Übung – Multichat

Kontext

In dieser Übung soll ein eigentlich vollständiges, jedoch schlecht umgesetztes, fehlerhaftes und mangelhaft dokumentiertes Programm analysiert, die essenziellen Funktionen und Abläufe dokumentiert und ein Refactoring durchgeführt werden. Ziel ist eine gute Klassenstruktur und sauberen gut dokumentierten Code zu erhalten und die enthaltenen Fehler zu beseitigen.

In dieser Übung arbeiten Sie in Zweiergruppen. Pro Klasse ist eine Dreiergruppe möglich.

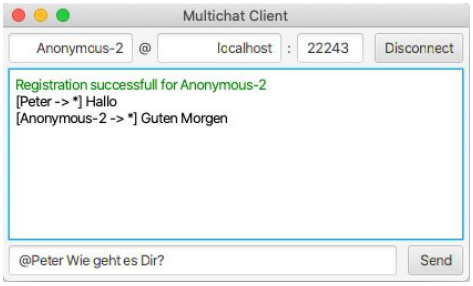
Einbettung in die Lernziele von PM2

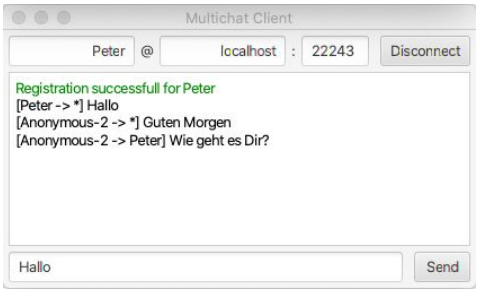
Die Übung trägt zur Erreichung der folgenden Lernzielen von PM2 bei:

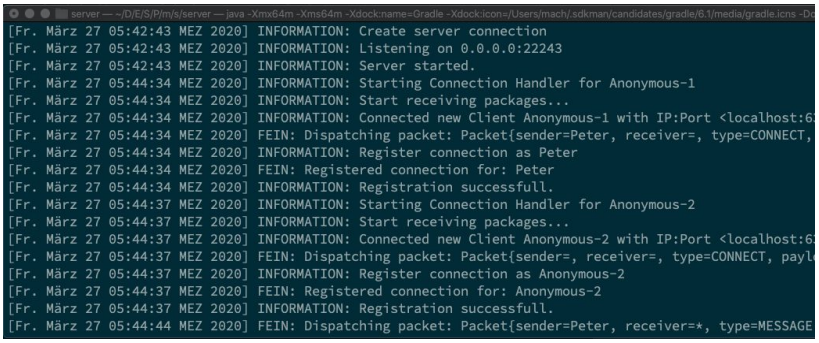
- Die Studierenden können ein bestehendes Programm strukturell und funktionell analysieren, dokumentieren und optimieren.
- Sie arbeiten in einem Team aktiv und zielführend zusammen und übernehmen dabei Verantwortung für die Erarbeitung des gemeinsamen Projektes wie auch für den Lernfortschritt aller Teammitglieder.

Multichat

Die Anwendung Multichat implementiert einen Chat-Dienst, in welchem mehrere Benutzer einfache Textmeldungen austauschen können. Sie besteht aus einem Serverteil, der als Dienst (ohne GUI, Log-Meldungen in Konsole) gestartet werden muss und einem JavaFX basierten Client, den mehrere Anwender starten, sich mit dem Server verbinden und Nachrichten austauschen können.







Beim Verbinden mit dem Server kann der Benutzer eingeben, wie er heissen will (z.B. Peter). Der Name muss eindeutig sein. Falls man keinen Namen angibt, erstellt der Server einen eindeutigen Namen. Meldungen werden an den Server geschickt, der sie an alle verbundenen Clients weiterschickt (broadcast). Wenn eine Meldung mit einem @Benutzername beginnt (z.B. @Peter) wird diese vom Server nur an den entsprechenden Benutzer weitergeleitet (direct).

Auftrag

Analysieren und Überarbeiten Sie die vorhandene Anwendung, so dass:

- die geforderte Funktion fehlerfrei erfüllt wird. Das heisst, alle Clients korrekt bedient werden und keine illegalen Zustände auftreten können.
- eine saubere Modul-/Klassenstruktur umgesetzt ist. Das heisst: passende Entitäten werden verwendet, Vererbung einsetzen; wo sinnvoll ist, lose Koppelung zwischen den Komponenten; Einhalten von bekannten Pattern (z.B. MVC), saubere Projektstruktur, …
- Clean Code eingehalten wird und die Klassen gut dokumentiert sind (Javadoc)

Dokumentieren Sie zudem in einem kurzen Bericht die Analyse und das Resultat ihrer Arbeit.

