] E --> G A -.-> H[User Interface] H --> A C -.->|registers| E style A fill:#f9f,stroke:#333,stroke-width:4px style B fill:#bbf,stroke:#f66,stroke-width:2px,stroke-dasharray: 5, 5 style C fill:#fea,stroke:#333,stroke-width:2px style D fill:#bbf,stroke:#f66,stroke-width:2px,stroke-dasharray: 5, 5 style E fill:#f99,stroke:#333,stroke-width:4px style F fill:#bbf,stroke:#f66,stroke-width:2px,stroke-dasharray: 5, 5 style G fill:#dff,stroke:#333,stroke-width:2px style H fill:#fff,stroke:#333,stroke-width:2px

Project 1: ChatServer

1.) IChatClient

```
import java.rmi.Remote;
import java.rmi.RemoteException;
public interface IChatClient extends Remote {
      public void tell(String name) throws RemoteException;
```

```
        public String getName() throws RemoteException;
  }
```

2.) IChatServer

```
  import java.rmi.Remote;
  import java.rmi.RemoteException;
  import java.util.ArrayList;
  public interface IChatServer extends Remote {
      public boolean login(IChatClient client) throws RemoteException;
      public void publish(String s) throws RemoteException;
      public ArrayList<IChatClient> getConnected() throws RemoteException;
  }
```

3.) ChatServer

```
  import java.rmi.RemoteException;
  import java.rmi.server.UnicastRemoteObject;
  import java.util.ArrayList;
  public class ChatServer extends UnicastRemoteObject implements IChatServer
{
      private static final long serialVersionUID = 4705993250126188735L;
      private ArrayList<IChatClient> v = new ArrayList<IChatClient>();
      public ChatServer() throws RemoteException {
      }
      public boolean login(IChatClient client) throws RemoteException {
          System.out.println(client.getName() + " got connected....");
          client.tell("client connected");
          publish(client.getName() + " connected.");
          v.add(client);
          return true;
      }
      public void publish(String s) throws RemoteException {
          System.out.println(s);
          for (int i = 0; i < v.size(); i++) {
              try {
                  IChatClient tmp = (IChatClient) v.get(i);
                  tmp.tell(s);
              } catch (Exception e) {
                  e.printStackTrace();
              }
          }
      }
      public ArrayList<IChatClient> getConnected() throws RemoteException {
          return v;
      }
  }
```

4.) StartServer

```
  import java.rmi.Naming;
  public class StartServer {
      public static void main(String[] args) {
          try {
              // System.setSecurityManager(new RMISecurityManager());
              java.rmi.registry.LocateRegistry.createRegistry(1099);
              IChatServer server = new ChatServer();
              Naming.rebind("rmi://localhost:1099/chatserver", server);
              System.out.println("server ready.");
          } catch (Exception e) {
              e.printStackTrace();
          }
      }
  }
```

Client Side

Project 2: ChatClient

1.) ChatClient

```
  import java.rmi.RemoteException;
  import java.rmi.server.UnicastRemoteObject;
  public class ChatClient extends UnicastRemoteObject implements IChatClient
{
      private String name;
      private ChatUI ui;
      public ChatClient(String n) throws RemoteException {
          name = n;
      }
      public void tell(String st) throws RemoteException {
          System.out.println(st);
          ui.writeMsg(st);
      }
      public String getName() throws RemoteException {
          return name;
      }
      public void setGUI(ChatUI t) {
          ui = t;
      }
  }
```

