Advanced IT - Testat 3

Aufgabenstellung

Die Aufgabenstellung bleibt entsprechend der vorherigen Aufgabe erhalten.

Umsetzung

In Addition zur bereits erfüllten Aufgabe mittels eines manuellen Clients, würde noch ein automatischer Client erstellt, welche nach Auswahl des Modus einige Testfälle automatisch durchspielt. Dies soll zeigen, dass das Programm entsprechend der Aufgabenstellung funktioniert. Innerhalb dieser Dokumentation soll nun der automatische Client genauer beleuchtet werden. Dafür werden die Dateien `test1.txt` und `test2.txt`, welche im Ordner "*Test Files*" gefunden werden können, vorausgesetzt. Der automatische Client funktioniert auch ohne diese, wird jedoch hauptsächlich Fehlermeldungen zurückgeben, da die automatisch erstellten Dateien leer sind, solange sie nicht durch die entsprechenden Kommandos beschrieben worden sind.

Beim Start eines Clients wird der Benutzer aufgefordert den Modus zu wählen, welchen er benutzen möchte. Wählt er den Modus 2, so startet der automatische Client und `case 2` der Switch-Verzweigung wird ausgeführt. Dadurch durchläuft der Client automatisch vorbereitete Testfälle / Methoden und gibt die dazugehörigen Ausgaben zurück. Diese können in den folgenden Beispielen gefunden werden. Am Ende benachrichtigt der Client den Benutzer, dass alle Tests vollständig durchgeführt wurden und beendet sich daraufhin selbst.

Client.java

```
case 2:


System.out.println("************************************************************
******************************");
    System.out.println("ATTENTION: You have chosen mode 2! The automatic
tests will start now...");

    socket.setSoTimeout(timeout);

    automaticParallelReadingFromSameFile();
    automaticParallelReadFromDifferentFiles();
    automaticParallelWritingInSameDocument();
    automaticParallelWritingInDifferentDocument();
    automaticParallelReadingAndWritingInSameDocument();
    automaticParallelReadAndWriteInDifferentFiles();
    automaticReadFromNonExistentFile();
    automaticWriteInNonExistentFile();
    automaticReadNonExistentLine();
    automaticWriteNonExistentLine();
    automaticOverwriteExistingLine();
    automaticIncompleteReadCommand();
    automaticIncompleteWriteCommand();
    automaticImproperReadCommand();
    automaticImproperWriteCommand();
    automaticReadNegativeLine();
    automaticWriteNegativeLine();
```

```
    automaticUnknownCommand();
    automaticMoreRequestsThanWorker();


System.out.println("************************************************************
*****************************");
    System.out.println("SUCCESS: All tests have passed! Exiting the
client...");
    break;
```

Der Server wird dabei ganz normal, wie auch beim manuellen Client gestartet.

Server.java

```
SUCCESS: Server was startet on port 5999!
SUCCESS: Worker 1 was started!
ATTENTION: Worker 1 is running...
SUCCESS: Worker 2 was started!
SUCCESS: Worker 3 was started!
ATTENTION: Worker 2 is running...
ATTENTION: Worker 3 is running...
SUCCESS: Worker 4 was started!
SUCCESS: Worker 5 was started!
ATTENTION: Worker 4 is running...
ATTENTION: Worker 5 is running...
```

Beispiele

Für die folgenden Beispiele werden die folgenden 2 Text-Dateien vorausgesetzt bzw. verwendet. Diese können im Ordner "*Test Files*" gefunden werden.

test1.txt

```
Hello!
I am a
test
file.
Nice to
meet you!



Lovely weather today,
isn't it? :)

...
```

test2.txt

```
Lorem Ipsum
Lorem Ipsum
Lorem Ipsum
Lorem Ipsum
Lorem Ipsum
```

Auf die Serverausgabe in diesen Beispielfällen wird verzichtet, da sie repetitiv sind und im Zuge des automatischen Ablaufs der Befehle eher schlecht nachzuvollziehen sind.