Anforderungen

Anwendung:

- Der Server muss beliebig viele Clients (soweit es die vorhandenen Systemressourcen zulassen) korrekt bedienen können.
- Wenn ein Client sich verbindet muss er korrekt eingebunden, wenn er sich abmeldet die entsprechenden Ressourcen wieder freigegeben werden.
- Beim Beenden des Servers werden die verbundenen Clients automatisch getrennt.
- Es ist nicht erforderlich, das Ihr Client und Server mit denjenigen anderer Gruppen kommunizieren kann. Sie sind also frei in der Strukturierung der Daten die übers Netzwerk versendet werden.
- Das grundsätzliche Protokoll zwischen Client und Server (Ablauf der Kommunikation) soll jedoch beibehalten werden. Das heisst, die Reihenfolge und Art der Nachrichten für einen bestimmten Vorgang (z.B. für die Registrierung, Senden einer Meldung, Abmeldung, …) soll nicht verändert werden. Jedoch können sie die Umsetzung (Datentypen, Ablauf, Kontrolle, …) anpassen und verbessern.
- Der Client muss das MVC-Pattern korrekt umsetzen.
- Das Client-GUI muss verhindern, dass der Anwender unzulässige Operationen ausführen kann (z.B. senden von Nachrichten, wenn nicht verbunden).
- Beim Beenden des Clients muss er korrekt auf dem Server abgemeldet werden.
- Zusätzliche Features, die über die geforderte Funktionalität hinausgehen, werden nicht bewertet.
- Der Code aller Klassen muss ausreichend dokumentiert sein (Javadoc).
- Server und Client sollen sinnvolle Statusmeldungen auf der Konsole ausgeben.
  Tip: Verwenden Sie dazu die Java-Logger -Funktionalität (siehe in Hinweise zum ausgegebenen Code)
- Build-Tooling wird korrekt verwendet. Die Server und Client muss mit dem Befehl `'gradle run'` gestartet werden können.
- Die Projektstruktur muss sinnvoll sein und Dateien am richtigen Ort liegen.
- Ihr Projekt muss die im CleanCode-Handbuch definierten Regeln der Stufe L1, L2 und L3 erfüllen.

Bericht:

Der Bericht besteht aus zwei Teilen:

1. Resultat der Analyse
    - Dokumentation der aktuellen Struktur (Klassendiagramm)
    - Dokumentation des Protokolls zwischen Client und Server
    - Beschreibung der gefundenen funktionalen Fehler
    - Beschreibung der gefundenen strukturellen Probleme
2. Beschreibung ihrer Lösung
    - Dokumentation der neuen Struktur (Klassendiagramm, ggf. mit Erläuterung)
    - Darlegung wie die strukturellen Probleme besser umgesetzt und funktionalen Probleme gelöst wurden und warum die vorgeschlagene Lösung optimal ist.

- Der Umfang des Berichts soll im Bereich von 3 - 6 Seiten liegen. Da es sich um einen technischen Bericht handelt, können sie Fachsprache verwenden und müssen die Fachbegriffe nicht erläutern (sofern sie eindeutig sind).
- Verwenden Sie geeignete Grafiken. Zum Beispiel korrekte UML-Syntax für Klassendiagramme.
- Wählen Sie die Dokumentation des Protokolls zwischen Client und Server eine geeignete Darstellung, welche den Ablauf (Anfragen und Antworten) so eindeutig und klar wie möglich zeigt. Welche Anfragen und Antworten werden in welcher Reihenfolge übermittelt und welche Informationen (Datentypen und oder Klassen) beinhalten diese. Bewertet wird die Richtigkeit und Verständlichkeit und nicht das Einhalten einer bestimmten Notation.

Hinweise zum Vorgehen:

- Erstellen Sie ein GitHub-Repository für Ihre Teilgruppe in Ihrer Klassenorganisation (Schema: `Uebung-<LoginStudent1>-<LoginStudent2>`) und fügen Sie sich als Kollaboratoren hinzu.
- Laden Sie die ZIP-Datei mit dem Ursprungsprojekt herunter und checken Sie den Inhalt in das neu erstellte Repository ein.
- Tipps zur Analyse des Ursprungsprojekts:
    - Was funktioniert in der Ursprungsversion von Client und Server nicht? Welche funktionalen Fehler sind enthalten?
      Tipp: Der Client muss gleichzeitig Nachrichten Senden und Empfangen und der Server muss mehrere Clients gleichzeitig bedienen können.
    - Analysieren Sie die strukturellen Probleme? Was haben Server und Client gemeinsam, was ist unterschiedlich? Ist das Klassenmodell sinnvoll? Wo wäre der Einsatz von Vererbung sinnvoll? Ist MVC richtig umgesetzt? ...
- Beheben Sie die funktionalen Fehler, so dass die Anwendung grundsätzlich funktioniert.
- Überarbeiten Sie die Struktur und einigen Sie sich auf ein Klassenmodell.
- Transformieren Sie die Anwendung in die neue Struktur.
- Wir empfehlen Ihnen die GitHub features für die Koordination Arbeit zu nutzen.

