

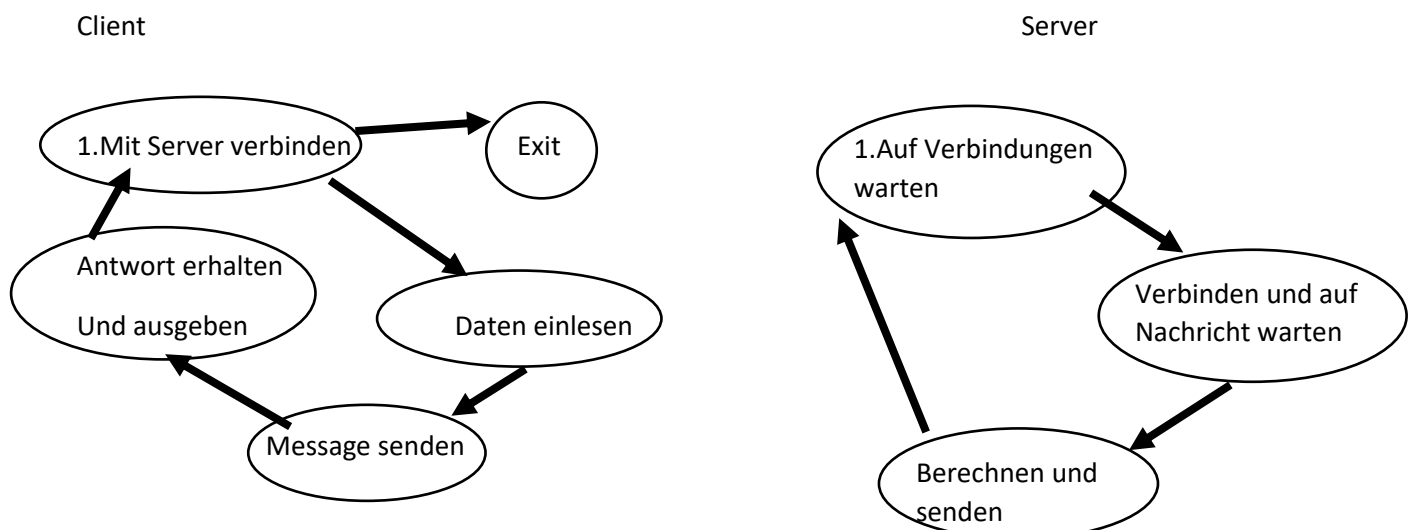
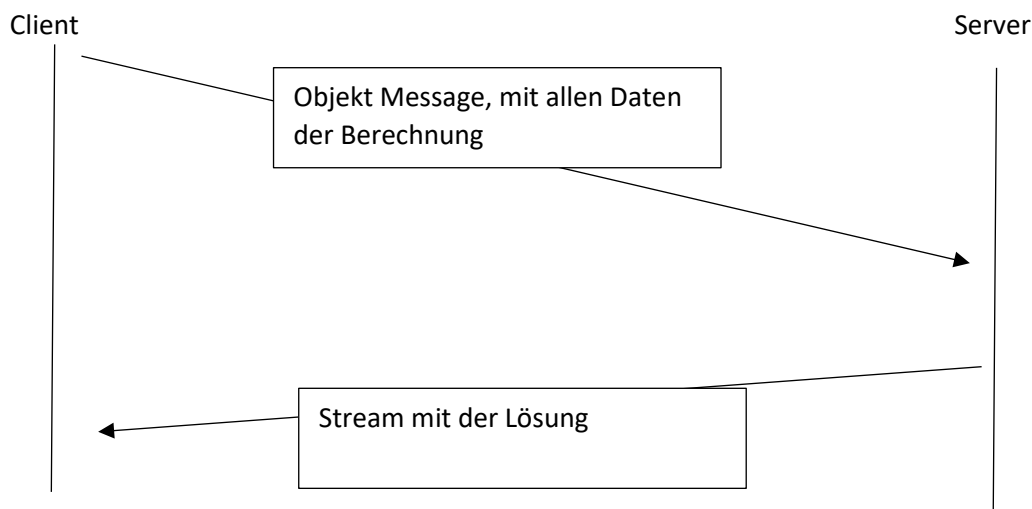
Protokolle aufgaben

Protokoll Aufgabe 1:

Kurze Aufgabenstellung: Ein Server Client Programm erstellen, indem der Client dem Server Rechnungen sendet. Der Server berechnet sie und sendet diese wieder zurück.

Das Programm wurde mit IntelliJ realisiert. Die JAR Files befinden sich in \out\artifacts

Zum Ablauf: Der Client liest die IP des Servers mittels Konsole ein und verbindet sich dorthin. Der Server gibt die Verbindung auf der Konsole aus. Der Client liest nun vom Benutzer die erste Zahl, das Berechnungszeichen und die zweite Zahl ein. Speichert diese in der Klasse *Message*. Und sendet die ganze Klasse dem Server. Dieser liest die Werte aus der Klasse aus, berechnet, gibt die Berechnung auf der Konsole aus und sendet das Ergebnis als Stream zurück. Der Client gibt dieses auf der Konsole aus. Der Client kann nun wieder von vorne Beginnen. Mit dem Befehl „exit“ kann der Client das Programm beenden. Der Server kann eine Verbindung auf einmal haben, kann aber viele Verbindungen hintereinander abarbeiten. Eingabefehler können aufgefangen werden, bei anderen Fehlern, stürzt nur der Client ab, der Server läuft weiter.



Screenshots:

Client:

```
IP des Servers eingeben, auf dem der Calc-service auf Port 4242 läuft
(Default 127.0.0.1)Tippe 'exit' to exit
127.0.0.1
Erste Zahl zum berechnen(ganze Zahlen)
5
Operationszeichen eingeben (+,-,*,/)
+
Zweite Zahl zum berechnen(ganze Zahlen)
8
= 13
IP des Servers eingeben, auf dem der Calc-service auf Port 4242 läuft
(Default 127.0.0.1)Tippe 'exit' to exit
```

Server:

