Chat Server 1

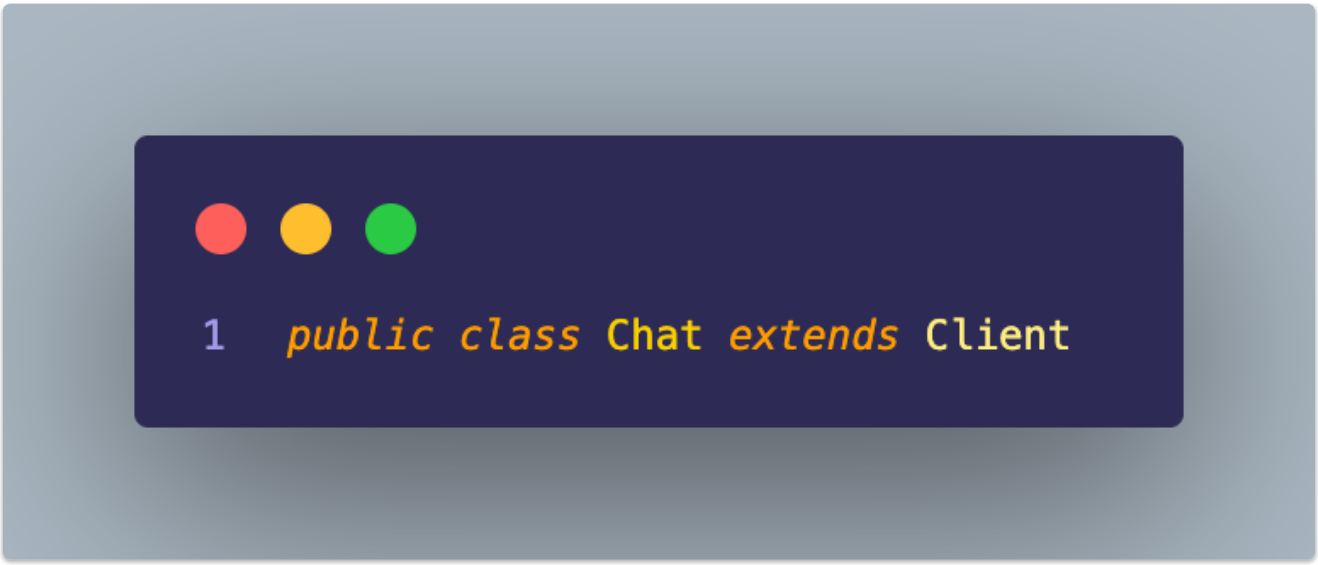
> ⓘ **Info**
>
> Der Chat Server benutzt die Klassen Client und Server und erstellt eine Java Frame GUI bei der man Nachrichten an den Server senden kann. Dieser verarbeitet die Nachrichten anschließend und sendet diese an alle Teilnehmer im Netzwerk (per Port) zurück.

Ordnerstruktur

- 🗁 `src`
  - 🗁 `classes`
    - 🗁 `client`
      - 🗎: `Chat.java`
      - 🗎: `Interface.java`
    - 🗁 `server`
      - 🗎: `ServerProcess.java`
  - 🗁 `imports`
    - 🗎: `Client.java`
    - 🗎: `Server.java`
    - 🗎: `List.java`
  - 🗎: `Main.java`

Client Scripts

> ⓘ **Info**
>
> Der Client repräsentiert alles was local auf dem PC einer Person passiert. Hierbei werden der Chat und das Interface zum Senden der Nachrichten genutzt.



Die Klasses Chat wird hierbeit als Extention der Klasse Client verwendet und stellt die Verbindung zum Server auf und das Interface, das genutzt wird um die Nachrichten anzuzeigen.

Die Klasse "Chat" erbt von einer anderen Klasse und hat einen Konstruktor, der eine IP-Adresse und einen Port als Parameter erwartet.

Im Konstruktor wird zunächst die Oberfläche (Interface) für den Chat erstellt und dem Konstruktor die aktuelle Instanz von "Chat" übergeben.

Anschließend wird geprüft, ob eine Verbindung zum Server aufgebaut werden konnte. Sollte dies nicht der Fall sein, wird eine Fehlermeldung ausgegeben, die Oberfläche geschlossen und das Programm beendet.

```java
@Override
public void processMessage(String pMessage) {
    do {
        try {
            TimeUnit.SECONDS.sleep(1);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    } while (this.chatInterface == null);
    this.chatInterface.AddMessage(pMessage);
}
```

Dieser Code definiert die Methode "`processMessage`", die eine Textnachricht als Eingabe erhält und dann bestimmte Aktionen ausführt. Die Methode verarbeitet die Nachricht, indem sie zuerst darauf wartet, dass das "`chatInterface`" Objekt initialisiert wird, und dann prüft, ob die Nachricht mit "`TriggerClientEvent`" beginnt.
Wenn dies der Fall ist, wird das zweite Element der Nachricht analysiert, um eine entsprechende Aktion auszuführen.
Nachdem das zweite Element "`Connected`" ist, wird eine Meldung ausgegeben, dass der Benutzer dem Chat-Server beigetreten ist.
Falls kein Match gefunden wird, wird die Methode einfach beendet, andernfalls wird die ursprüngliche Nachricht an das "`chatInterface`" Objekt weitergeleitet, um sie anzuzeigen.