Hinweise zum ausgegebenen Ursprungsprojekt:

Das Ursprungsprojekt ist bereits in drei Teilmodule (server, client, protocol) gegliedert und Gradle dafür vorkonfiguriert, dass es in die IDE importiert werden kann.

- **`server`**
  Dieses Modul beinhaltet den Code für den Server-Dienst. Hier finden Sie neben der Hauptanwendung `'Server'` auch den `'ServerConnectionHandler'`, welcher das Kommunikations-Protokoll auf Serverseite umsetzt.
- **`client`**
  Dieses Modul beinhaltet die JavaFX-Anwendung für den Client. Neben der Hauptanwendung `'Client'` enthält es auch die GUI-Komponenten für das Chat-Fenster (JavaFX-Application, zugehörige FXML-Datei und Controller-Klasse). Auch hier ist die Kommunikation mit dem Server in die Klasse `'ClientConnectionHandler'` ausgelagert.
- **`protocol`**
  Dieses Modul enthält die Komponenten, welche von den Modulen server und client gemeinsam genutzt werden können. Das beinhaltet vor allem die Hilfsklasse `'NetworkHandler'` in welcher die Netzwerk-Kommunikation abstrahiert ist. Mit Hilfe der statischen Methoden können Sie einerseits einen `NetworkServer` erstellen der auf einem definierten Port auf Anfragen wartet. Andererseits auch vom Client eine `NetworkConnection` (Verbindung/Session) zu diesem Server (Hostname/Port) öffnen. Über den dann erzeugten Kanal vom Typ `NetworkConnection` können bidirektional Java-Objekte gesendet werden.
  Die Klasse verwendet Generics, d.h. der Datentyp der zu übertragenden Java-Objektes, welches übermittelt werden kann, muss bei der Initialisierung angegeben werden (Generics). Studieren Sie dazu die ausführliche Javadoc Dokumentation der Klasse.
  Die Klassen in der Datei `NetworkHandler.java` (inkl. innere Klassen) soll nur verwendet und nicht verändern werden.

Ausgabe von Status-/Log-Meldungen

Für die Ausgabe von Status-Meldungen auf der Konsole sollten Sie statt `System.out.println` oder `System.err.println` die Java-Logger Funktionalität verwenden. Das hat den Vorteil, dass jede Meldung mit einem Schweregrad (Severity-Level) versehen und dann zentral festlegen können, ab welchem Level die Meldungen ausgegeben werden sollen.
Das heisst sie können auch Debug-Meldungen in den Code einbauen, die nur beim Debuggen ausgegeben werden und im produktiven Betrieb unterdrückt werden.

Die Level sind aufsteigend gegliedert:

- Debug-Meldungen: FINEST, FINER, FINE
- Status-Meldungen: CONFIG, INFO
- Fehler-Meldungen: WARNING, SEVERE

Ab welchem Level die Meldungen ausgegeben werden können Sie in der Datei log.properties definieren (z.B. wenn `zh.zhaw.pm2.multichat.level=FINE` gesetzt ist, werden alle Meldungen mit Level FINE, CONFIG, INFO, WARNING, SEVERE ausgegeben, jedoch nicht FINER und FINEST)

Nehmen Sie den Logger-Einsatz in den Klassen Server und Client als Beispiel und setzen Sie die Funktionalität im Rest der Anwendung entsprechend ein.

Abgabe

Die Abgabe erfolgt bis zum im Wochenplan definierten Unterrichtstag um 23:59.

- Der erarbeitete Code muss bis zur Deadline im Projektrepository auf GitHub hochgeladen sein, inklusive Bericht als PDF im Projekt-Root.
- Der letzte Commit im Master-Branch vor diesem Zeitpunkt wird als Abgabestand verwendet.

Bewertung

In die Bewertung fliessen die folgenden Kriterien ein:

Umsetzung (50%)

- Das Programm besitzt die geforderte Funktionalität. Insbesondere die parallele Verarbeitung und dass das GUI nur aktuell sinnvolle Aktionen zulässt.
- Die in den Anforderungen definierten Regeln zu Clean Code wurden eingehalten.
  Code ist sauber und umfassend dokumentiert.
  Ausnahmsweise werden Tests nicht bewertet und damit auch nicht verlangt.

Bericht (50%)

- Der Bericht ist vollständig, fachlich korrekt und nachvollziehbar
- Die Klassenstruktur ist sinnvoll, die Koppelung ist niedrig, die Aufteilung der Klassen in Module ist sinnvoll. Vererbung ist sinnvoll eingesetzt.