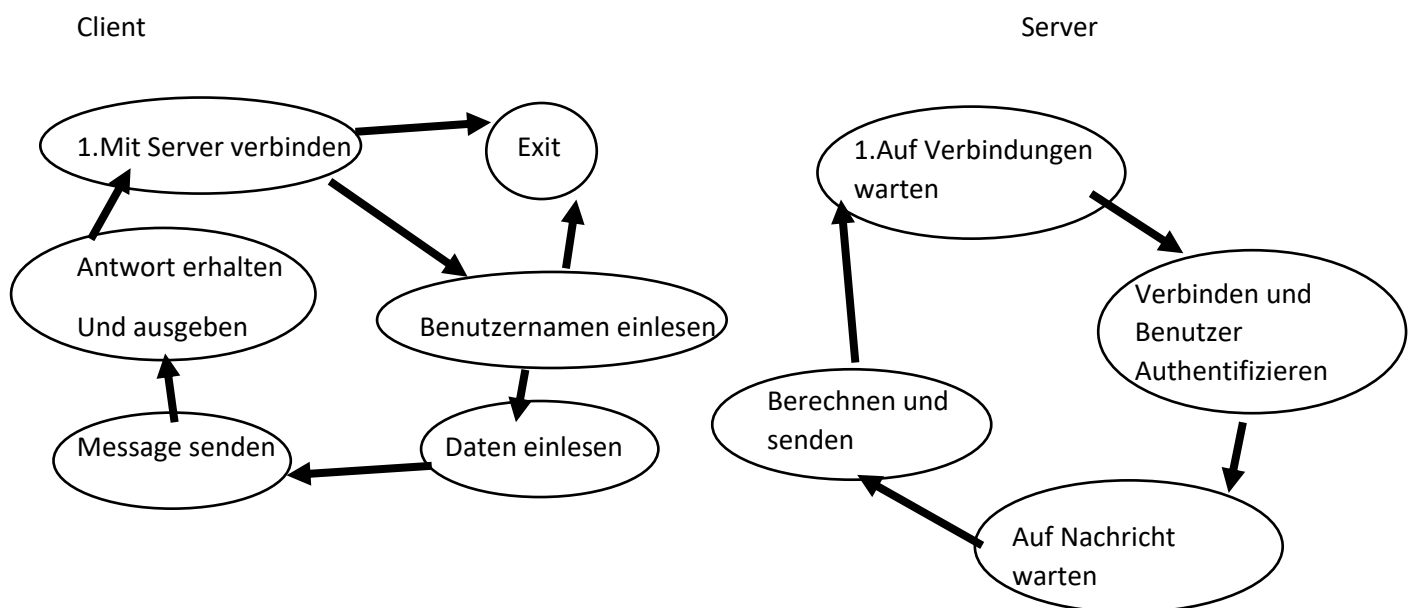
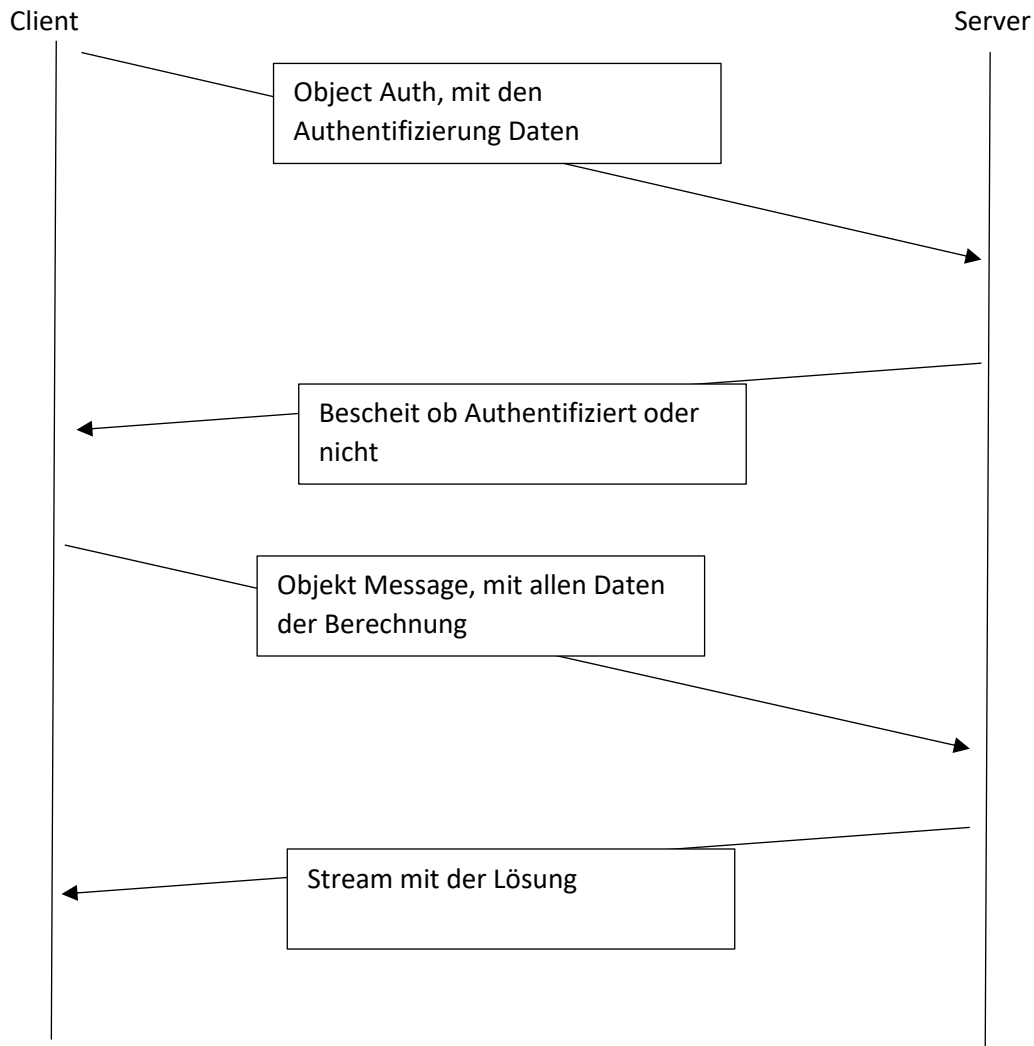
```
Server gestartet
verbunden mit:/127.0.0.1:4242
5+8
=13
Berechnung abgeschlossen und getrennt
```

Protokoll Aufgabe 2:

Kurze Aufgabenstellung: Ein Server Client Programm erstellen, indem der Client sich beim Server Authentifiziert durch einen Benutzernamen um dann Berechnungen an den Server zu senden. Der Server berechnet sie und sendet diese wieder zurück. Falls er nicht Authentifiziert wird kann der Benutzer nichts berechnen lassen.

Das Programm wurde mit IntelliJ realisiert. Die JAR Files befinden sich in \out\artifacts

Zum Ablauf: Der Client liest die IP des Servers mittels Konsole ein und verbindet sich dorthin. Der Server gibt die Verbindung auf der Konsole aus. Der Benutzer wird aufgefordert einen Benutzernamen einzugeben, mit dem er sich beim Server Authentifizieren kann. Dies geschieht durch die Klasse *Auth* die übergeben wird. Falls er Authentifiziert wurde also vom Server eine Bestätigung erhält, liest das Client-Programm vom Benutzer die erste Zahl, das Berechnungszeichen und die zweite Zahl ein. Speichert diese in der Klasse *Message*. Und sendet die ganze Klasse dem Server. Dieser liest die Werte aus der Klasse aus, berechnet, gibt die Berechnung auf der Konsole aus und sendet das Ergebnis als Stream zurück. Der Client gibt dieses auf der Konsole aus. Der Client kann nun wieder von vorne Beginnen mit der Authentifizierung. Mit dem Befehl „exit“ kann der Client das Programm beenden. Der Server kann eine Verbindung auf einmal haben, kann aber viele Verbindungen hintereinander abarbeiten. Eingabefehler können aufgefangen werden.



