BASIS:

Wir haben uns in unserem Team für ein Börsensystem entschieden. Das System implementiert verschiene Klassen die miteinander in Echtzeit kommunizieren wobei die Synchronisation und Konsistenz der Daten jederzeit gewährleistet ist. Daten werden in unserer Anwendung nicht persisten gespeichert und gehen nach Ende der Laufzeit verloren. Der grösste Teil der Netzwerkfunktionalität des Systems wird unter Verwendung von RMI implementiert, wir benutzen gleichzeitig einen UDP Server der die Klasse FeedMessage instanziiert und die Objekte dieser Klasse an die Klienten schickt. Durch diese Anwendung ist es Nutzern möglich Aufträge mithilfe von einem Broker zu erstellen die dann an der Börse gehandelt werden. Die Aktien in diesen Aufträgen können dann von anderen Klienten gekauft werden.

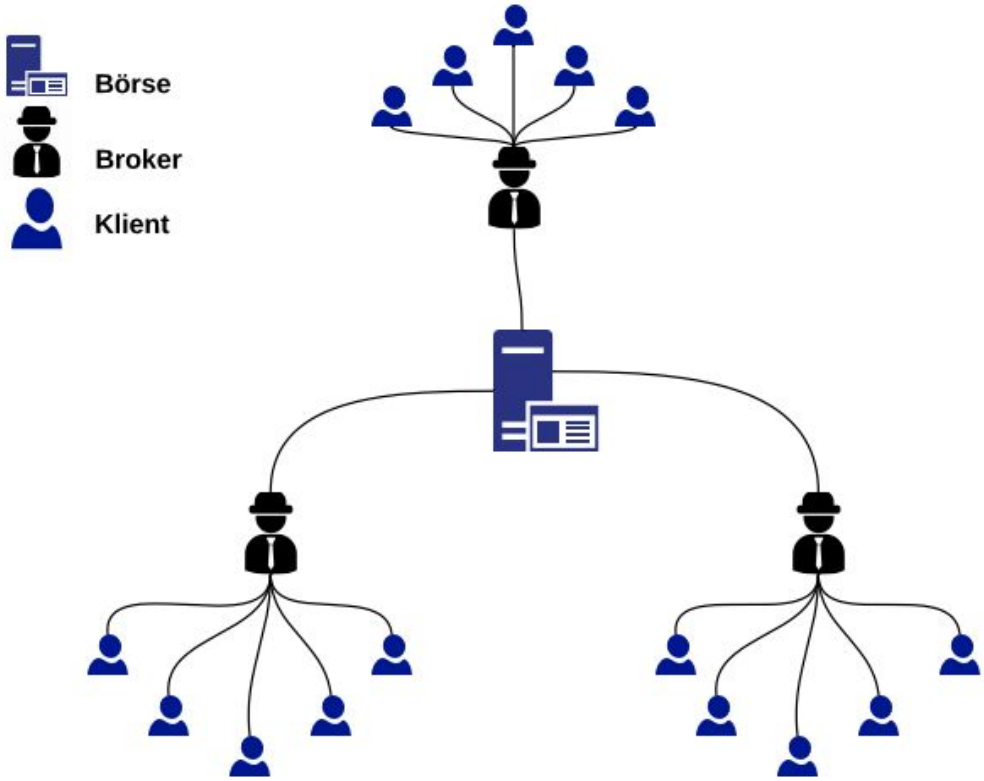
Die Architektur der verteilten Anwendung besteht aus verschiedenen Klassen. Im Zentrum der ganzen Börsenapplikation ist die Klasse Namens BörseServer. Diese Klasse agiert als TCP und UDP Server. Die Klasse BoerseServer implementiert drei Interfaces:

- **BoersePublic**: mittels diesem Interface können Klienten erforderliche Daten für die Kommunikation beziehen
- **BoerseAdmin**: mittels diesem Interface kann man die Börse verwalten
- **BoerseClient**: mittels diesem Interface können Broker Aufträge handeln

Broker sind eine Schnittstelle zwischen Klienten. Klienten möchten Aktien verkaufen und Kaufen, dies kann nur über einen Broker gemacht werden. Zu diesem Zweck haben wir eine Klasse BrokerServer erstellt, diese Klasse implementiert zwei Interfaces:

- **BrokerClient**:mittels diesem Interface können die Klienten Aufträge erstellen
- **BrokerAdmin**: mittels diesem Interface kann man einen Broker verwalten

Klienten erstellen einen Auftrag um Aktien zu handeln. Für Veranschaulichung der hierarchischen Architektur unserer verteilten Anwendung haben wir ein informelles Modell erstellt.



Klienten können mittels einem Graphical User Interface Aufträge erstellen und mit Hilfe von UDP die Daten in Echtzeit beziehen. Wodurch jede Änderung gleich sichtbar gemacht wird.