Alle folgenden Beispiele finden mittels **Client-Modus 2** statt:

```
Available mode:
1 - manual user input
2 - prepared automatic input
Please choose one of the modes above:
> 2
************************************************************************
**********
ATTENTION: You have chosen mode 2! The automatic tests will start now...
************************************************************************
**********
```

Hierbei wird nur mit einem Client gearbeitet, der jedoch die Anfragen so übergibt, als würden mehrere Clients diese parallel an den Server senden. Hierbei findet keine Verzpögerung durch menschliches Handeln statt.

Dabei wurde der Server entsprechend der folgenden Ausgabe gestartet:

```
SUCCESS: Server was startet on port 5999!
SUCCESS: Worker 1 was started!
ATTENTION: Worker 1 is running...
SUCCESS: Worker 2 was started!
SUCCESS: Worker 3 was started!
ATTENTION: Worker 2 is running...
ATTENTION: Worker 3 is running...
SUCCESS: Worker 4 was started!
SUCCESS: Worker 5 was started!
ATTENTION: Worker 4 is running...
ATTENTION: Worker 5 is running...
```

Da die Testfälle den Beispielen der "normalen" Dokumentation entsprechen, werde ich hier auf eine genauere Beschreibung / Erklärung verzichten. Genauere Details zu den einzelnen Testfällen können bei den Beispielen in `Dokumentation.md` oder `Dokumentation.pdf` gefunden werden.

Beispiel 1: Paralleles Lesen aus einer Datei

In diesem Beispiel soll mit 2 Clients parallel aus einer Datei ausgelesen werden.

Die Benutzereingabe/Clientausgabe für dieses Beispiel sieht wie folgt aus:

```
************************************************************************
**********
ATTENTION: Starting test for parallel reading from a file...
Send: <READ test1,1>
Send: <READ test1,2>
Send: <READ test1,3>
Send: <READ test1,4>
Send: <READ test1,5>
SUCCESS: Answer received: <test>
SUCCESS: Answer received: <I am a>
SUCCESS: Answer received: <Nice to>
SUCCESS: Answer received: <Hello!>
SUCCESS: Answer received: <file.>
************************************************************************
**********
```

Beispiel 2: Paralleles Lesen aus mehreren Dateien

In diesem Beispiel soll mit 2 Clients parallel aus mehreren Dateien ausgelesen werden.

Die Benutzereingabe/Clientausgabe für dieses Beispiel sieht wie folgt aus:

```
**************************************************************************
**********
ATTENTION: Starting test for reading from different files...
Send: <READ test1,1>
Send: <READ test2,3>
Send: <READ test1,5>
SUCCESS: Answer received: <Lorem Ipsum>
SUCCESS: Answer received: <Nice to>
SUCCESS: Answer received: <Hello!>
**************************************************************************
**********
```

Beispiel 3: Paralleles Schreiben in eine Datei

In diesem Beispiel soll mit 2 Clients parallel in eine Datei geschrieben werden.

Die Benutzereingabe/Clientausgabe für dieses Beispiel sieht wie folgt aus:

```
**************************************************************************
**********
ATTENTION: Starting test for parallel writing into the same file...
Send: <WRITE test1,3,I am tired...>
Send: <WRITE test1,5,Me too...>
SUCCESS: Answer received: <Overwritten to: Me too...>
SUCCESS: Answer received: <Overwritten to: I am tired...>
**************************************************************************
**********
```

Beispiel 4: Paralleles Schreiben in mehrere Dateien

In diesem Beispiel soll mit 2 Clients parallel in mehrere Dateien geschrieben werden.

Die Benutzereingabe/Clientausgabe für dieses Beispiel sieht wie folgt aus:

```
**************************************************************************
**********
ATTENTION: Starting test for parallel writing in different files...
Send: <WRITE test1,3,Random Data>
Send: <WRITE test2,5,Random Data in line 5>
SUCCESS: Answer received: <Overwritten to: Random Data in line 5>
SUCCESS: Answer received: <Overwritten to: Random Data>
**************************************************************************
**********
```

Beispiel 5: Paralleles Lesen und Schreiben in eine Datei

In diesem Beispiel soll mit 2 Clients parallel in mehrere Dateien geschrieben werden.