2.) ChatUI

```
  import java.awt.BorderLayout;
  import java.awt.GridLayout;
  import java.awt.event.ActionEvent;
  import java.awt.event.ActionListener;
  import java.rmi.Naming;
  import java.util.ArrayList;
  import javax.swing.DefaultListModel;
  import javax.swing.JButton;
  import javax.swing.JFrame;
  import javax.swing.JLabel;
  import javax.swing.JList;
  import javax.swing.JOptionPane;
  import javax.swing.JPanel;
  import javax.swing.JScrollPane;
  import javax.swing.JTextArea;
  import javax.swing.JTextField;
  import javax.swing.border.EmptyBorder;
  public class ChatUI {
      private ChatClient client;
      private IChatServer server;
      public void doConnect() {
          if (connect.getText().equals("Connect")) {
              if (name.getText().length() < 2) {
                  JOptionPane
                          .showMessageDialog(frame, "You need to type a
name.");
                  return;
              }
              if (ip.getText().length() < 2) {
                  JOptionPane.showMessageDialog(frame, "You need to type an
IP.");
                  return;
              }
              try {
                  client = new ChatClient(name.getText());
                  client.setGUI(this);
                  server = (IChatServer)
Naming.lookup("rmi://localhost/chatserver");
                  server.login(client);
                  updateUsers(server.getConnected());
                  connect.setText("Disconnect");
              } catch (Exception e) {
                  e.printStackTrace();
                  JOptionPane.showMessageDialog(frame,
                          "ERROR, we wouldn't connect....");
              }
          } else {
              updateUsers(null);
              connect.setText("Connect");
          }
```

```
      }
      public void sendText() {
          if (connect.getText().equals("Connect")) {
              JOptionPane.showMessageDialog(frame, "You need to connect
first.");
              return;
          }
          String st = tf.getText();
          st = "[" + name.getText() + "] " + st;
          tf.setText("");
          // Remove if you are going to implement for remote invocation
          try {
              server.publish(st);
          } catch (Exception e) {
              e.printStackTrace();
          }
      }
      public void writeMsg(String st) {
          tx.setText(tx.getText() + "\n" + st);
      }
      public void updateUsers(ArrayList<IChatClient> users) {
          DefaultListModel<String> listModel = new
DefaultListModel<String>();
          if (users != null)
              for (int i = 0; i < users.size(); i++) {
                  try {
                      String tmp = ((IChatClient) users.get(i)).getName();
                      listModel.addElement(tmp);
                  } catch (Exception e) {
                      e.printStackTrace();
                  }
              }
          list.setModel(listModel);
      }
      public static void main(String[] args) {
          System.out.println("Hello World !");
          ChatUI c = new ChatUI();
      }
      // User Interface code.
      public ChatUI() {
          frame = new JFrame("Group Chat");
          JPanel main = new JPanel();
          JPanel top = new JPanel();
          JPanel cn = new JPanel();
          JPanel bottom = new JPanel();
          ip = new JTextField();
          tf = new JTextField();
          name = new JTextField();
          tx = new JTextArea();
          connect = new JButton("Connect");
          JButton bt = new JButton("Send");
```

```
            list = new JList();
            main.setLayout(new BorderLayout(5, 5));
            top.setLayout(new GridLayout(1, 0, 5, 5));
            cn.setLayout(new BorderLayout(5, 5));
            bottom.setLayout(new BorderLayout(5, 5));
            top.add(new JLabel("Your name: "));
            top.add(name);
            top.add(new JLabel("Server Address: "));
            top.add(ip);
            top.add(connect);
            cn.add(new JScrollPane(tx), BorderLayout.CENTER);
            cn.add(list, BorderLayout.EAST);
            bottom.add(tf, BorderLayout.CENTER);
            bottom.add(bt, BorderLayout.EAST);
            main.add(top, BorderLayout.NORTH);
            main.add(cn, BorderLayout.CENTER);
            main.add(bottom, BorderLayout.SOUTH);
            main.setBorder(new EmptyBorder(10, 10, 10, 10));
            // Events
            connect.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                    doConnect();
                }
            });
            bt.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                    sendText();
                }
            });
            tf.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                    sendText();
                }
            });
            frame.setContentPane(main);
            frame.setSize(600, 600);
            frame.setVisible(true);
        }
        JTextArea tx;
        JTextField tf, ip, name;
        JButton connect;
        JList list;
        JFrame frame;
    }
```

Feladat 1.: implementáljuk, hogy új kliensek belépése esetén, a régiek is megkapják az új belépő nevét.

Feladat 2.: implementáljuk továbbá, hogy kilépés esetén a felhasználók törlődjenek.

Feladat 3.: implementájuk, hogy az új belépők megkapják a chat history-t