Aktien habe einen Preis der sich während der Laufzeit ändern kann. Der zulezt bezahlte Preis für eine Aktie ist der aktuelle Preis für den die Aktie gehandelt wird. Die Börse schickt Daten an alle Klienten mittels UDP Feed. Dazu implementieren wir die Klasse FeedMessage. Also sobald ein Klient einen Auftrag erstellt, sei es zum Verkauf oder Ankauf von Aktien werden alle Klienten darüber Informiert.

Der Broker agiert gleichzeitig als Klient und Server. Er bedient die Anfragen von Klienten, also seinen Kunden muss aber dazu auch die Börse kontaktieren um Informationen anzufordern.
Die Börse hat auch einen Status: geschlossen und aktiv. Dieser lässt sich von der Börsenadmin Konsole verwalten.

Der Kern unserer Anwendung besteht aus folgenden Klassen:

-Börse (zentrale Plattform)
-Klient (Privatperson)
-Broker (Schnittstelle zwischen Börse und Klient)
-Auftrag (repräsentiert einen Ver- oder Ankauf von Aktien/Emittenten)
-Emittent (Aktie)

GUI:

Des weiteren kommen wir zum Graphical User Interface.
Wir implementieren insgesamt drei Klassen die als Frontend für die Verwaltung von Daten gedacht sind.

1 .

Die umfangreichste Schnittstelle zum Enduser stellt das Klienten GUI da. Dieses befindet sich in der Klasse **ClientGUI**. Das Interface beschränkt sich auf ein Fenster.

Folgende Informationen kann der Kunde sehen:

-Den Konstostand (nur Bargeld)
-Liste der Aktien die er Besitzt inklusive der Mengen davon
-Die FeedDaten von der Börse die über UDP gestreamt werden um die neuesten Informationen zu erhalten
-eine Liste der aktiven Aufträge

Folgende Tasten sind auf dem Interface verfügbar:
-Emittent hinzufügen (Ein neuer Auftrag wird erstellt)
-Auftrag stornieren (der Kunde muss einen Auftrag aus der Liste ausählen)

2 .

Das zweite User Interface beschränkt sich auf die Börse, und es wird in der Klasse **BrokerGUI** implementiert. Dieses GUI ist einfacher gehalten. Die Schnittstelle beschränkt sich auf ein Fenster.

Folgende Funktionen werden vom Interface zur Verfügung gestellt:
-Liste von Klienten anzeigen Lassen
-Klienten hinzufügen
-Klienten sperren

3 .

Mit dem dritten User Interface kann man die Börse verwalten. Dieses wird in der Klasse **ServerGUI** implementiert. Dieses Fenster hat zwei Tabs:

<u>Diese Funktionen werden zur Verfügung gestellt:</u>

1. **Aktien:**

Aktie hinzufügen- damit eine Aktie gehandelt werden kann muss die Börse sie kennen. Jede Aktie wird über einen Ticker identifiziert, zum Beispiel = Apple:AAPL

Aktien anzeigen- eine Liste der gehandelten Aktien wird angezeigt

Aktien sperren- eine Aktie kann gesperrt werden und kann dadurch nicht mehr gehandelt werden

2 . Broker:

Parallel zu Aktien-Tab, Broker können angezeigt, hinzugefügt und gesperrt werden.

INFORMATIONSTRANSFER:

Für den Transfer von Marktdaten zwischen den einzelnen Komponenten (Börse, Klient, Broker) implementieren wir zwei Klassen.

1. **FeedMessage**
2. **FeedRequest**

Wir möchten alternativ die Elternklasse **Report** erstellen von der dann verschiedene Reportarten erben um die Reports der Situation anzupassen. Die wurde noch nicht detailliert besprochen und werden dementsprechend später implementiert.

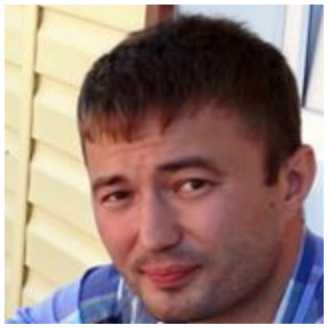
AUTOMATISATION:

Um das Testen zu automatisieren haben wir vor eine Klasse **TradingBot** zu implementieren. Diese Klasse würde als virtueller Klient agieren und Zufallswerte für den An- und Verkauf von Aktien benutzen, Dieser Teil wird noch besprochen.

ROLLENVERTEILUNG:

TEAM 104

Ayrat Zinatulin:
Projektleiter, konzeptioneller Entwurf und Implementation von Programmlogik. Zentraler Ansprechspunkt für Fragen zur Projektumsetzung und Applikationsarchitektur.



Daniil Rabizo:
Entwurf und Implementation von GUI für Börse und Broker, zuständig für die Weiterentwicklung und Optimierung des bestehenden UML KLassendiagramms.

Jakub Zvonek:
Entwurf und Implementation von GUI für den Klienten und die Erstellung von Projektdokumentation. Weiterhin zuständig für die Präsentation des IST-Standes und die teilweise Umsetzung der JUnit Tests.