ⓘ Info

Es wird gewartet ob eine Instanz vom GUI existiert, da der Server bei der Verbindung einer Client eine Bestätigungsnachricht sendet und dies beim Start des Programms dazu führt (aufgrund der asynchronen Processen), dass die Instanz des GUI's noch nicht

existiert. Dementsprechend wird so verhindert, dass eine nicht-existierende Instanz eines Objekts verwendet wird.

Ablauf der Nachrichtenübertragung (Sicht des Clients)

Important

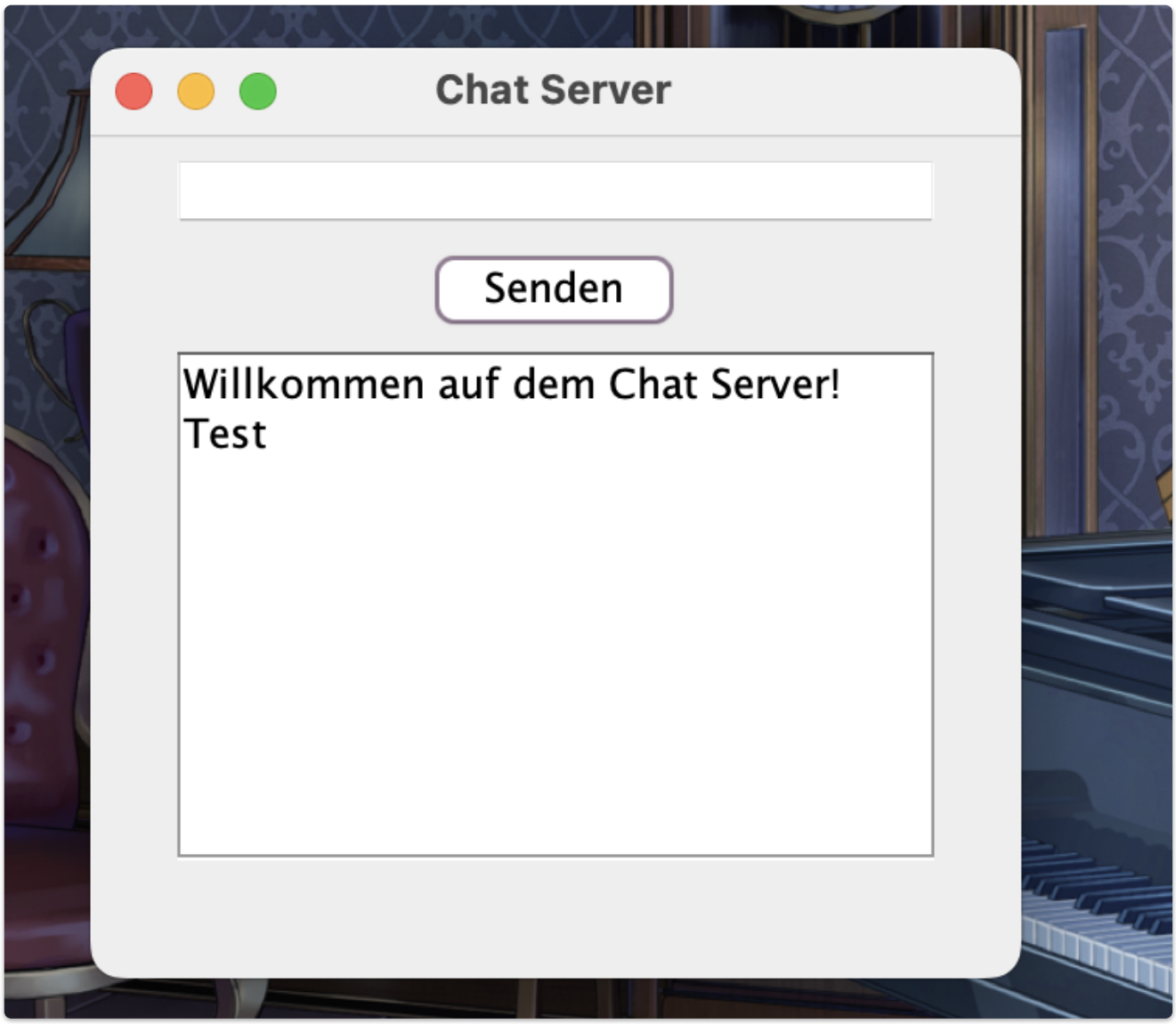
Diese Bilder sind nicht mehr aktuell! Das GUI sieht inzwischen anders aus und hat auch andere Ausgaben!

