technische universität
dortmund

Fakultät für Informatik
Lehrstuhl 4

M. Sc.-Inf. Dimitri Scheftelowitsch
M. Sc.-Math. Alexander Frank

Wintersemester 2016/17

Rechnernetze und verteilte Systeme
Programmieraufgabe

Ausgabe: 12. Dezember, **Besprechung:** 10. Januar - 13. Januar, **Abgabe bis 4. Januar**

1 Problem

A. Nonymous kennt genug dunkle Ecken des Internets, und hat für das neue Jahr sich vorgenommen, eine eigene dunkle Ecke zu gestalten. Dazu möchte er ein *Textboard* mit einem selbst überlegten Protokoll einrichten. Textboards sind textbasierte Internet-Foren, die in Japan in der Frühzeit des Internets verbreitet waren. Die Funktionsweise eines Textboards besteht darus, dass eine Menge an Text-Nachrichten verwaltet wird und sowohl neue Nachrichten entgegen genommen als auch existierende Nachrichten übermittelt werden.

In dieser Aufgabe geht es darum, ein TCP-basiertes Protokoll für einen Textboard-Server zu implementieren. Der Server soll die oben beschriebene Funktionalität umsetzen. Zur Umsetzung soll ein Protokoll verwendet werden, welches im Folgenden genauer ausgeführt ist.

2 Protokoll – Allgemeines

Das Kommunikationsprotokoll zwischen Client und Server ist zeilenbasiert, d. h., eine Anfrage wird stets in Zeilen gegliedert. Da das Protokoll dem Austausch textueller Nachrichten dient, wird ein Format für eine Nachricht vorgegeben: Eine Nachricht besteht aus mehreren Zeilen, wobei die erste Zeile das *Thema* und die *Zeit* der Nachricht enthält. Die Zeit ist eine positive ganze Zahl, die die Anzahl der Sekunden seit dem 1. Januar 1970 UTC (*Unix-Timestamp*) bis zum Empfang der Nachricht angibt. Das Format einer Nachricht ist

```
Anzahl der Zeilen
Zeit Thema
<Nachrichtentext>
```

Zum Beispiel kann eine Nachricht wie folgt aussehen.

```
3
0 Frage an die Biologen
Hi, ich bin vor Kurzem auf ein Einhörnchen gestoßen.
Weiß jemand, ob es mit dem zentraleuropäischen Einhorn verwandt ist?
```

Eine Liste aus n Nachrichten wird wie folgt versendet.

```
Anzahl der Nachrichten
<Nachricht 1>
...
<Nachricht n>
```

Ein Beispiel für eine solche Liste wäre

```
2
2
360 Frage an die Biologen
Hi, ich habe auch ein Einhörnchen gesehen. Es hat mich böse angeschaut.
2
360 IRL-Treffen
Geht noch jemand zum Konzert der Nuclear Meltdowns am Samstag?
```

Als Zeit einer Nachricht gilt der Zeitpunkt ihres Empfangs auf dem Server. Falls ein Client eine Nachricht verschickt, darf an Stelle der Zeit eine beliebige Zahl stehen.

3 Protokoll – Client

Auf der Client-Seite können folgende Anfragen geschickt werden.

- `W <Zeitpunkt>` gibt eine Liste der Nachrichten zurück, die seit `<Zeitpunkt>` (Zeitpunkt in Sekunden seit 1. Januar 1970 UTC) geschrieben wurden.
- `P` schreibt eine Liste der Nachrichten, die in der nachfolgenden Zeile anfängt, auf das Textboard.
- `T <Thema>` gibt eine Liste der Nachrichten zurück, die als Thema `<Thema>` haben, in zeitlich absteigender Reihenfolge (neueste zuerst).
- `L <Anzahl>` gibt eine Liste von `<Anzahl>` Themen zurück, die zuletzt geändert wurden, in zeitlich absteigender Reihenfolge (neueste zuerst). Wenn `<Anzahl>` leer ist, werden alle Themen zurückgegeben. Das Format dafür ist an das Format einer Liste der Nachrichten angelehnt und sieht wie folgt aus.

  ```
  <Anzahl der Themen>
  <Zeit der letzten Nachricht in Thema 1> <Thema 1>
  <Zeit der letzten Nachricht in Thema 2> <Thema 2>
  ...
  <Zeit der letzten Nachricht in Thema N> <Thema N>
  ```

- `X` beendet die Sitzung. Nach einem `X` schließt der Server die Verbindung.

4 Protokoll – Server

Der Server kümmert sich um die korrekte Speicherung der Nachrichten und um die Verarbeitung der Anfragen. Wenn ein Client eine Nachricht veröffentlicht, schreibt der Server an alle gerade verbundenen Clients, sobald er ihre Anfragen beantwortet hat, eine Nachricht der Form

```
N <Anzahl neuer Nachrichten>
<Zeit der ersten neuen Nachricht> <Thema der ersten neuen Nachricht>
...
<Zeit der letzten neuen Nachricht> <Thema der letzten neuen Nachricht>
```

Der Server soll mehrere Verbindungen gleichzeitig verarbeiten können. Bei nicht korrekt geformten Anfragen soll eine Antwort der Form

```
E <Fehlermeldung>
```

verschickt werden. Eine Anfrage ist nicht korrekt geformt, wenn sie nicht zu den im Protokoll angegebenen Anfragen gehört oder das vorgegebene Format nicht einhält.