Die Benutzereingabe/Clientausgabe für dieses Beispiel sieht wie folgt aus:

```
**************************************************************************
**********
ATTENTION: Starting test for parallel reading and writing in the same
document...
Send: <READ test1,2>
Send: <WRITE test1,3,SURPRISE>
Send: <READ test1,3>
Send: <READ test1,4>
Send: <WRITE test1,5,SURPRISE in line 5>
Send: <READ test1,5>
SUCCESS: Answer received: <file.>
SUCCESS: Answer received: <Random Data>
SUCCESS: Answer received: <I am a>
SUCCESS: Answer received: <Overwritten to: SURPRISE in line 5>
SUCCESS: Answer received: <Overwritten to: SURPRISE>
SUCCESS: Answer received: <SURPRISE in line 5>
**************************************************************************
**********
```

Beispiel 6: Paralleles Lesen und Schreiben in mehrere Dateien

In diesem Beispiel soll mit 2 Clients aus mehreren Dateien gelesen und dabei gleichzeitig in mehrere Dateien geschrieben werden.

Die Benutzereingabe/Clientausgabe für dieses Beispiel sieht wie folgt aus:

```
**************************************************************************
**********
ATTENTION: Starting test for reading parallel from a file and writing into
another file...
Send: <READ test1,1>
Send: <WRITE test2,3,Trying to access in parallel...>
Send: <READ test1,5>
Send: <READ test2,3>
SUCCESS: Answer received: <Hello!>
SUCCESS: Answer received: <Lorem Ipsum>
SUCCESS: Answer received: <SURPRISE in line 5>
SUCCESS: Answer received: <Overwritten to: Trying to access in parallel...>
**************************************************************************
**********
```

Beispiel 7: Lesen aus einer nicht vorhandenen Datei

In diesem Beispiel soll aus einer nicht vorhandenen Datei gelesen werden.

Die Benutzereingabe/Clientausgabe für dieses Beispiel sieht wie folgt aus:

```
**************************************************************************
**********
ATTENTION: Starting test for reading a non existent file...
Send: <READ test3,50>
SUCCESS: Answer received: <ERROR: The corresponding file does not exists!>
**************************************************************************
**********
```

Beispiel 8: Schreiben in eine nicht vorhandene Datei

n diesem Beispiel soll in eine noch nicht vorhandene Datei geschrieben werden.

Die Benutzereingabe/Clientausgabe für dieses Beispiel sieht wie folgt aus:

```
**************************************************************************
**********
ATTENTION: Starting test for writing in a non existent file...
Send: <WRITE test100,1,Will probably work...>
SUCCESS: Answer received: <Overwritten to: Will probably work...>
**************************************************************************
**********
```

Beispiel 9: Lesen einer nicht vorhandenen Zeile

In diesem Beispiel soll eine noch nicht vorhandene Zeile gelesen werden.

Die Benutzereingabe/Clientausgabe für dieses Beispiel sieht wie folgt aus:

```
**************************************************************************
**********
ATTENTION: Starting test for reading a non existent line...
Send: <READ test1,50>
Send: <READ test2,30>
SUCCESS: Answer received: <ERROR: READ failed - line 50 could not be found
in file>
SUCCESS: Answer received: <ERROR: READ failed - line 30 could not be found
in file>
**************************************************************************
**********
```

Beispiel 10: Beschreiben einer nicht vorhandene Zeile

In diesem Beispiel soll eine noch nicht vorhandene Zeile einer Datei beschrieben werden.

Die Benutzereingabe/Clientausgabe für dieses Beispiel sieht wie folgt aus:

```
**************************************************************************
**********
ATTENTION: Starting test for writing in a non existent line...
Send: <WRITE test1,25,I am the last line>
Send: <WRITE test2,10,I am the last line>
SUCCESS: Answer received: <Overwritten to: I am the last line>
SUCCESS: Answer received: <Overwritten to: I am the last line>
**************************************************************************
**********
```

Beispiel 11: Überschreiben einer bereits vorhandenen Zeile

In diesem Beispiel soll eine bereits vorhandene Zeile einer Datei überschrieben werden.

