Testdokumentation - Testataufgabe 3

Allgemeines

1. Das Programm erwartet einen Ordner `Files` auf dem Desktop. Die angefragten Dateien sollten vor der Nutzung angelegt werden.

2. Starten des Servers:

   ```
   java Dispatcher
   ```

3. Starten des dialogbasierten Clients:

   ```
   java Client
   ```

4. Starten des Clients für vordefinierte Testfälle:

   ```
   java DebugClient
   ```

   Der DebugClient zeigt keine Antworten des Servers. Diese Anfragen dienen lediglich der Demonstration der serverseitigen Synchronisation und Prioritäten. Die eigentliche "Funktionalität" der Anwendung wird über Testfälle des dialogbasierten Clients demonstriert. Deshalb werden sie bei den automatisierten Testfällen nach erfolgreicher Synchronisation der Anfragen als gegeben angenommen.

Testfällt mit diaglogbasiertem Client

Start des Servers (mit 5 Workern)

```
java Dispatcher
```

```
Dispatcher running on port 5999
Worker awaiting request
Worker awaiting request
Worker awaiting request
Worker awaiting request
Worker awaiting request
```

Start des Clients

```
java Client
```

```
======================
Please enter a Request
```

Fehlerhafte Nutzereingaben

Anfrage mit nicht unterstützter Methode:

```
======================
Please enter a Request
PRINT test
Server response:
BAD REQUEST: Only READ or WRITE allowed.
======================
```

READ-Anfrage ohne Dateiname und Zeile:

```
======================
Please enter a Request
READ
Server response:
BAD REQUEST: Invalid parameters.
======================
```

READ-Anfrage ohne Zeile:

```
======================
Please enter a Request
READ file.txt
Server response:
BAD REQUEST: Command READ takes 2 arguments.
======================
```

READ-Anfrage mit fehlerhafter Zeile:

```
======================
Please enter a Request
READ file.txt,a
Server response:
ILLEGAL LINE NUMBER
======================
```

READ-Anfrage mit fehlerhaftem Dateiname:

```
======================
Please enter a Request
READ file.txt,1
Server response:
FILE NOT FOUND
======================
```

WRITE-Anfrage ohne Zeileninhalt:

```
======================
Please enter a Request
WRITE file1.txt,1
Server response:
BAD REQUEST: Command WRITE takes 3 arguments.
======================
```

READ und WRITE über das Dateiende hinaus:

```
======================
Please enter a Request
READ file1.txt,10
Server response:
LINE NUMBER OUT OF BOUNDS
======================
```

```
======================
Please enter a Request
WRITE file1.txt,10,test
Server response:
LINE NUMBER OUT OF BOUNDS
======================
```

Erfolgreiche READ-Anfrage:

- Server

```
Worker awaiting request
Added Element to Queue. Size: 1
Removing Element from Queue. New size: 0
```

```
===Worker handling request===
Client request: <READ file1.txt,1>
Reading
Sending response: Hallo
===Worker finished request===

Worker awaiting request
```

- Client

```
======================
Please enter a Request
READ file1.txt,1
Server response:
Hallo
======================
```

Erfolgreiche WRITE-Anfrage

- Server

```
Added Element to Queue. Size: 1
Removing Element from Queue. New size: 0

===Worker handling request===
Client request: <WRITE file1.txt,1,Hallo Welt>
Writing
Sending response: OK
===Worker finished request===
```

- Client

```
======================
Please enter a Request
WRITE file1.txt,1,Hallo Welt
Server response:
OK
======================
```

- anschließendes READ

```
======================
Please enter a Request
READ file1.txt,1
Server response:
```

```
Hallo Welt
======================
```

Modifizieren anderer Zeilen und Dateien.

- Inhalt von `file2.txt`:

```
Ich
bin
eine
Datei.
```

- Auslesen von Zeile 3:

```
======================
Please enter a Request
READ file2.txt,3
Server response:
eine
======================
```

- modifizieren von Zeile 3:

```
======================
Please enter a Request
WRITE file2.txt,3,eine kurze
Server response:
OK
======================
```

- auslesen von Zeile 3:

```
======================
Please enter a Request
READ file2.txt,3
Server response:
eine kurze
======================
```

- Inhalt `file2.txt`:

```
Ich
bin
```

```
eine kurze
Datei.
```

Testfälle des automatisierten Clients

Die folgenden Testfälle dienen dazu, die Synchronisation sowie die Auftragswarteschlange zu demonstrieren.