Willkommensnachricht bei beitreten des Chat Servers

Eingabe der Nachricht

Senden einer Nachricht

Anzeige der Übertragenen Nachricht



Server Script

```java
public class ServerProcess extends Server {
    SQL MySQL;

    public ServerProcess(int pPort, SQL MySQL) {
        super(pPort);
        this.MySQL = MySQL;
    }

    @Override
    public void processNewConnection(String pClientIP, int pClientPort) {
        if (!this.isOpen()) {
            return;
        }
        // String messageHistory;
        String[][] messages = this.MySQL.select("SELECT * FROM Chat WHERE 1");
        if (messages != null) {
            for (String[] row : messages) {
                for (String message : row) {
                    System.out.println("Nachricht:");
                    System.out.println(message + "\t");
                }
            }
        }
        this.send(pClientIP, pClientPort, "Verbunden");
    }

    @Override
    public void processMessage(String pClientIP, int pClientPort, String pMessage) {
        System.out.println("Nachricht wurde empfangen: " + pClientIP + " " + pClientPort + " " + pMessage);
        this.sendToAll(pMessage);
        String[] args = pMessage.split(":");
        this.MySQL.insert("INSERT INTO CHAT (owner, content) VALUES (" + args[0] + ", " + args[1] +")");
    }

    @Override
    public void processClosingConnection(String pClientIP, int pClientPort) {
        this.sendToAll("Verbindung zum Server wurde abgebrochen. Grund: Server wurde geschlossen.");
    }
}

```

Diese Klasse "ServerProcess" erweitert die "Server"-Klasse und fügt spezifische Funktionen hinzu. Der Konstruktor erhält einen Port und eine Instanz der SQL-Klasse, um eine Verbindung zur Datenbank herzustellen.

Die Methode "processNewConnection" wird aufgerufen, wenn ein neuer Client eine Verbindung zum Server herstellt. Zunächst wird überprüft, ob der Server geöffnet ist. Wenn dies nicht der Fall ist, wird die Methode beendet. Ansonsten wird die Verbindung als erfolgreich bestätigt, indem eine Nachricht an den Client gesendet wird. Außerdem werden die vorherigen Chat-Nachrichten aus der Datenbank abgerufen und auf der Serverkonsole ausgegeben.

Die Methode "processMessage" wird aufgerufen, wenn eine Nachricht von einem Client empfangen wird. Die Methode sendet die Nachricht an

alle verbundenen Clients weiter und fügt die empfangene Nachricht in die Datenbank ein.

Die Methode "processClosingConnection" wird aufgerufen, wenn eine Verbindung zu einem Client geschlossen wird. Eine Nachricht wird an alle verbundenen Clients gesendet, um den Grund für die Trennung zu informieren.

Empfangene Nachrichten der Clients

```
Nachricht wurde empfangen: 127.0.0.1
49809 Test
```

```
ublic Interface(Chat chatServer, String username) {
        this.chatServer = chatServer;

        this.username = username;
        this.chatServer.send(this.username + ": ist dem Chat beigetreten!");

        this.frame = new JFrame("Chat Server");
        this.frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.frame.setSize(800, 600);

        JPanel panel = new JPanel();

        this.textField = new JTextField(40);
        panel.add(this.textField);

        JButton button = new JButton("Senden");
        panel.add(button);

        this.textArea = new JTextArea(30, 50);
        this.textArea.setEditable(false);

        JScrollPane scrollPane = new JScrollPane(this.textArea);
        panel.add(scrollPane);

        button.addActionListener(e -> {
            String text = this.textField.getText();
            this.textField.setText("Sende..");
            this.chatServer.send(this.username + ": " + text);
        });

        this.frame.add(panel);
        this.frame.setVisible(true);
    }

    public void AddMessage(String message) {
        this.textField.setText("");
        this.textArea.append(message + "\n");
    }

    public void CloseAll() {
        this.frame.setVisible(false);
    }
}
```

Dieser Code definiert eine Benutzeroberfläche für einen Chat-Client, die es Benutzern ermöglicht, Textnachrichten zu senden und empfangene Nachrichten anzuzeigen. Die "Interface" -Klasse enthält eine Konstruktor-Methode, die eine Chat-Server-Instanz und einen Benutzernamen als Eingabe erhält. Beim Aufruf des Konstruktors wird

zunächst eine Willkommensnachricht an den Chat-Server gesendet, die den Benutzernamen enthält. Anschließend wird die Benutzeroberfläche erstellt, die aus einem Textfeld zum Eingeben von Nachrichten, einem "Senden"-Button und einem Bereich zum Anzeigen von Nachrichten besteht.

Wenn der "Senden"-Button gedrückt wird, wird der eingegebene Text an den Chat-Server gesendet und die Texteingabe wird gelöscht. Die "AddMessage" -Methode wird verwendet, um empfangene Nachrichten zum Anzeigen auf der Benutzeroberfläche hinzuzufügen. Die "CloseAll" -Methode wird aufgerufen, um die Benutzeroberfläche zu schließen, wenn der Chat-Client geschlossen wird.