Die Benutzereingabe/Clientausgabe für dieses Beispiel sieht wie folgt aus:

```
**************************************************************************
**********
ATTENTION: Starting test to overwrite an already existing line...
Send: <WRITE test1,1,BYE!>
```

```
Send: <WRITE test2,1,HELLO!>
SUCCESS: Answer received: <Overwritten to: BYE!>
SUCCESS: Answer received: <Overwritten to: HELLO!>
**********************************************************************
**********
```

Beispiel 12: Unvollständiger Lese-Befehl

In diesem Beispiel soll getestet werden, wie das Programm mit einem unvollständigen Lese-Befehl umgeht. In diesem Fall wird kein Dateiname mitgegeben, aus welchem gelesen werden soll.

Die Benutzereingabe/Clientausgabe für dieses Beispiel sieht wie folgt aus:

```
**********************************************************************
**********
ATTENTION: Starting test with incomplete read command...
Send: <READ ,1>
SUCCESS: Answer received: <ERROR: Invalid command. Please enter a valid
filename.>
**********************************************************************
**********
```

Beispiel 13: Unvollständiger Schreib-Befehl

In diesem Beispiel soll getestet werden, wie das Programm mit einem unvollständigen Schreib-Befehl umgeht. In diesem Fall wird kein Dateiname mitgegeben, in welche geschrieben werden soll.

Die Benutzereingabe/Clientausgabe für dieses Beispiel sieht wie folgt aus:

```
**********************************************************************
**********
ATTENTION: Starting test with incomplete write command...
Send: <WRITE ,5,TEST>
SUCCESS: Answer received: <ERROR: Invalid command. Please enter a valid
filename.>
**********************************************************************
**********
```

Beispiel 14: Unzulässiger Lese-Befehl

In diesem Beispiel soll getestet werden, wie das Programm mit einem unzulässigen Lese-Befehl umgeht. In diesem Fall wird keine Integer als Zeilennummer angegeben.

Die Benutzereingabe/Clientausgabe für dieses Beispiel sieht wie folgt aus:

```
**********************************************************************
**********
ATTENTION: Starting test with improper read command...
Send: <READ test1, >
SUCCESS: Answer received: <ERROR: Bad line number input. Line number has to
be an integer number greater than 0.>
**********************************************************************
**********
```

Beispiel 15: Unzulässiger Schreib-Befehl

In diesem Beispiel soll getestet werden, wie das Programm mit einem unzulässigen Schreib-Befehl umgeht. In diesem Fall wird keine Integer als Zeilennummer angegeben.

Die Benutzereingabe/Clientausgabe für dieses Beispiel sieht wie folgt aus:

```
**************************************************************************
**********
ATTENTION: Starting test with improper write command...
Send: <WRITE test2, ,TEST2>
SUCCESS: Answer received: <ERROR: Bad line number input. Line number has to
be an integer number greater than 0.>
**************************************************************************
**********
```

Beispiel 16: Lesen einer negativen Zeilennummer

In diesem Beispiel soll aus einer negative Zeilennummer gelesen werden.

Die Benutzereingabe/Clientausgabe für dieses Beispiel sieht wie folgt aus:

```
**************************************************************************
**********
ATTENTION: Starting test to read a negative line number...
Send: <READ test1,-1>
Send: <READ test2,-100>
SUCCESS: Answer received: <ERROR: Bad line number input. Please choose a
line number greater than 0.>
SUCCESS: Answer received: <ERROR: Bad line number input. Please choose a
line number greater than 0.>
**************************************************************************
**********
```

Beispiel 17: Beschreiben einer negativen Zeilennummer

In diesem Beispiel soll in eine negative Zeilennummer geschrieben werden.

