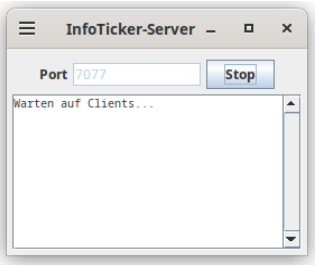
Übung Client/Server - Info(Ticker)Client



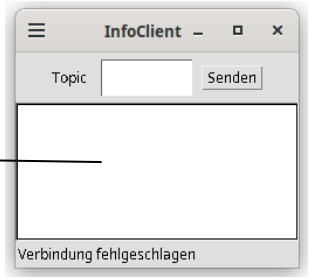
Ein bestehender **InfoTicker-Server**, stellt Clients Informationen zu verschiedenen Themen (Topics) bereit und aktualisiert diese bei Änderungen.

Starten Sie diesen Server durch auführen von **InfoTicker.jar**

Nach Eingabe des gewünschten (Server-)Ports und „Start“ wartet der InfoTicker-Server auf diesem Port auf eingehende Verbindungen.

Bislang liegt nur das Grundgerüst **InfoClient.java** für einen (GUI)-Client vor, bei dem die Kommunikation/Fachlogik noch fehlen.

AWT-**List**-Komponente
(Hilfe zu den möglichen Methoden siehe:
JavaDocs – API – package java.awt – class List)

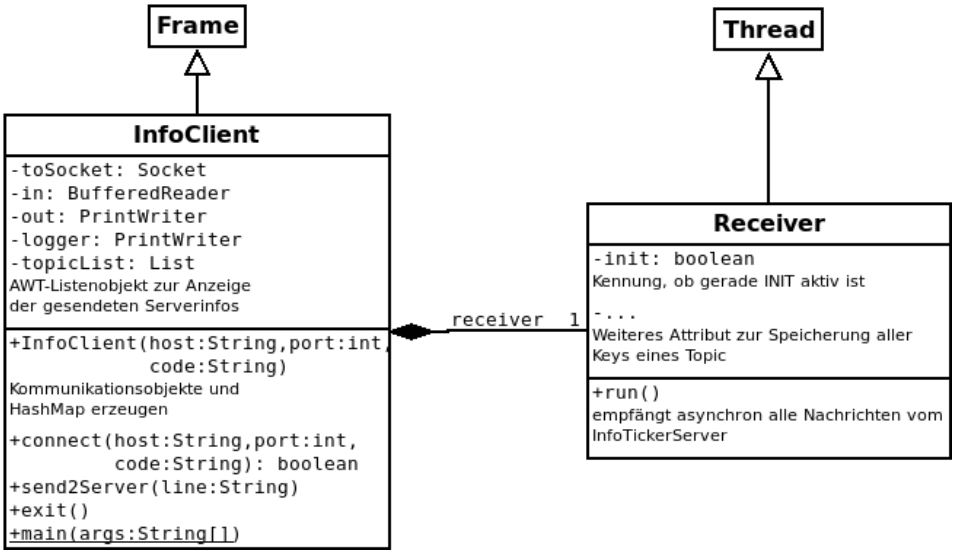


Der Server kann nach erfolgreichem Verbindungsaufbau über die folgenden (Text-) Kommandos angesprochen werden:

Richtung	**Kommando** (als Zeichenkette)	**Beschreibung**
Client => Server	<LOGIN>:name	Der Client muss nach Verbindungsaufbau ein Login mit dem Namen durchführen.
Server => Client	<END_LOGIN>:OK:text oder <END_LOGIN>:ERROR:text	Bestätigung des Login oder Fehlermeldung
Client => Server	<INIT>:topic	Der Client initialisiert beim Server das gewünschte Thema (topic). Zulässige Topics sind: ‚wetter’, ‚boerse’, ‚none’
Server => Client	<START_INIT> key1:value1 ... keyN:value1 <END_INIT>:OK oder <END_INIT>:ERROR:text	Der Server antwortet mit mehreren Informationszeilen, mit dem Aufbau key:value, die mit <START_INIT> eingeleitet und mit <END_INIT> abgeschlossen werden.
Server => Client	keyX:newvalue ...	Solange der Client auf kein anderes Topic mit einem neuen <INIT>:... umschaltet, sendet der Server in regelmässigen Abständen einzelne Wertaktualisierungen zu den bereits übermittelten keys.
Client => Server	<LOGOUT>	Der Server beendet die aktuelle Session und schließt seine Verbindungen. Es erfolgt keine Rückantwort an den Client.

Übung Client/Server - Info(Ticker)Client

Implementieren Sie die Funktionalität des InfoClient anhand des UML-Klassendiagramms und den folgenden Anforderungen. Ergänzen Sie die Attribute bei Bedarf.



Anforderungen:

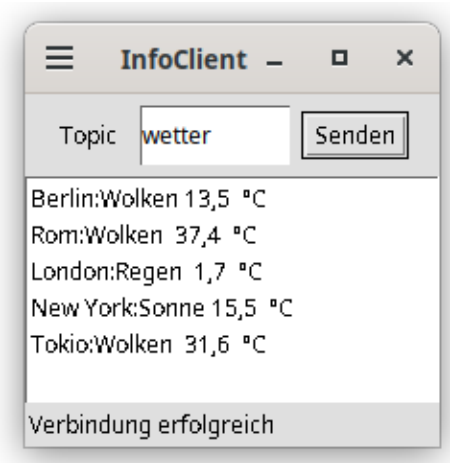
- connect: Baut eine Verbindung zum Server auf, erzeugt und initialisiert alle benötigten Objekte und speichert diese in den dafür vorgesehenen Attributen. Erzeugt einen PrintWriter zum Schreiben einer Log-Datei.
  Sendet das Login-Kommando mit dem Anmeldenamen an den Server und wertet die Antwort aus. Bei Erfolg wird der Receiver-Thread erzeugt und gestartet. Je nach Erfolg wird true oder false zurückgegeben.
- send2Server: Sendet die übergebene Zeile an den Server
- Receiver-Thread: Erstellen Sie eine **innere** Threadklasse, die nach der Initialisierung alle Rückantworten des Servers empfängt, verarbeitet und in der GUI anzeigt:

  ◦ Während einer INIT-Phase werden alle empfangenen Infos in die topicList hinzugefügt. Parallel dazu müssen Sie sich zu jedem Schlüssel den Indexwert in der List-Komponente merken.

  ◦ Aktualsierungsnachrichten nach <END_INIT> ersetzen das jeweilige Item in der Liste mit dem aktuellen neuen Wert.

  ◦ Beim Start einer neuen Init-Phase wird die topicList geleert und die gemerkten Schlüssel-Indizes verworfen, da ja in Folge andere Informationen empfangen werden müssen.

  ◦ Der Thread muss sich bei Lesefehlern beenden.



- exit: Beendet den receiver.Thread , sendet das Logout-Kommando an den Server schliesst alle lokale Ressourcen.

Optional:

- Alle vom Server empfangenen Daten sollen mit Hilfe des PrintWriter-loggers in eine LogDatei geschrieben werden. (Tipp: erstellen Sie eine eigene Lese-Methode, in der gleichzeitig „ge-loggt“ wird)
- Alle Logeinträge sind mit einem Zeitstempel versehen