Testfall 1

Dieser Testfall demonstriert die Schreiberpriorität bei aufeinander folgenden Anfragen zur selben Datei. Außerdem ist erkennbar, dass Schreibzugriffe sequenziell ausgeführt werden, während Lesezugriffe parallel erlaubt sind.

Anzahl Worker: 5

Ausgeführter Code:

```
private void testcase1() throws IOException {
    sendWriteFile1();
    sendReadFile1();
    sendReadFile1();
    sendReadFile1();
    sendWriteFile1();
    sendWriteFile1();
}
```

Protokoll des Servers:

```
Dispatcher: running on port 5999
        Worker 1: awaiting request
                Worker 2: awaiting request
                        Worker 3: awaiting request
                                Worker 4: awaiting request
                                        Worker 5: awaiting request
```

- Der erste Request wird von Worker 1 entgegengenommen. Worker 1 schreibt.

```
Dispatcher: added Element to Queue. Size: 1
        Worker 1: Removing Element from Queue. New size: 0
        Worker 1: handling request
        Worker 1: Client request: <WRITE file1.txt,1,Test>
        Worker 1: Writing
```

- Der zweite Request wird von Worker 2 entgegengenommen. Da Worker 1 schreibt, kann Worker 2 noch nicht mit dem Lesen beginnen.

```
Dispatcher: added Element to Queue. Size: 1
                Worker 2: Removing Element from Queue. New size: 0
                Worker 2: handling request
                Worker 2: Client request: <READ file1.txt,1>
```

- Der dritte Request wird von Worker 3 entgegengenommen. Da Worker 1 schreibt, kann Worker 3 noch nicht mit dem Lesen beginnen.

```
Dispatcher: added Element to Queue. Size: 1
                        Worker 3: Removing Element from Queue. New size: 0
                        Worker 3: handling request
                        Worker 3: Client request: <READ file1.txt,1>
```

- Der vierte Request wird von Worker 4 entgegengenommen. Da Worker 1 schreibt, kann Worker 4 noch nicht mit dem Lesen beginnen.

```
Dispatcher: added Element to Queue. Size: 1
                                Worker 4: Removing Element from Queue. New size: 0
                                Worker 4: handling request
                                Worker 4: Client request: <READ file1.txt,1>
```

- Der fünfte Request wird von Worker 5 entgegengenommen. Da Worker 1 schreibt, kann Worker 5 noch nicht mit dem Schreiben beginnen.

```
Dispatcher: added Element to Queue. Size: 1
                                        Worker 5: Removing Element from Queue. New
size: 0
                                        Worker 5: handling request
                                        Worker 5: Client request: <WRITE
file1.txt,1,Test>
```

- Der sechste Request wird in die Warteschlange eingefügt. Da alle Worker blockiert sind, wird der Request von keinem Worker entgegengenommen. Da Worker 1 schreibt, kann Worker 5 noch nicht mit dem Schreiben beginnen.

```
Dispatcher: added Element to Queue. Size: 1
```

- Worker 1 beendet das Schreiben und gibt die Monitorkontrolle ab. Aufgrund der Schreiberpriorität erhält Worker 5 die Monitorkontrolle. Worker 1 versendet die Antwort an den Client.

```

```

```
Worker 1: Ending write
                                Worker 5: Writing
Worker 1: Sending response: OK
Worker 1: finished request
```

- Worker 1 ist frei und nimmt sich erneut einen Auftrag aus der Queue. Die Queue ist nun leer. Da Worker 5 schreibt, muss Worker 1 warten.

```
Worker 1: awaiting request
Worker 1: Removing Element from Queue. New size: 0
Worker 1: handling request
Worker 1: Client request: <WRITE file1.txt,1,Test>
```

