Übungsblatt 4
Sockets (Forts.)

Aufgabe: Erstellung eines verteilten Chat-Systems mit Hilfe von Sockets

Es soll ein Client-Server-System zum gemeinsamen Chatten über das Internet vervollständigt werden.

Das Software-Design liegt bereits vor und wichtige Klassen sind implementiert (siehe Abbildungen 1 – 3 im Anhang). Es fehlen lediglich die „Komponenten“ auf Client- und Serverseite zur Kommunikation über Sockets.

Ihre Aufgabe ist es, die *aktiven* Klassen `ClientEndpoint`, `ServerEndpoint` und `Dispatcher` zu implementieren:

- `ClientEndpoint` Anmeldedaten und neue Nachrichten, die vom User über die Chat-Oberfläche eingegeben werden, werden an diesen Endpoint übergeben, der sie entsprechend nachfolgendem Protokoll in `OPEN`-, `PUBL`- oder `EXIT`-Nachrichten übersetzt und an seinen `ServerEndpoint` mit Hilfe seines Sockets sendet. Innerhalb der run()-Methode werden `SHOW`-, `ADMN`- oder `EXIT`-Nachrichten des Servers empfangen und entsprechend an die Chat-Oberfläche weitergeben bzw. umgesetzt.

- `ServerEndpoint` Jedem `Client-Endpoint` ist ein `ServerEndpoint` zugeordnet, der die Gegenseite des Sockets darstellt; innerhalb der `run()`-Methode wartet der Endpoint fortlaufend auf Nachrichten des Client-Endpoints und ruft anschließend die Methode des `ChatServer` auf, um die Nachrichten verteilen zu lassen; gleichzeitig stellt der `ServerEndpoint` Methoden zur Verfügung, um neue `SHOW`- oder `ADMN` -Nachrichten an seinen Client zu senden, die ihm vom `ChatServer` über die Methode `onMessage` zugestellt werden.

- `Dispatcher` Der `Dispatcher` implementiert auf Serverseite das `ServerSocket`, das auf einem angegebenen Port (Standardmäßig auf `1234`) auf neue Socketverbindungen horcht; für jede neu akzeptierte Socketverbindung wird ein Objekt der Klasse `ServerEndpoint` instanziiert

Die Kommunikation zwischen Ihren Endpunkten und den bestehenden Klassen `ChatServer` und `ClientGui` ist beidseitig über Interfaces entkoppelt (siehe UML-Klassendiagramm), das heißt, es können gegenseitig Methoden aufgerufen werden. Die Objekte des jeweils anderen werden über die Konstruktoren übergeben. Hierfür gibt es die beiden Konfigurationsklassen, über die der Server bzw. der Client auch gestartet wird, z. B.

```
$ java chat.server.Server <port>

$ java chat.client.Client
```

Für die Kommunikation über die Sockets wurde ein rudimentäres, String-basiertes Protokoll entwickelt, deren Befehle nachfolgender Tabelle zu entnehmen sind:

Protokoll-Befehl und Bedeutung	Gültig für Richtung Client→Server	Server→Client
`OPEN#<username>` Erste Nachricht des Clients, um sich mit dem gewählten Usernamen anzumelden	X	
`EXIT` Client oder Server möchte Verbindung beenden, Endpoints beenden sich	X	(X)
`PUBL#<message>` Client sendet Chat-Nachricht an Server, der an alle anderen Client verteilen wird (siehe `SHOW`); Username des Absenders ist dem `ServerEndpoint` bereits durch die `OPEN`-Nachricht bekannt	X	
`SHOW#<username>#<message>` Server sendet Chat-Nachricht an Client, damit dieser die Nachricht an die `ChatGui` weitergeben kann, `username` ist der Alias des Absenders		X
`ADMN#<message>` Administrative Nachricht an den Client zur Anzeige in der `ChatGui`, z. B. für Information, dass ein bestimmter User den Chat betreten oder verlassen hat		X

Nach Implementierung der drei Klassen muss eine Serverinstanz gestartet werden. Daraufhin können sich beliebig viele Clients mit diesem Server verbinden. Wenn alles klappt sollten Unterhaltungen und Verbindungen wie in folgendem Beispiel möglich sein:

Zusatzfragen:

- Überprüfen Sie Ihre Implementierungen, ob dafür Monitore (`synchronized` oder Locks) nötig sind; sind weitere Absicherungen notwendig?
- Ausbau zu einem skalierbaren Produktivsystem:
  Diskutieren Sie Möglichkeiten, wie dieses Chat-System skalierbar ausgebaut werden kann. Wären Thread-Pools hier möglich? Unter welchen Bedingungen ist das für dieses System möglich und/oder sinnvoll?