Screenshots:

Client:

```
IP des Servers eingeben, auf dem der Calc-service mit Authentifizierung auf Port 4242 läuft
(Default 127.0.0.1)Tippe 'exit' to exit
127.0.0.1
Benutzername eingeben (oder exit)
Hans
Authentifiziert!
Erste Zahl zum berechnen(ganze Zahlen)
9
Operationszeichen eingeben (+,-,*,/)
*
Zweite Zahl zum berechnen(ganze Zahlen)
3
= 27
Benutzername eingeben (oder exit)
exit
Programm beendet
```

Server:

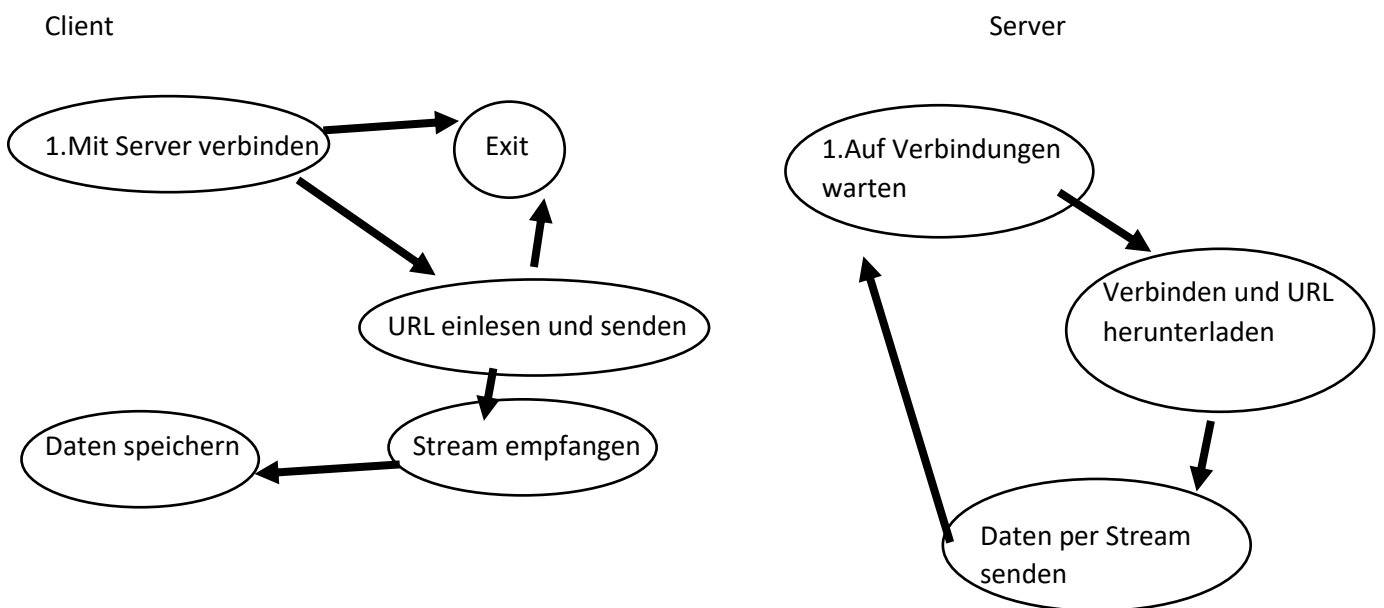
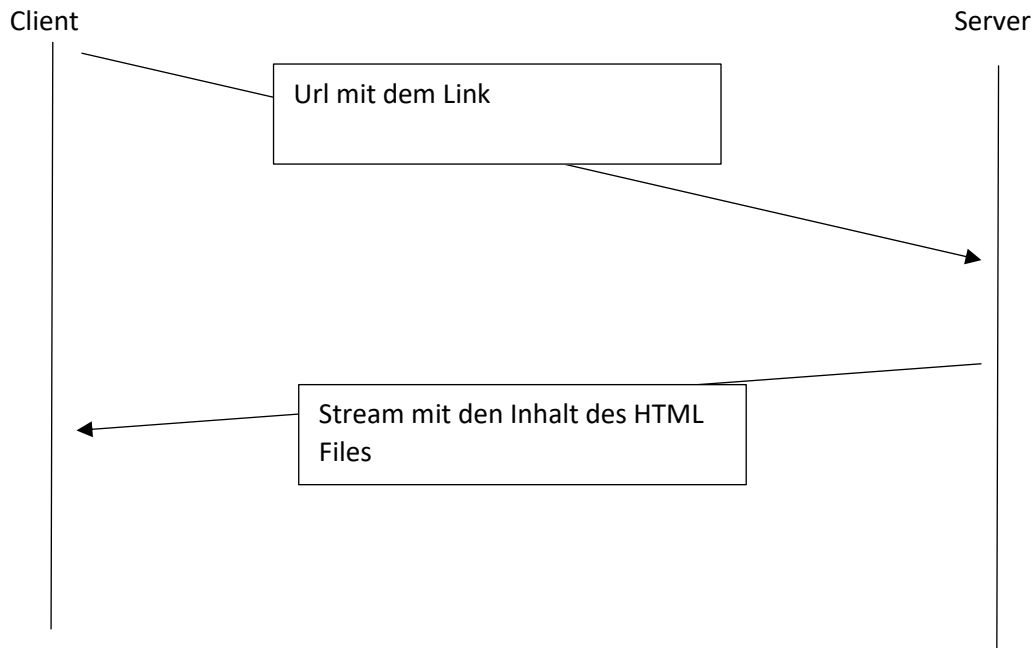
```
Server gestartet
verbunden mit:/127.0.0.1:4242
Verbunden mit:Hans
9*3
=27
Berechnung abgeschlossen und getrennt
```

Protokoll Aufgabe 3:

Kurze Aufgabenstellung: Ein Server Client Programm erstellen, indem der Client dem Server eine URL einer Webseite sendet. Der Server lädt das HTML File herunter und sendet es dem Client. Der Client speichert diese ab.

Das Programm wurde mit IntelliJ realisiert. Die JAR Files befinden sich in \out\artifacts

Zum Ablauf: Der Client liest die IP des Servers mittels Konsole ein und verbindet sich dorthin. Der Server gibt die Verbindung auf der Konsole aus. Der Benutzer muss jetzt die URL der Webseite eingeben die er herunterladen möchte. Diese wird in die Klasse *Url* gespeichert und dann gesendet. Das http(s):// muss er auch angeben damit der Server weiß welches Protokoll er verwenden muss. Der Server lädt das HTML File lokal ab und sendet es per Stream an den Client, dieser speichert diese dann ab. Der Server kann eine Verbindung auf einmal haben, kann aber viele Verbindungen hintereinander abarbeiten. Eingabefehler können aufgefangen werden, bei anderen Fehlern, stürzt nur der Client ab, der Server läuft weiter.









Screenshots:

Client:

```
IP des Servers eingeben, auf dem der URL-Download Port 4242 läuft
(Default 127.0.0.1)Tippe 'exit' to exit
127.0.0.1
Url eingeben mit https:// (oder exit)
http://www.stol.it
Programm beendet
```

Server:

```
Server gestartet  
verbunden mit:/127.0.0.1:4242  
http://www.stol.it Wird heruntergeladen  
gedownloadet, gesendet und getrennt
```

 .idea	05.02.2017 10:01	Dateiordner	
 out	05.02.2017 09:28	Dateiordner	
 src	05.02.2017 09:28	Dateiordner	
 Client-downloadet.html	05.02.2017 10:01	Chrome HTML Do...	169 KB
 IntelliJ.iml	09.01.2017 10:55	IML-Datei	1 KB
 Server-exit.html	05.02.2017 10:01	Chrome HTML Do...	169 KB

Protokoll Aufgabe 4:

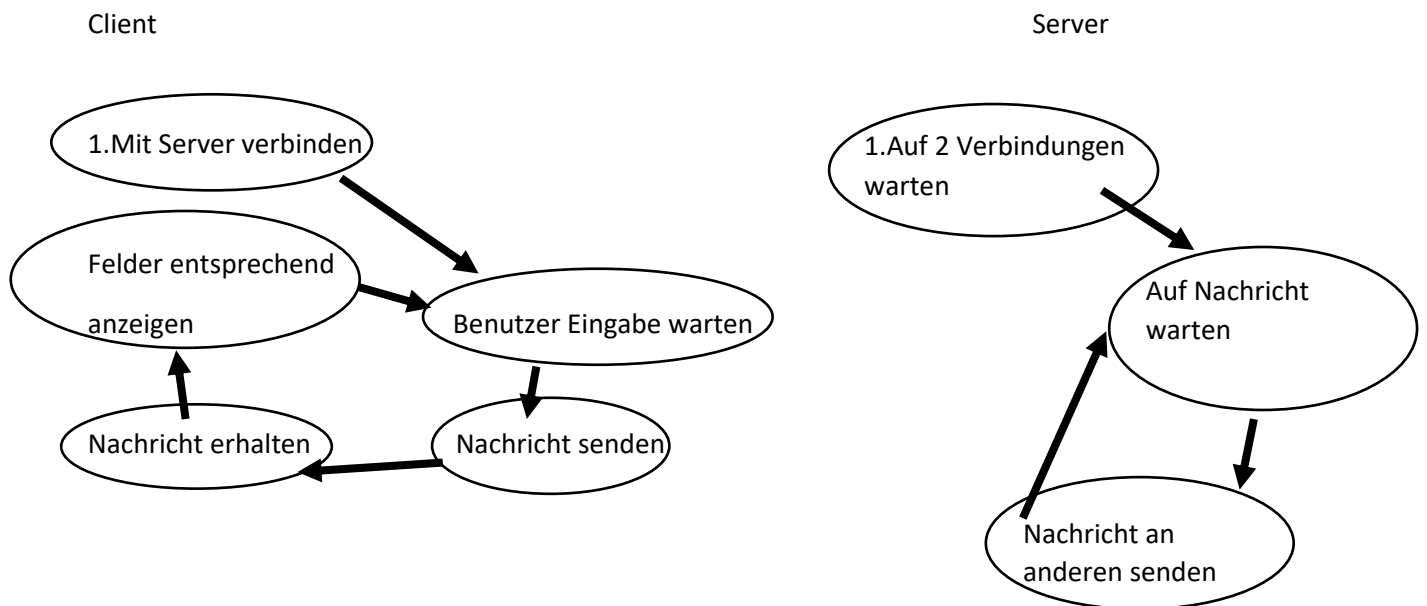
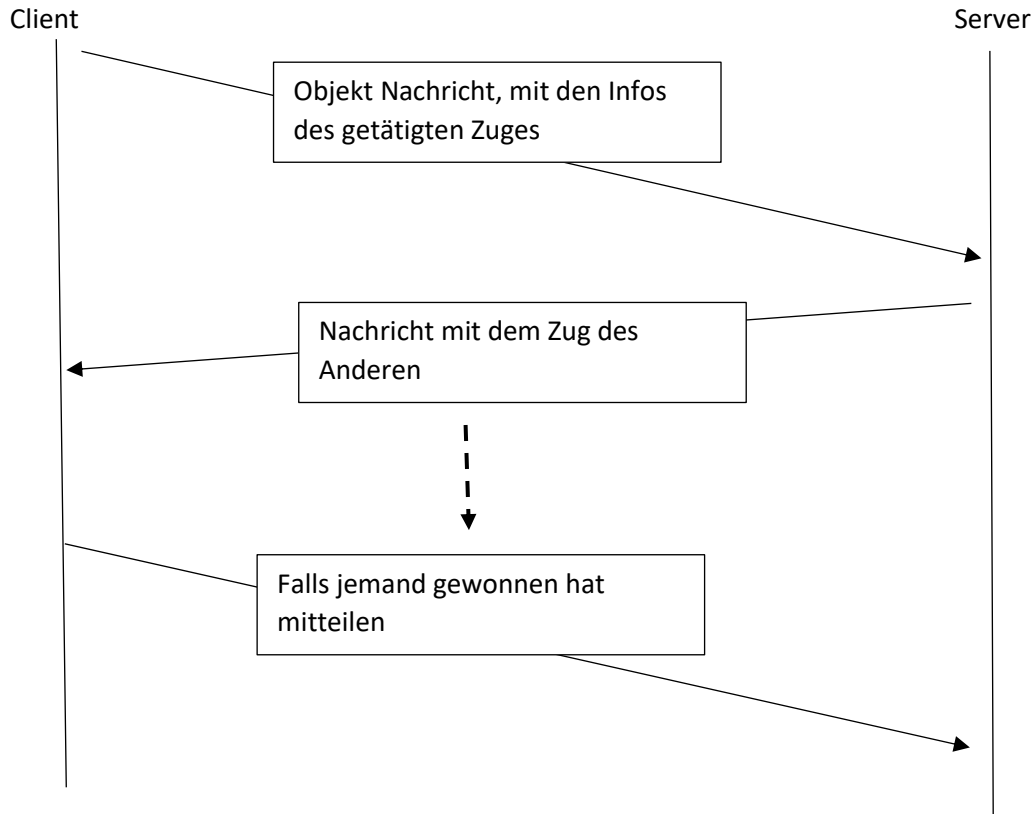
Kurze Aufgabenstellung: Ein Server Client Programm erstellen, indem zwei Clients über einen Server miteinander 4-Gewinnt spielen können.

Dieses Programm wurde auf einen anderes selbst programmiertes Programm aufgebaut.

Das Programm wurde mit IntelliJ realisiert. Die JAR Files befinden sich in \out\artifacts

Zum Ablauf: Der Client kann sich mittels IP zu einem Server Verbinden, oder einen Server erstellen. Alles wird mittels Grafischer Oberfläche eingegeben. Der Client sendet dem Server, mittels der Klasse Nachricht, den Punkt an dem er seinen Stein einfüge will. Der Server speichert dies und sendet es dem anderen Client. Beide Clients haben *Listener* die auf solche Nachrichten Warten. Der Server nutzt zudem die Klasse *writer* um Nachrichten zu senden. Falls jeman gewinnt erkennen dies die Clients und geben dies aus.

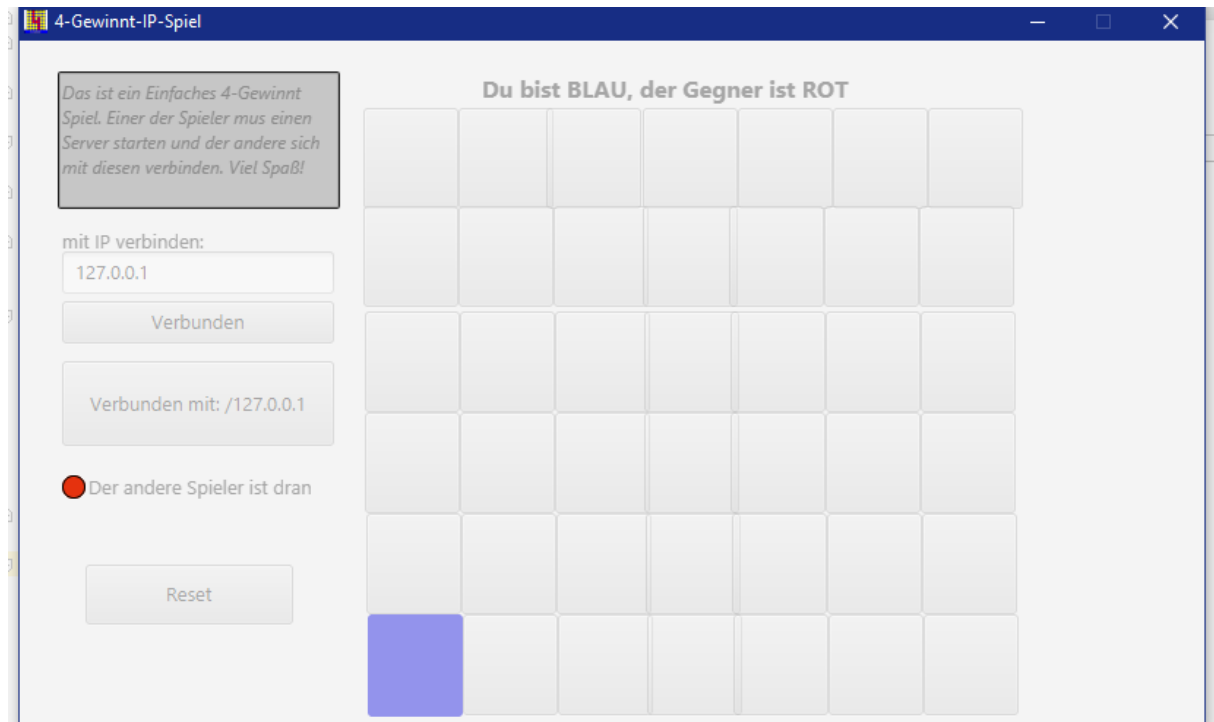
#Das Programm ist nicht ganz fertig, weil ein Fehler alles aufhält. Die Spielregeln und die Gewinn-Erkennung müssen noch implementiert werden und auch der Fehler muss gefunden werden damit alles funktioniert.