- Worker 5 beendet das Schreiben und gibt die Monitorkontrolle ab. Aufgrund der Schreiberpriorität erhält Worker 1 die Monitorkontrolle. Worker 5 versendet die Antwort an den Client.

```
                                Worker 5: Ending write
                                Worker 5: Sending response: OK
Worker 1: Writing
                                Worker 5: finished request
```

- Worker 5 kann wieder Aufträge entgegennehmen, jedoch ist die Queue leer. Worker 5 verweilt im Wartezustand.

```
                                Worker 5: awaiting request
```

- Worker 1 beendet das Schreiben und gibt die Monitorkontrolle ab. Da keine weiteren Schreibzugriffe warten, können nach einem `notifyAll()` alle Leserprozesse loslaufen:

```
Worker 1: Ending write
Worker 1: Sending response: OK
Worker 1: finished request
                        Worker 4: Reading
                Worker 3: Reading
        Worker 2: Reading
```

- Worker 1 wartet an der Auftragswarteschlange.

```
Worker 1: awaiting request
```

- Die Worker 2 - 4 beenden ihre Lesezugriffe und warten an der Queue.

```
                        Worker 3: Ending read
                Worker 2: Ending read
                Worker 2: Sending response: Test
                                Worker 4: Ending read
                Worker 2: finished request
                        Worker 3: Sending response: Test
                Worker 2: awaiting request
                                Worker 4: Sending response: Test
                        Worker 3: finished request
                                Worker 4: finished request
                        Worker 3: awaiting request
                                Worker 4: awaiting request
```

- Jetzt warten alle Worker wieder an der Queue.

Testfall 2

Dieser Testfall demonstriert das parallele Lesen auf verschiedenen Dateien.

```
private void testcase2() throws IOException {
    sendWriteFile1();
    sendReadFile1();
    sendReadFile1();
    sendReadFile1();
    sendWriteFile2();
    sendWriteFile2();
    sendReadFile2();
}
```

Protokoll des Servers

- Startroutine

```
Dispatcher: running on port 5999
                Worker 2: awaiting request
        Worker 1: awaiting request
                        Worker 3: awaiting request
                                Worker 4: awaiting request
                                        Worker 5: awaiting request
```

- Eingehender WRITE-Request für Datei 1. Worker 2 nimmt den Auftrag und erhält die Monitorkontrolle für Datei 1.

```
Dispatcher: added Element to Queue. Size: 1
                Worker 2: Removing Element from Queue. New size: 0
                Worker 2: handling request
                Worker 2: Client request: <WRITE file1.txt,1,Test>
                Worker 2: Writing
```

- Eingehender Leseauftrag für Datei 1. Worker 1 nimmt den Auftrag, muss aber aber auf Worker 2 warten.

```
Dispatcher: added Element to Queue. Size: 1
        Worker 1: Removing Element from Queue. New size: 0
        Worker 1: handling request
        Worker 1: Client request: <READ file1.txt,1>
```

- Eingehender Leseauftrag für Datei 1. Worker 3 nimmt den Auftrag, muss aber aber auf Worker 2 warten.

```
Dispatcher: added Element to Queue. Size: 1
                        Worker 3: Removing Element from Queue. New size: 0
                        Worker 3: handling request
                        Worker 3: Client request: <READ file1.txt,1>
```

- Eingehender Leseauftrag für Datei 1. Worker 4 nimmt den Auftrag, muss aber aber auf Worker 2 warten.

```
Dispatcher: added Element to Queue. Size: 1
                                Worker 4: Removing Element from Queue. New size: 0
                                Worker 4: handling request
                                Worker 4: Client request: <READ file1.txt,1>
```

- Eingehender Schreibauftrag für Datei 2. Da Datei 1 und Datei 2 parallel beschrieben werden können, kann der zugewiesene Worker 5 direkt mit dem Schreiben beginnen.

```
Dispatcher: added Element to Queue. Size: 1
                                        Worker 5: Removing Element from Queue. New
size: 0
                                        Worker 5: handling request
                                        Worker 5: Client request: <WRITE
file2.txt,1,Test>
                                        Worker 5: Writing
```