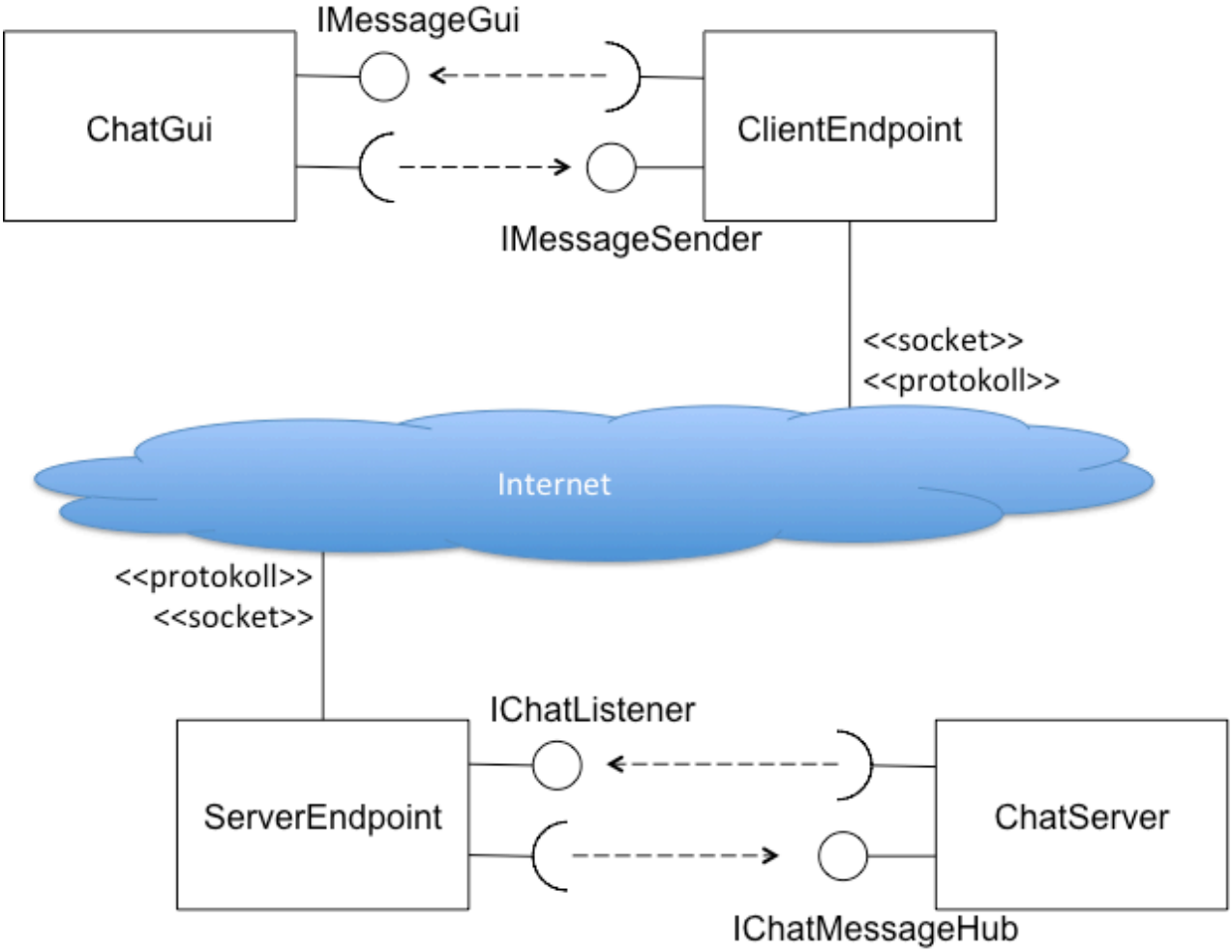

Abbildung 1: Komponentendiagramm mit Interfacebeziehungen

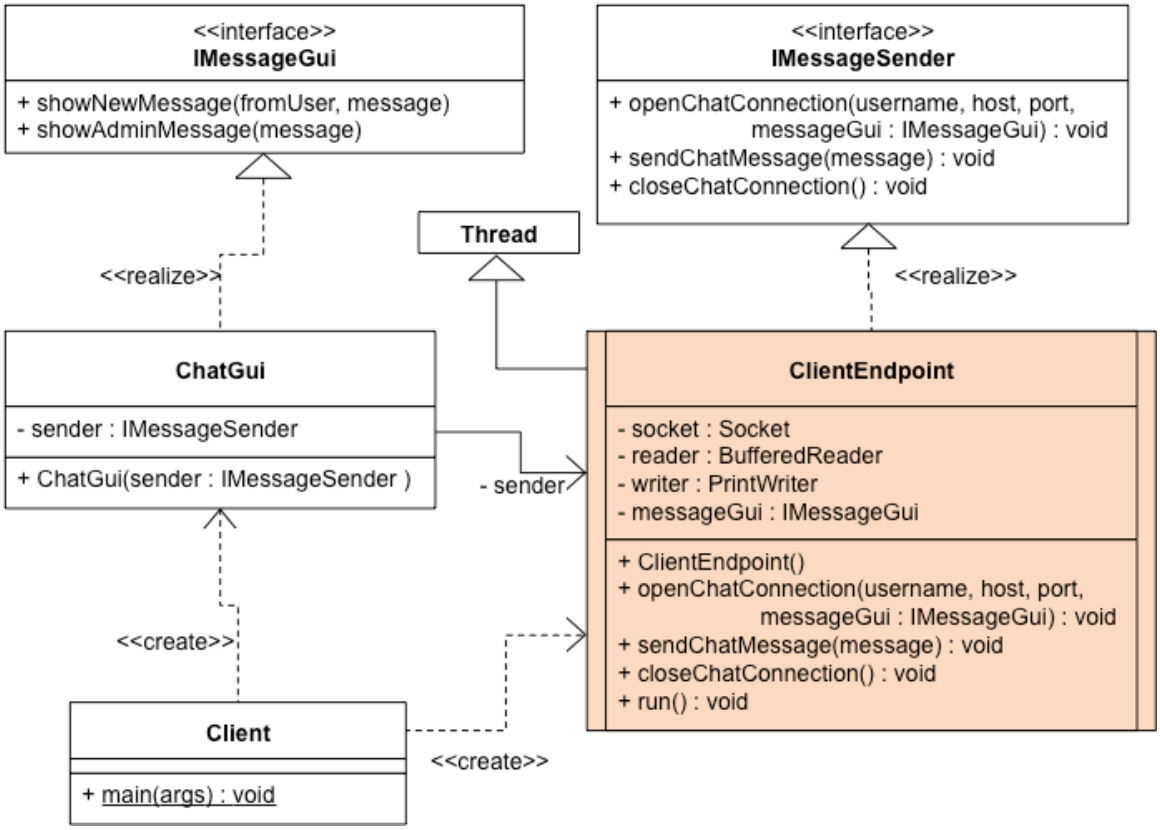