Mirco Borri

Screenshots:

Client:



Server:

```
Server Gestartet
Client /127.0.0.1 mit server verbunden
mit server verbunden
java.io.StreamCorruptedException: invalid type code: AC
    at java.io.ObjectInputStream.readObject0(ObjectInputStream.java:1381)
    at java.io.ObjectInputStream.readObject(ObjectInputStream.java:373)
    at listerner.run(listerner.java:31)
Fehler aufgetreten!!!
```

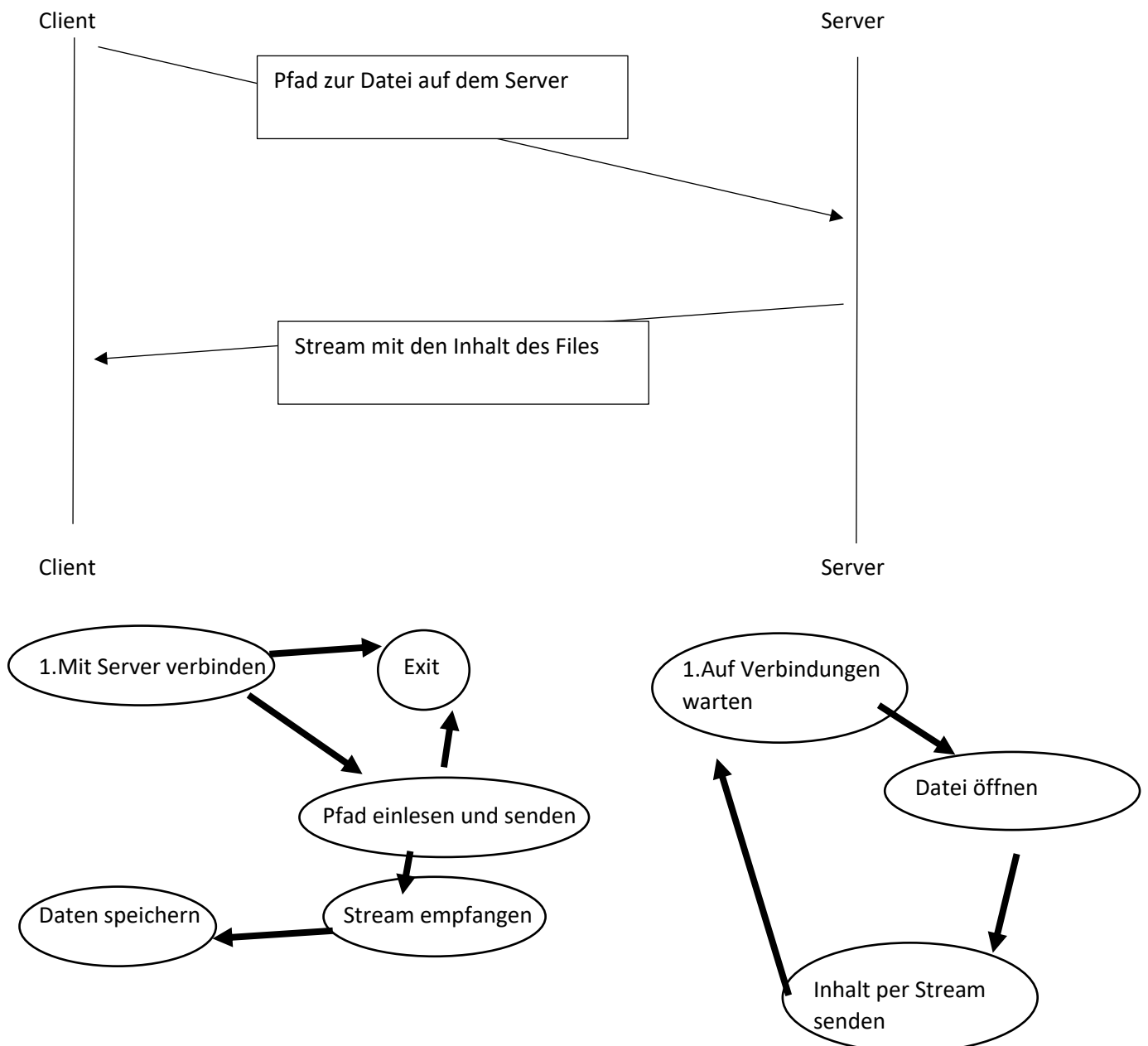
Der Fehler

Protokoll Aufgabe 5:

Kurze Aufgabenstellung: Ein Server Client Programm erstellen, indem der Client dem Server einen Pfad zu einer, auf dem Server liegenden, Datei angibt. Der Server sendet dies dem Client. Der Client speichert es ab. Mehrere Clients sollen parallel dieses dienst Nutzen können.

Das Programm wurde mit IntelliJ realisiert. Die JAR Files befinden sich in \out\artifacts






Zum Ablauf: Der Client liest die IP des Servers mittels Konsole ein und verbindet sich dorthin. Der Server gibt die Verbindung auf der Konsole aus. Der Benutzer muss jetzt den Pfad der Datei eingeben die er herunterladen möchte. Diese wird in die Klasse *Message* gespeichert und dann gesendet. Der Server öffnet das File und sendet den Inhalt per Stream an den Client, dieser speichert dies dann ab. Der Server kann mehrere Verbindungen auf einmal haben, dies wurde mit einem Executor realisiert. Eingabefehler können aufgefangen werden, bei anderen Fehlern, stürzt nur der Client ab, der Server läuft weiter.