- Zwei weitere Aufträge gehen ein, können aber nicht bearbeitet werden, da keiner der Worker frei ist. Die Queue hat eine Länge von 2.

```
Dispatcher: added Element to Queue. Size: 1
Dispatcher: added Element to Queue. Size: 2
```

- Worker 2 beendet das Schreiben auf Datei 1. Daher können die Worker 1, 3 und 4 mit ihren Leseaufträgen beginnen. Währenddessen versendet Worker 2 seine Antwort und wartet wieder an der Queue.

```
                Worker 2: Ending write
                                Worker 4: Reading
        Worker 1: Reading
                         Worker 3: Reading
                Worker 2: Sending response: OK
                Worker 2: finished request
                Worker 2: awaiting request
```

- In der Queue liegen noch zwei Aufträge. Daher entnimmt Worker 2 direkt den ersten Auftrag - Schreiben auf Datei 2 - muss aber noch auf Worker 5 warten, der auf Datei 2 schreibt. Nachdem Worker 5 das Schreiben beendet hat, kann Worker 2 beginnen.

```
                Worker 2: Removing Element from Queue. New size: 1
                Worker 2: handling request
                Worker 2: Client request: <WRITE file2.txt,1,Test>
                                        Worker 5: Ending write
                                        Worker 5: Sending response: OK
                Worker 2: Writing
```

- Worker 5 versendet die Antwort des Schreibvorgangs auf Datei 2.

```
                                        Worker 5: finished request
                                        Worker 5: awaiting request
```

- Der wieder frei gewordene Worker 5 entnimmt den letzten Auftrag aus der Queue - Lesen aus Datei 2. Da Worker 2 aber noch auf Datei 2 schreibt, muss Worker 5 warten. Unabhängig davon beenden die Worker 1, 3, 4 ihre Lesevorgänge auf Datei 1.

```
                                        Worker 5: Removing Element from Queue. New
size: 0
                                        Worker 5: handling request
                                        Worker 5: Client request: <READ
file2.txt,1>
        Worker 1: Ending read
                         Worker 3: Ending read
                                 Worker 4: Ending read
```

```
                        Worker 3: Sending response: Test
        Worker 1: Sending response: Test
                        Worker 3: finished request
                                Worker 4: Sending response: Test
                        Worker 3: awaiting request
        Worker 1: finished request
                                Worker 4: finished request
        Worker 1: awaiting request
                                Worker 4: awaiting request
```

- Die Worker 1, 3, 4 haben ihre Antworten versendet und warten wieder an der Queue, diese ist jedoch mittlerweile leer. Nun beendet Worker 2 das Schreiben der Datei 2. Daher kann Worker 5 mit dem Lesen dieser Datei beginnen.

```
                Worker 2: Ending write
                Worker 2: Sending response: OK
                                        Worker 5: Reading
                Worker 2: finished request
                Worker 2: awaiting request
                                        Worker 5: Ending read
                                        Worker 5: Sending response: Test
                                        Worker 5: finished request
                                        Worker 5: awaiting request
```

- Alle Aufträge sind abgearbeitet und Worker 1 - 5 warten an der Queue.

Testfall 3

Dieser Testfall demonstriert das Einreihen von Aufträgen in die Warteschlange, falls mehr Anfragen eingehen, als Worker vorhanden sind.

```
private void testcase3() throws IOException {
    for (int i = 0; i < 10; i++)
        sendReadFile1();
}
```

- Startprotokoll

```
Dispatcher: running on port 5999
        Worker 1: awaiting request
                        Worker 3: awaiting request
                Worker 2: awaiting request
                                Worker 4: awaiting request
                                        Worker 5: awaiting request
```