Die Benutzereingabe/Clientausgabe für dieses Beispiel sieht wie folgt aus:

```
**************************************************************************
**********
ATTENTION: Starting test to write in a negative line number...
Send: <WRITE test1,-10,Will not work...>
Send: <WRITE test2,-5,Will not work either...>
SUCCESS: Answer received: <ERROR: Bad line number input. Please choose a
line number greater than 0.>
SUCCESS: Answer received: <ERROR: Bad line number input. Please choose a
line number greater than 0.>
**************************************************************************
**********
```

Beispiel 18: Unbekannter Befehl

In diesem Beispiel soll getestet werden, wie das Programm mit einem unbekannten Befehl umgeht.

Die Benutzereingabe/Clientausgabe für dieses Beispiel sieht wie folgt aus:

```
**************************************************************************
**********
ATTENTION: Starting test with unknown command...
Send: <Go to sleep already!>
SUCCESS: Answer received: <ERROR: The given command is unknown!>
**************************************************************************
**********
```

Beispiel 19: Mehr Befehle als Worker

In diesem Beispiel soll getestet werden, was passiert, wenn man mehr Befehle als Worker hat.

Die Benutzereingabe/Clientausgabe für dieses Beispiel sieht wie folgt aus:

```
**************************************************************************
**********
ATTENTION: Starting test with more requests than worker...
Send: <READ test1,1>
Send: <READ test1,2>
Send: <READ test1,3>
Send: <READ test1,4>
Send: <READ test1,5>
Send: <READ test1,6>
Send: <READ test1,7>
Send: <READ test1,8>
Send: <READ test1,9>
Send: <READ test1,10>
Send: <READ test1,11>
Send: <READ test1,12>
Send: <READ test1,13>
SUCCESS: Answer received: <BYE!>
SUCCESS: Answer received: <SURPRISE in line 5>
SUCCESS: Answer received: <file.>
SUCCESS: Answer received: <SURPRISE>
SUCCESS: Answer received: <I am a>
SUCCESS: Answer received: <>
SUCCESS: Answer received: <Lovely weather today,>
SUCCESS: Answer received: <>
SUCCESS: Answer received: <>
SUCCESS: Answer received: <meet you!>
SUCCESS: Answer received: <>
SUCCESS: Answer received: <isn't it? :)>
SUCCESS: Answer received: <...>
**************************************************************************
**********
SUCCESS: All tests have passed! Exiting the client...

Process finished with exit code 0
```

Beispiel 20: Befehl ohne gestarteten Server

In diesem Beispiel soll mit getestet werden, was passiert, wenn kein Server gestartet ist und somit der Client keine Antwort erhält. Dieses Beispiel ist in den automatischen Testfällen nicht implementiert. Sollte jedoch versucht werden diese ohne Server auszuführen, so tritt dieser Fall ein.

Die Benutzereingabe/Clientausgabe für dieses Beispiel sieht wie folgt aus:

```
Available mode:
1 - manual user input
2 - prepared automatic input
Please choose one of the modes above:
> 2
**************************************************************************
**********
ATTENTION: You have chosen mode 2! The automatic tests will start now...
**************************************************************************
**********
ATTENTION: Starting test for reading parallel from a file...
Send: <READ test1,1>
Send: <READ test1,2>
Send: <READ test1,3>
Send: <READ test1,4>
Send: <READ test1,5>
ERROR: java.net.SocketTimeoutException: Receive timed out
No connection to server available. The client will be closed now...
```

Gesamtauswertung

Die passenden Ausgaben bzw. Fehlerausgaben zeigen, dass das Programm mit allen Eventualitäten klarkommt und somit die Aufgabe entsprechend der Anforderungen erfüllt wurde. Die Testdateien sollten am Ende des Durchlaufs wie folgt aussehen:

test1.txt

```
BYE!
I am a
SURPRISE
file.
SURPRISE in line 5
meet you!



Lovely weather today,
isn't it? :)

...









I am the last line
```

test2.txt

```
HELLO!
```

```
Lorem Ipsum
Trying to access in parallel...
Lorem Ipsum
Random Data in line 5



I am the last line
```