Screenshots:

Client:

```
IP des Servers eingeben, auf dem der File-Download Port 4242 läuft
(Default 127.0.0.1)Tippe 'exit' to exit
127.0.0.1
File mit Pfad eingeben (oder exit)
A:/Test.txt
Programm beendet
```

	.idea	05.02.2017 10:11	Dateiordner	
	out	05.02.2017 09:28	Dateiordner	
	src	05.02.2017 10:11	Dateiordner	
	Client-downloadet.txt	05.02.2017 10:11	Textdokument	1 KB
	IntelliJ.iml	09.01.2017 10:55	IML-Datei	1 KB

Server:

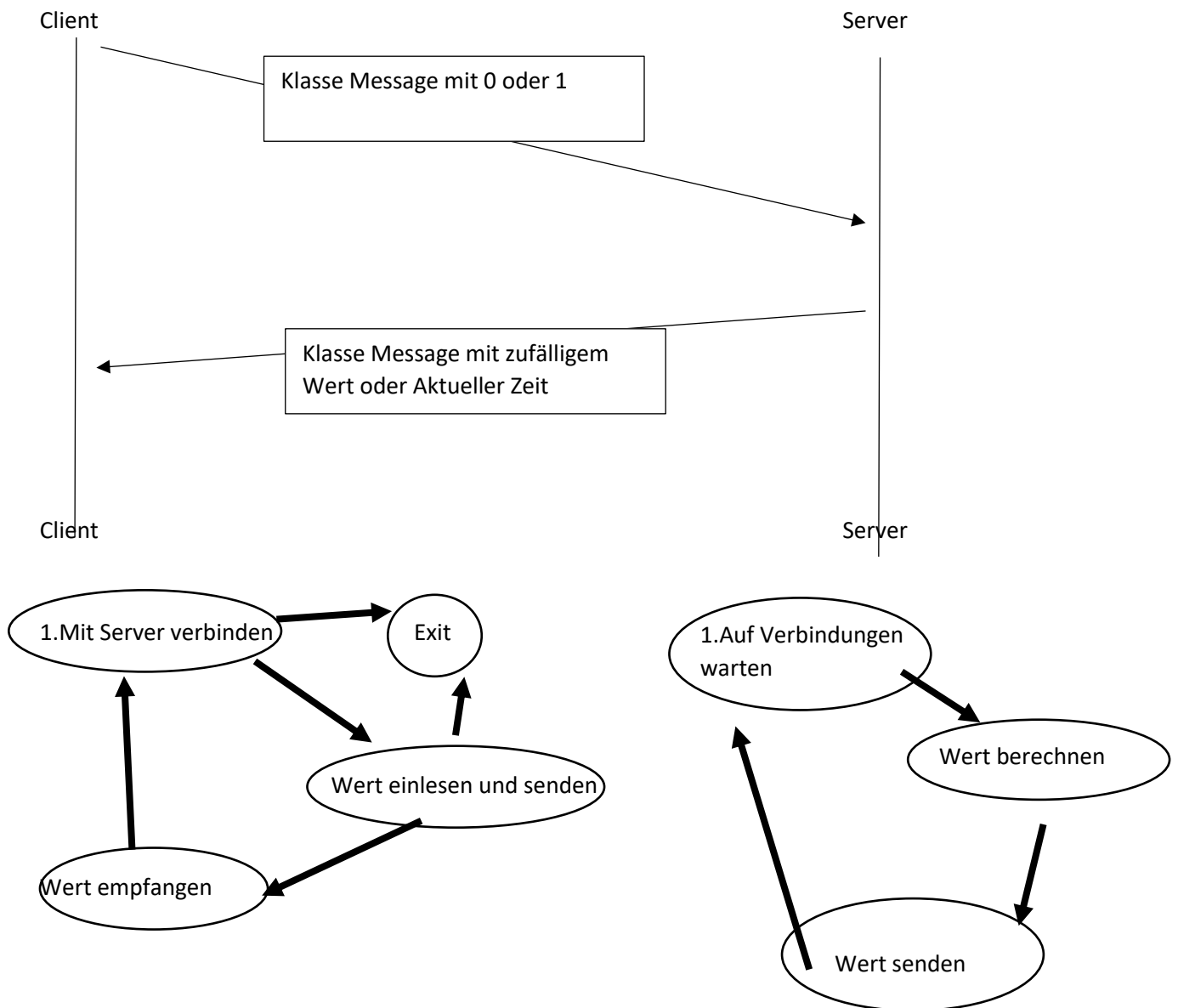
```
Server gestartet
verbunden mit:/127.0.0.1:4242
A:\Test.txt Wird gesendet
gesendet und getrennt
```

Protokoll Aufgabe 6:

Kurze Aufgabenstellung: Ein Server Client Programm erstellen, indem der Client dem Server eine Anfrage zu Aktuelle Zeit oder ein zufälliger Wert sendet. Der Server soll dafür länger brauchen und es zurücksenden. Eine Version soll mit einem Thread und eine mit mehreren realisiert werden.

Das Programm wurde mit IntelliJ realisiert. Die JAR Files befinden sich in \out\artifacts

Zum Ablauf: Der Client liest die IP des Servers mittels Konsole ein und verbindet sich dorthin. Der Server gibt die Verbindung auf der Konsole aus. Der Benutzer muss jetzt 0 oder 1 eingeben, zufälliger Wert oder Aktuelle Zeit. Die Nachricht wird mit der Klasse Message übertragen, und mit einer gleichen Klasse wieder zurück gesendet. Eine Version arbeitet immer einen Client nach einander ab, die andere kann mehrere gleichzeitig bedienen. Mit Thread.sleep() soll eine längere Berechnung simuliert werden. Beobachtung: Die Clients bekommen die Antwort ca. gleich schnell, nur es kann sich nur ein einziger Client zu der Version mit einem Thread verbinden.



Screenshots:

Client:

```
IP des Servers eingeben, auf dem die lange Wertberechnung auf Port 4242 läuft
(Default 127.0.0.1)Tippe 'exit' to exit
127.0.0.1
Radom Wert(0) oder getTime(1)?(oder exit)
0
Wert 6113.142516444354
Radom Wert(0) oder getTime(1)?(oder exit)
exit
Programm beendet
```

Server:

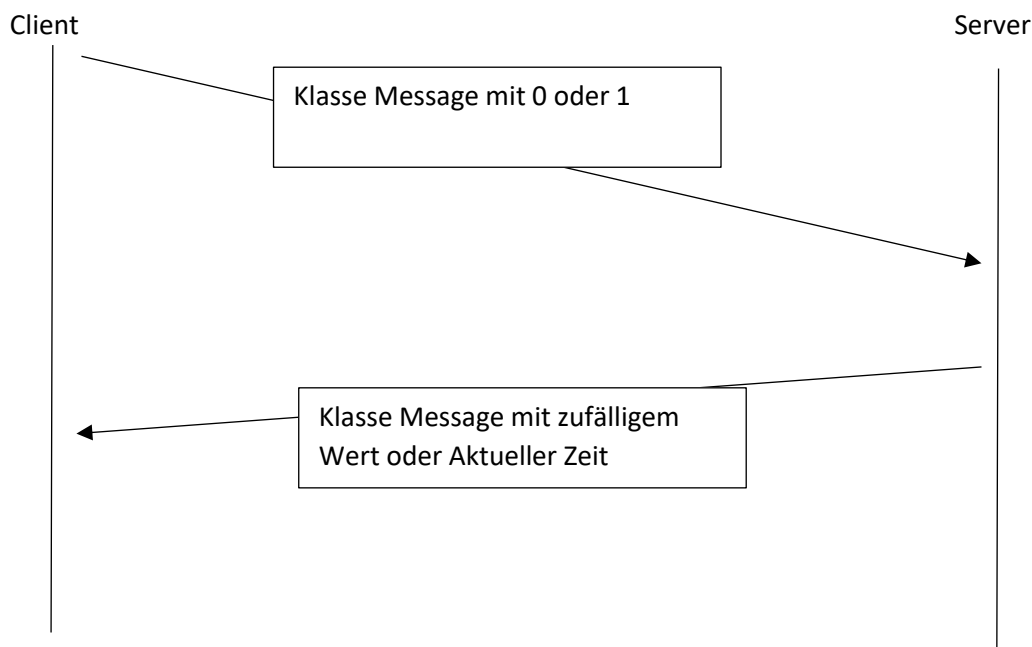
```
Server gestartet  
verbunden mit:/127.0.0.1:4242  
Wert:6113.142516444354
```