Unten stehen Beispiele für Anfragen und Antworten. Mit > ist die Kommunikation vom Client zum Server gekennzeichnet, mit < vom Server zum Client. Anmerkung: < und > und das nachfolgende Leerzeichen gehören NICHT zum Protokoll!

```
> P
> 1
> 2
> 0 Wartungshinweis
> Morgen ist das Textboard offline!
< N 1
< 1481101380 Wartungshinweis
```

```
> W 1481101380
< 1
< 2
< 1481101380 Wartungshinweis
< Morgen ist das Textboard offline!

> T Dieses Thema existiert nicht
< 0

> L
< 3
< 1481101380 Wartungshinweis
< 1481101164 Frage an die Biologen
< 1481101164 IRL-Treffen

> HELP
< E Befehl nicht erkannt...
```

5 Organisatorisches

Die Lösungen dürfen in Java, nach Absprache mit dem Übungsleiter auch in einer anderen Programmiersprache abgegeben werden.

5.1 Abgabe

Die Abgabe findet über AsSESS statt. Die Abgabe kann bis zum Abgabetermin beliebig oft durchgeführt werden. Es wird dann die zuletzt abgegebene Version gewertet.
Abzugeben sind die folgenden Dateien:

- ein Archiv (.zip) mit allen Quelltexten (Für Java: `*.java`-Dateien)
- readme.txt: Hinweise, wie sich Ihr Programm in der Kommandozeile (für Java: mit `javac`) kompilieren und starten lässt.
- nochmals einzeln alle zu Ihrer Applikation gehörenden `*.java`-Quelldateien

5.2 Vorgehen

Bevor Sie mit der Implementierung beginnen, machen Sie sich mit den nötigen Java-Klassen (bzw. den relevanten Teilen der jeweiligen Standardbibliothek) vertraut und überlegen Sie sich, wie Sie diese einsetzen können. Erstellen Sie dann ein Modell, das alle nötigen Elemente enthält, die für die Funktion der Applikation nötig sind. Erst nach der Modellierung beginnen Sie mit der Implementierung.

5.3 Kommentare und Dokumentation

Kommentieren Sie Ihren Quelltext ordentlich! Abgaben ohne ausreichende Kommentare werden mit Punktabzug bewertet. Zusätzlich sollten Sie die für die Bedienung der Applikation vorgesehenen Befehle/Kommandozeilenparameter dokumentieren, um das spätere Testen während der Korrektur zu ermöglichen. Dabei ist es auch zulässig, die Befehle beim Starten der Applikation zu erläutern.

5.4 Fehlerbehandlung

Im Programm sollten Exceptions (bzw. Fehlerzustände), die beispielsweise beim Verbindungsaufbau oder Versenden von Nachrichten entstehen können, verarbeitet werden. Eine unzureichende Fehlerbehandlung führt zu Punktabzug.

5.5 Bibliotheken

Zur Bearbeitung der Aufgabe sind ausschließlich die Java-Bibliotheken aus `java.*`/`javax.*` (für andere Programmiersprachen: jeweils nur die Standardbibliothek) zugelassen. Zusätzliche Bibliotheken und die Nutzung des Quelltexts anderer Fremdquellen ist daher nicht erlaubt.

6 Hinweise zur Implementierung

In diesem Kapitel werden einige Basisprogrammiertechniken für Java vorgestellt, die für diese Programmieraufgabe nützlich sind.

6.1 Sockets

Um zu vermeiden, dass jeder Programmierer, der über eine Netzwerkschnittstelle kommunizieren will, alle Protokolle, die dafür nötig sind (z.B. Ethernet, IP, TCP, UDP, etc.) selbst implementieren muss, gibt es ein Software-Abstraktion mit dem Namen *Socket*.

In Java werden zwei unterschiedliche Arten von Sockets unterschieden: *Sockets* und *ServerSockets*. Beide verhalten sich unterschiedlich.

Sockets sind diejenigen Sockets, über die tatsächlich Nachrichten verschickt werden können. Um eine Verbindung aufzubauen dient folgender Code:

```
Socket mySocket = new Socket();
mySocket.connect(
  new InetSocketAddress( "123.234.111.100", 12345 )
  );
```

Dadurch wird eine Verbindung mit dem Rechner aufgebaut, der die Adresse `123.234.111.100` hat. Zur Adresse gehört noch ein Port. Da ein Rechner mehrere Verbindungen verwalten kann, kann mit Hilfe des Ports auf der Gegenstelle das zugehörige Programm ausgewählt werden.