- Die ersten fünf eingehende Aufträge können direkt von den Workern verarbeitet werden. Da es sich lediglich um Leseaufträge handelt, ist parallele Verarbeitung möglich.

```
Dispatcher: added Element to Queue. Size: 1
        Worker 1: Removing Element from Queue. New size: 0
        Worker 1: handling request
        Worker 1: Client request: <READ file1.txt,1>
        Worker 1: Reading
Dispatcher: added Element to Queue. Size: 1
                        Worker 3: Removing Element from Queue. New size: 0
                        Worker 3: handling request
                        Worker 3: Client request: <READ file1.txt,1>
                        Worker 3: Reading
Dispatcher: added Element to Queue. Size: 1
                Worker 2: Removing Element from Queue. New size: 0
                Worker 2: handling request
                Worker 2: Client request: <READ file1.txt,1>
                Worker 2: Reading
Dispatcher: added Element to Queue. Size: 1
                                Worker 4: Removing Element from Queue. New size: 0
                                Worker 4: handling request
                                Worker 4: Client request: <READ file1.txt,1>
                                Worker 4: Reading
Dispatcher: added Element to Queue. Size: 1
                                        Worker 5: Removing Element from Queue. New
size: 0
                                        Worker 5: handling request
                                        Worker 5: Client request: <READ
file1.txt,1>
                                        Worker 5: Reading
```

- Alle Worker sind belegt. Die verbleibenden 5 Aufträge werden in die Queue eingereiht.

```
Dispatcher: added Element to Queue. Size: 1
Dispatcher: added Element to Queue. Size: 2
Dispatcher: added Element to Queue. Size: 3
Dispatcher: added Element to Queue. Size: 4
Dispatcher: added Element to Queue. Size: 5
```

- Die Worker beenden nacheinander ihre Lesevorgänge. Nach dem Versenden der Antwort warten sie wieder an der Queue, können jedoch sofort weiterlaufen, da die Queue noch Aufträge enthält.

```
        Worker 1: Ending read
        Worker 1: Sending response: Test
        Worker 1: finished request
        Worker 1: awaiting request
        Worker 1: Removing Element from Queue. New size: 4
        Worker 1: handling request
```

```
        Worker 1: Client request: <READ file1.txt,1>
        Worker 1: Reading
                        Worker 3: Ending read
                        Worker 3: Sending response: Test
                        Worker 3: finished request
                        Worker 3: awaiting request
                        Worker 3: Removing Element from Queue. New size: 3
                        Worker 3: handling request
                        Worker 3: Client request: <READ file1.txt,1>
                        Worker 3: Reading
                Worker 2: Ending read
                Worker 2: Sending response: Test
                Worker 2: finished request
                Worker 2: awaiting request
                Worker 2: Removing Element from Queue. New size: 2
                Worker 2: handling request
                Worker 2: Client request: <READ file1.txt,1>
                Worker 2: Reading
                                Worker 4: Ending read
                                Worker 4: Sending response: Test
                                Worker 4: finished request
                                Worker 4: awaiting request
                                Worker 4: Removing Element from Queue. New size: 1
                                Worker 4: handling request
                                Worker 4: Client request: <READ file1.txt,1>
                                Worker 4: Reading
                                        Worker 5: Ending read
                                        Worker 5: Sending response: Test
                                        Worker 5: finished request
                                        Worker 5: awaiting request
                                        Worker 5: Removing Element from Queue. New
size: 0
                                        Worker 5: handling request
                                        Worker 5: Client request: <READ
file1.txt,1>
                                        Worker 5: Reading
```

- Alle 5 Worker bearbeiten einen zweiten Auftrag. Im Folgenden beenden die Worker diesen zweiten Auftrag und warten wieder an der Queue. Nun ist diese jedoch leer, sodass die Worker im Wartezustand bleiben.

```
        Worker 1: Ending read
        Worker 1: Sending response: Test
        Worker 1: finished request
        Worker 1: awaiting request
                        Worker 3: Ending read
                        Worker 3: Sending response: Test
                        Worker 3: finished request
                        Worker 3: awaiting request
                Worker 2: Ending read
                Worker 2: Sending response: Test
```

```
                Worker 2: finished request
                Worker 2: awaiting request
                                Worker 4: Ending read
                                Worker 4: Sending response: Test
                                Worker 4: finished request
                                Worker 4: awaiting request
                                        Worker 5: Ending read
                                        Worker 5: Sending response: Test
                                        Worker 5: finished request
                                        Worker 5: awaiting request
```

- Alle Worker haben ihre Aufträge beendet und warten an der Queue.