Protokoll Aufgabe 7:

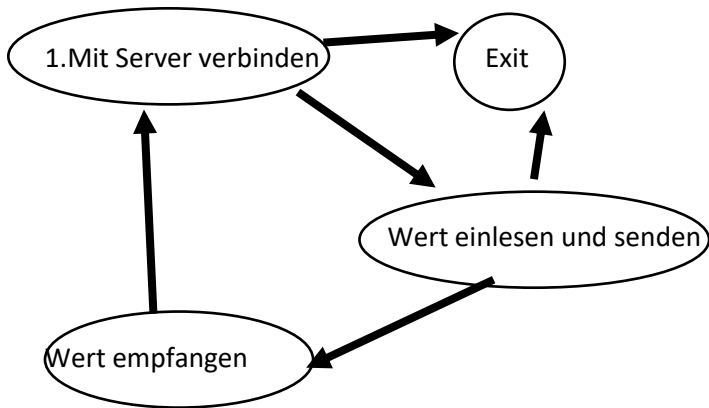
Kurze Aufgabenstellung: Ein Server Client Programm erstellen, indem der Client dem Server eine Anfrage zu Aktuelle Zeit oder ein zufälliger Wert sendet. Der Server soll dafür länger brauchen und es zurücksenden. Der Server und der Client sollen zudem alle Fehler abfangen, und der Server soll einen sauberen Shutdown durchführen können.

Das Programm wurde mit IntelliJ realisiert. Die JAR Files befinden sich in \out\artifacts

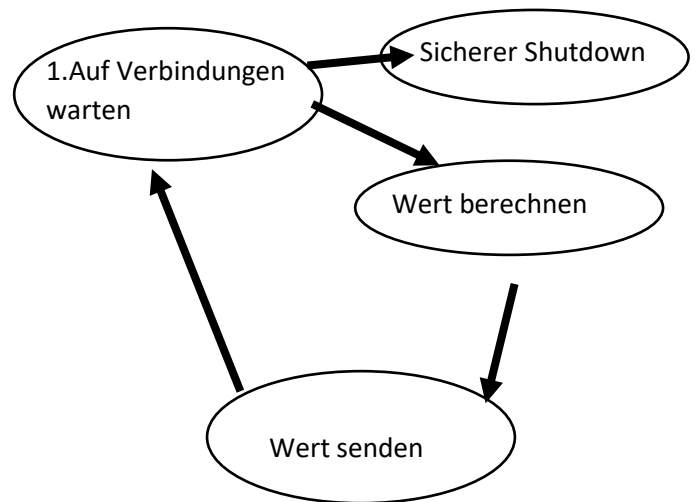
Zum Ablauf: Der Client liest die IP des Servers mittels Konsole ein und verbindet sich dorthin. Der Server gibt die Verbindung auf der Konsole aus. Der Benutzer muss jetzt 0 oder 1 eingeben, zufälliger Wert oder Aktuelle Zeit. Die Nachricht wird mit der Klasse Message übertragen, und mit einer gleichen Klasse wieder zurück gesendet. Mit Thread.sleep() soll eine längere Berechnung simuliert werden. Der Client kann zudem den Server auffordern sich sicher abzuschalten, alle anderen Threads werden sicher abgeschaltet mit dem ExecutorService. .



Client



Server



Screenshots:

Client:

```
IP des Servers eingeben, auf dem die lange Wertberechnung auf Port 4242 läuft
(Default 127.0.0.1)Tippe 'exit' to exit
127.0.0.1
Radom Wert(0) oder getTime(1)?(oder exit)
0
Wert 6113.142516444354
Radom Wert(0) oder getTime(1)?(oder exit)
exit
Programm beendet
```

Server:

```
Server gestartet
verbunden mit:/127.0.0.1:4242
Wert:6113.142516444354

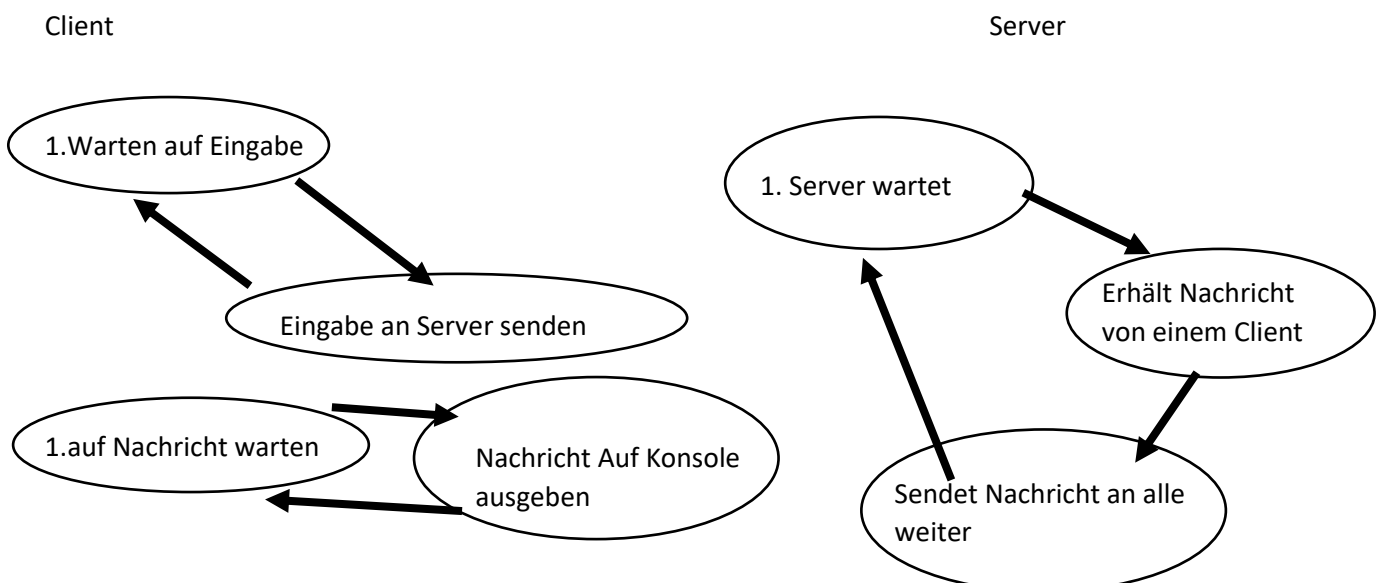
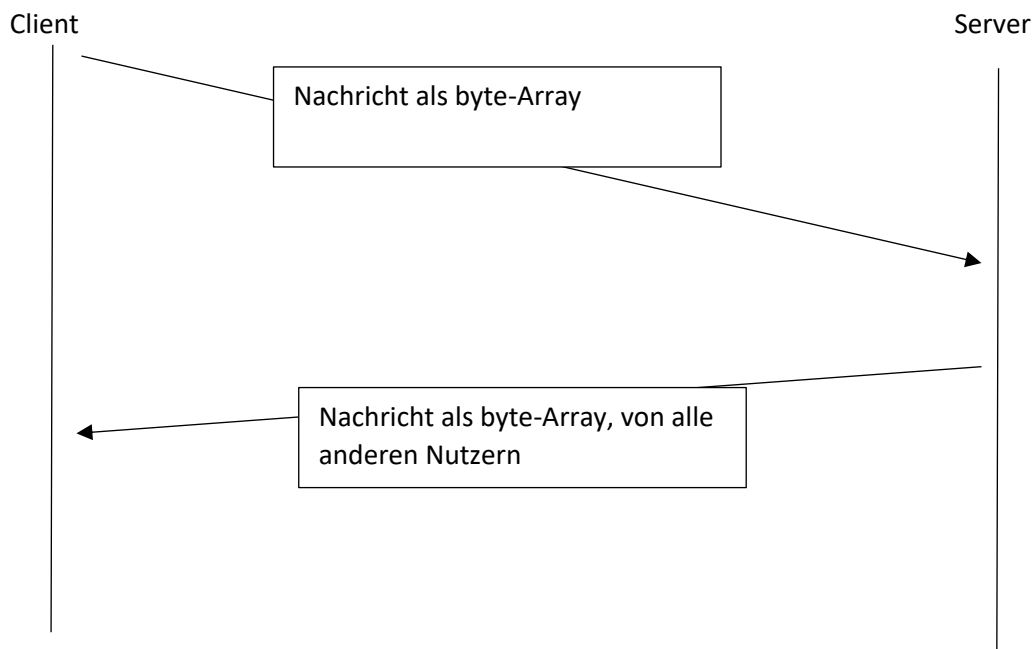
verbunden mit:/127.0.0.1:4242
Server wird heruntergefahren
Server beendet
```