Um textbasiert zu kommunizieren, werden in Java Helferobjekte benötigt. Das ist zum Senden der *PrintWriter* und für den Empfang der *BufferedReader*. Die *read-Funktionen* des *BufferedReader*s blockieren. Das heißt, dass sie warten, bis zu lesende Inhalte verfügbar sind. Dazu wieder ein Beispiel:

```
Socket mySocket = new Socket();
mySocket.connect(
  new InetSocketAddress( "123.234.111.100", 12345 )
  );
PrintWriter out = new PrintWriter(
      mySocket.getOutputStream(), true
      );
BufferedReader in = new BufferedReader(
      new InputStreamReader(
        mySocket.getInputStream()
      ) );
```

ServerSockets dienen dazu auf ankommende Verbindungen zu warten. Ein ServerSocket muss an einen lokalen Port gebunden werden:

```
ServerSocket ssocket = new ServerSocket();
ssocket.bind( new InetSocketAddress( 12345 ) );
```

Die Methode *accept* ist eine blockierende Methode, die wartet, bis eine Verbindung aufgebaut wird. Wird eine Verbindung aufgebaut, so kehrt die Methode zurück und übergibt als Ergebnis einen neuen Socket. Mit Hilfe dieses neuen Sockets kann dann mit der Gegenstelle kommuniziert werden.

```
Socket client = ssocket.accept();
```

Da die Netzwerkschnittstelle ein Bauteil des Rechners ist, ist auch die Interaktion mit dem Betriebssystem nötig. Daher „lebt“ jeder Socket im Betriebssystem und es muss dem Betriebssystem mitgeteilt werden, dass man die Verbindung nicht länger benötigt. Alle Sockets – also auch ServerSockets – müssen daher geschlossen werden. Ansonsten bleiben sie auch nach Ende des Programms offen. Ein erneutes Öffnen ist dann nicht mehr möglich – auch dann nicht, wenn das Programm neu gestartet wird. Denken sie also immer an

```
mySocket.close();
```

beziehungsweise

```
ssocket.close();
```

Weitere Informationen über Sockets finden Sie in “Learning Java” Kapitel 13, oder weiterer Java-Literatur Ihrer Wahl.

6.2 Threads

Threads sind parallel ablaufende Programmteile. Sie werden benötigt, wenn das Programm beispielsweise an einem Socket lauscht, ob Nachrichten eingehen oder auf eingehende Verbindungen wartet. Während gewartet wird, kann nichts anderes getan werden. Dieses Verhalten wird Blockieren genannt. Ein Thread kann nun Abhilfe schaffen, indem ein solches Warten *parallel* ausgeführt wird. Einen Thread erzeugt man in Java wie folgt.

```
public class MyThread extends Thread
{
  @Override
  public void run{
    /* do something useful */
  }
}
```

Die Methode run() ist dabei die “main”-Methode des Threads. Ansonsten verhalten sich Threads wie ganz normale Klassen. Der folgende Codeschnipsel erzeugt einen Thread und startet ihn.

```
public class Program
{
  public static void main( String[] args ){
    MyThread t;
    t = new MyThread();
    t.start();
  }
}
```

Beachten Sie, dass ein Thread gestartet werden muss, damit er mit seiner Arbeit beginnt. Eine weitere wichtige Erweiterung von “MyThread” ist das Einfügen einer Hauptschleife.

```
public class MyThread extends Thread
{

  private boolean _terminate;

  MyThread()
  {
    _terminate = false;
  }

  public void terminate()
  {
    _terminate = true;
  }

  @Override
  public void run
  {
    while ( !_terminate )
    {
      /* do something useful */
    }
  }
}
```

Wichtig ist es zu beachten, dass der Zugriff auf den Thread aus einem anderen Thread heraus dazu führen kann, dass Daten inkonsistent sind. Beispielsweise könnte ein Thread einen String gerade ändern, während ein anderer Thread diese Variable gerade lesen möchte. In dem Fall könnte vom zweiten Thread ein unsinniges Wort gelesen werden. Dieses Verhalten nennt man *konkurrierend*, da zwei Threads um den Zugriff auf eine Variable konkurrieren. In den Java-Paketen java.util.concurrent.* finden Sie Datenstrukturen, die gegenüber solch konkurrierenden Zugriffen geschützt sind.

Weitere Informationen über Threads finden Sie in "Learning Java" Kapitel 9, oder weiterer Java-Literatur Ihrer Wahl.

Achtung!

Bitte nur selbst erstellte Lösungen abgeben. Mehrfache Abgaben derselben Lösung (Plagiate) werden mit 0 Punkten gewertet und führen zum Nichtbestehen der Studienleistung!

Die Abgabe der Aufgabe erfolgt über ASSESS unter https://ess.cs.tu-dortmund.de/ASSESS/.

Die Abgabe kann in 2er bis 3er Gruppen erfolgen. Einzelabgaben sind erlaubt.