Abbildung 2: Klassen auf Clientseite

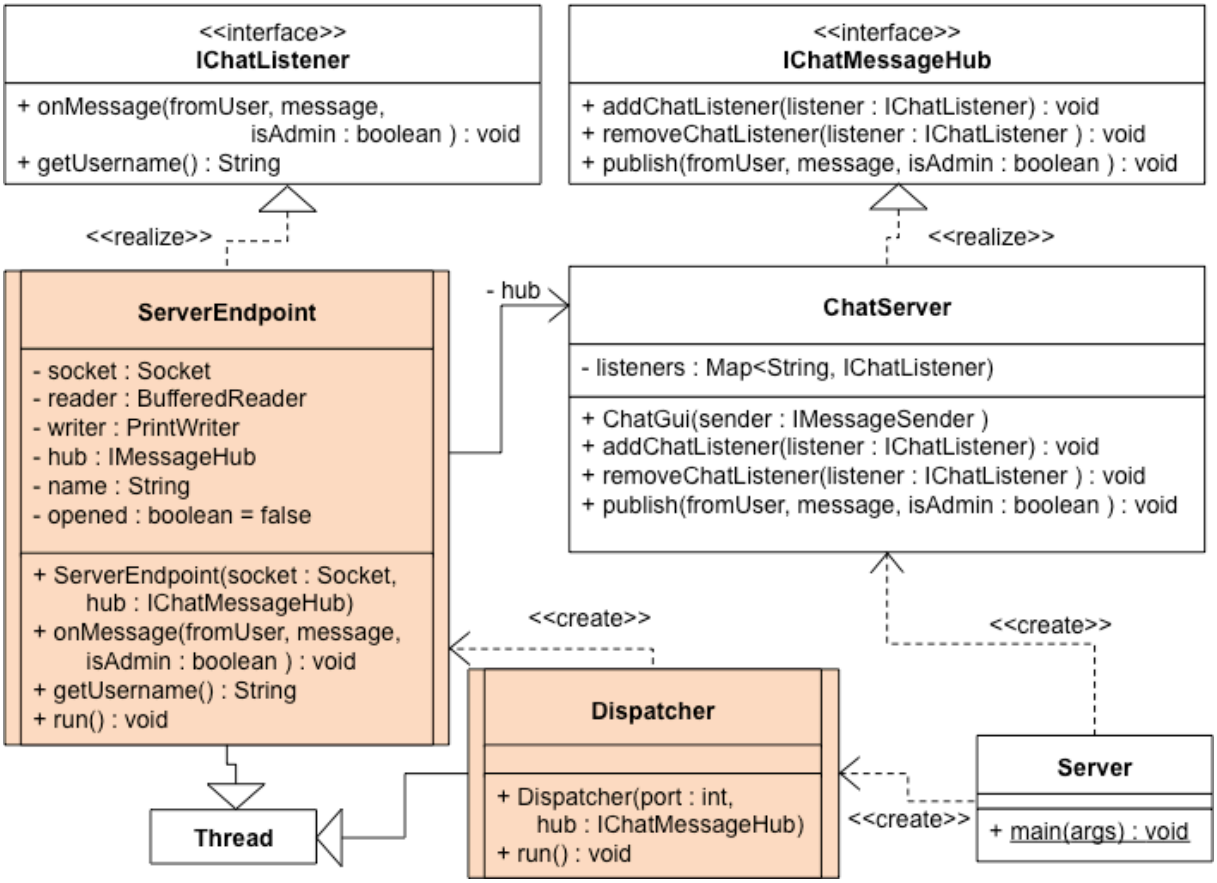

Abbildung 3: Klassen auf Serverseite