Protokoll Aufgabe 8:

Kurze Aufgabenstellung: Ein Server Client Programm erstellen, indem mehrere Clients in einem Netz mit einem Chat sich austauschen können. Dabei wird UDP verwendet.

Das Programm wurde mit IntelliJ realisiert. Die JAR Files befinden sich in \out\artifacts

Zum Ablauf: Der Client liest die IP des Servers mittels Konsole ein und sendet alle Eingaben des Benutzers zu dieser Adresse. Der Server empfängt diese und sendet die Nachricht mittels Broadcast an alle anderen Clients die dieses Programm verwenden. Der Chat funktioniert nur wenn alle Clients und der Server im gleichen Netz hängen. Jeder Client hat einen Listener-Thread der alle erhaltenen Nachrichten vom Server auf der Konsole ausgibt. Da der Server aber an alle die Nachricht weiterleitet, erhält der Client der es gesendet hat auch die Nachricht und gibt sie aus.



Screenshots:

Client:

```
IP des Servers eingeben, auf dem der UDP-Chat auf Port 4242 und 4545 läuft
(Default 127.0.0.1)Tippe 'exit' to exit
127.0.0.1
Eingabe oder exit:
Listener gestartet
Hallo
Anderer Benutzer(oder eigene Nachricht):Hallo
Wie Gehts?
Anderer Benutzer(oder eigene Nachricht):Wie Gehts?
```

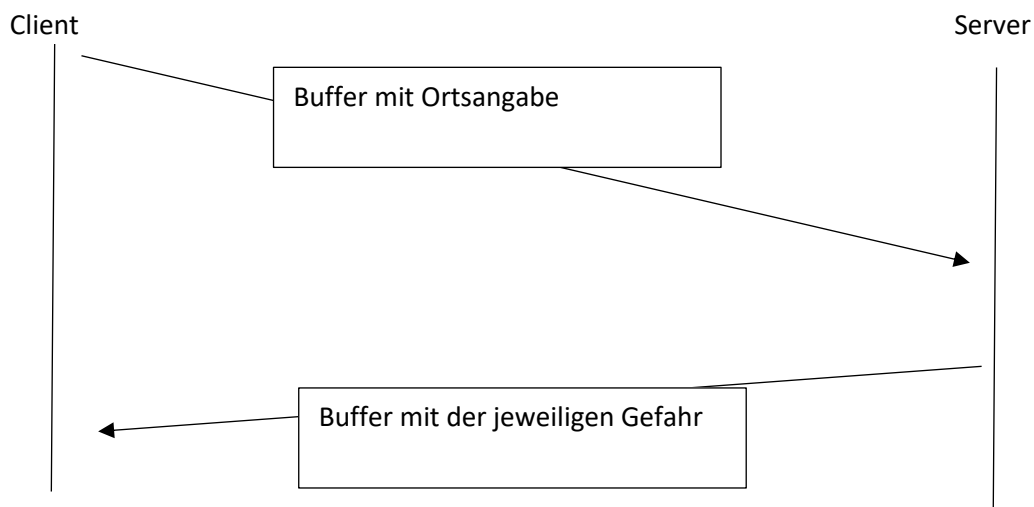
Server:

```
Server gestartet
/127.0.0.1: RECEIVED: Hallo
/127.0.0.1: RECEIVED: Wie Gehts?
```

Protokoll Aufgabe 9:

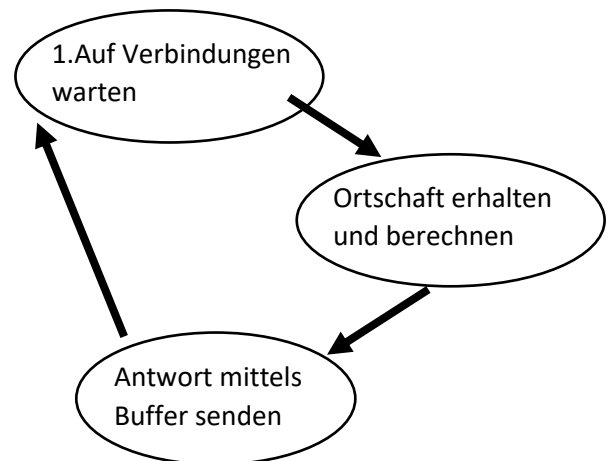
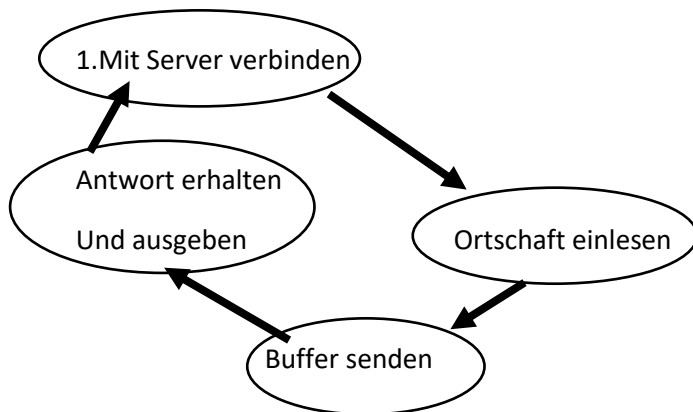
Kurze Aufgabenstellung: Ein Server Client Programm erstellen, indem der Client dem Server eine Ortschaft sendet und der Server antwortet mit der Jeweiligen Lawinengefahr. Hierbei wird C in Linux verwendet

Zum Ablauf: Der Client liest die IP des Servers mittels Konsole ein und verbindet sich dorthin. Der Client liest nun vom Benutzer die Ortschaft ein und sendet diese an den Server. Der Server berechnet die Gefahr(Zufällige Zahl in dieser Übung) und sendet es dem Client. Der Server gibt dabei auf diese Infos auf der Konsole aus.



Client

Server



Screenshots:

Client:

```
IP des Servers eingeben auf dem der Lawinen-berechnungs-service auf Port 4242 läuft:  
127.0.0.1  
Welche Ortschaft?  
Pfitsch  
Gefahr: 3
```

Server:

```
Warten...  
für Ortschaft berechnen: Pfitsch  
Gefahr: 3  
Warten...  
□
```