— Gruppenarbeit 2 (P): Streams und Sockets —

Aufgabe 1: Gruppen 1 und 2 **(Greif den Zaster!)**

Inspiriert von den vielen im Fernsehen ausgestrahlten Quizshows haben Sie beschlossen, ein netzwerkfähiges Ratespiel zu programmieren. Das Spiel mit dem vielversprechenden Namen `Greif den Zaster!` immitiert eine typische Gameshow, wobei der Spieler in die Rolle von Paul Bommel schlüpft, der als Kandidat in der Sendung auftritt. Als dieser bekommt er vom Showmaster mehrere Fragen gestellt, die er alle richtig beantworten muss. Im Gegensatz zu anderen Sendungen endet das Spiel aber nicht, wenn Paul eine Frage falsch beantwortet, sondern es verringert sich nur sein Punktestand. Diese Regel wurde bereits nach der ersten Sendewoche eingeführt, da es für den Sender sonst unmöglich gewesen wäre, die 90 minütige Sendezeit auszufüllen.

Da Sie bisher noch keine besseren Techniken kennengelernt haben und vor allem, weil Sie später den Unterschied zu anderen Techniken der verteilten Programmierung verstehen sollen, soll das Spiel mit einfachen Streams und Sockets umgesetzt werden[1]. Dabei übernimmt der Server die Rolle des Quizmasters und der Client die Rolle des Kandidaten. Ein typischer Programmablauf sieht daher wie folgt aus:

Server auf Port 4000 starten:

```
$ cd Server
$ java Main 4000
Bereit!
```

Client mit Server verbinden:

```
$ cd Client
$ java Main localhost 4000

Günther: Sind Sie bereit Herr Bommel? Dann kommen
         wir gleich zu Ihrer ersten Frage.

Günther: Wen verkörperte Elton John in der Rockoper "Thommy"?

[1] Crocodile Man
[2] Pinball Wizard
[3] Mega Heinz
[4] Power Paul

[P] Punktestand abfragen
[E] Spiel beenden
[S] Server herunterfahren

Paul sagt: 3

Günther: Tut mir Leid, Herr Bommel. Das ist falsch.
         Pinball Wizard wäre richtig gewesen.
```

[1] Vgl. Webprogrammierung, Kapitel 1: Streams und Sockets

```
Günther: In welchem Land spielt die Oper "Aida"?

[1] Ägypten
[2] Ätopien
[3] Kenia
[4] Israel

[P] Punktestand abfragen
[E] Spiel beenden
[S] Server herunterfahren

Paul sagt: P

Günther: Ihr aktueller Punktestand beträgt -13 Punkte.
```

In diesem Fall würden Client und Server folgende Daten austauschen, wobei zunächst der Server das Wort `WELCOME` an den Client schickt, um seine Empfangsbereitschaft zu signalisieren.

Server	`WELCOME`
Client	`NEXT QUESTION`
Server	`QUESTION/Wen verkörperte Elton John in der Rockoper "Thommy"?/Crocodile Man/Pinball Wizard/Mega Heinz/Power Paul`
Client	`ANSWER/1`
Server	`WRONG/Pinball Wizard`
Client	`NEXT QUESTION`
Server	`QUESTION/In welchem Land spielt die Oper "Aida"?/Ägypten/Ätopien/Kenia/Israel`
Client	`SCORE`
Server	`YOUR SCORE/-13`

Jeder Befehl besteht aus genau einer Zeile und kann daher mit den Methoden `println()` und `readLine()` des entsprechenden Datenstroms gesendet und empfangen werden. Falls neben dem eigentlichen Befehlswort weitere Daten übertragen werden sollen, werden diese durch einen Schrägstrich getrennt einfach an das Befehlswort angehängt. Dies wird zum Beispiel genutzt, um eine Frage und alle ihre Antworten an den Client zu schicken. Jener kann dann auf Befehlswort und Zusatzdaten wie folgt zugreifen. Das erste Element des Arrays beinhaltet dann immer das Befehlswort, alle weiteren Elemente beinhalten die optionalen Zusatzparameter.

```
while ((line = fromServer.readLine()) != null) {
    String[] command = line.split("/");
    // ...
}
```

Gruppe 1 (Server): Ihre Aufgabe besteht darin, den Serverteil der Anwendung zu schreiben, wobei einige Teile des Servers bereits fertig programmiert sind. Sie müssen daher nur den Teil des Servers programmieren, der das eigentliche Protokoll umsetzt.

Nehmen Sie zunächst folgende Änderungen an der Klasse `Main` vor:

1. Erweitern Sie die Klasse so, dass sie auf Verbindungsanfragen von den Clients reagiert. Sobald eine neue Verbindung aufgebaut wurde, soll ein neuer Thread erzeugt werden, der die eigentliche Kommunikation durchführt.

2. Übergeben Sie dem Thread das `questions`-Objekt, welches bereits in `main()` erzeugt wird. Achten Sie aber darauf, dass jeder Thread seine eigene Liste mit Fragen erhält, da die Liste von den Threads verändert wird.

Programmieren Sie nun die Threadklasse:

1. Nehmen Sie im Konstruktur einen Clientsocket sowie eine Liste mit den Quizfragen entgegen und speichern Sie beide Objekte in privaten Instanzattributen des Threads.

2. Besorgen Sie sich ebenfalls im Konstruktor einen `BufferedReader` und einen `PrintWriter`, um mit dem Client zu kommunizieren.

3. Fügen Sie der Klasse folgende Methode hinzu. Sie liefert Ihnen eine zufüllge Quizfrage und entfernt diese aus dem Vorrat, so dass sie nicht zweimal gestellt wird.

```
public Question getNextQuestion() {
    int i = (int) Math.floor(Math.random() * this.questions.size());
    return this.questions.remove(i);
}
```

4. Programmieren Sie die `run()`-Methode mit dem Kommunikationsprotokoll aus. Dabei müssen Sie als erstes das Wort `WELCOME` an den Client senden, um den Beginn des Spiels zu signalisieren. Anschließend lesen Sie in einer Endlosschleife immer eine Zeile vom Client ein und werten diese aus, wobei Sie als Antwort ebenfalls eine Zeile senden müssen. Setzen Sie dabei das in der nachfolgenden Tabelle beschriebene Protokoll um.

Empfangenes Kommando	**Reaktion des Servers**
Verbindungsaufbau	Antwortet mit `WELCOME`, um das Spiel zu beginnen.
`QUIT`	Sendet das Wort `FINISHED` an den Client und beendet die Kommunikation, indem die Methode `close()` des Clientsockets aufgerufen wird.
`SHUTDOWN`	Wie `QUIT`, zusäzlich wird aber die Variable `Main.quit` auf `true` gesetzt, um den Serverprozess zu beenden.
`SCORE`	Sendet das Kommando `YOUR SCORE` zusammen mit dem Punktestand des Spielers an den Client. Zum Beispiel: `YOUR SCORE/42`
`NEXT QUESTION`	Falls der Spieler bereits alle Fragen beantwortet hat (`this.questions.size()` ergiebt null), soll einfach die Meldung `NO MORE QUESTIONS` an den Client gesendet werden. Andernfalls sollen folgende Regeln gelten:

Empfangenes Kommando	Reaktion des Servers
	Wenn der Spieler auf die vorherige Frage noch nicht geantwortet hat (Befehl `ANSWER` wurde seit der letzten Frage nicht mehr empfangen), verliert er 26 Punkte. Es wird trotzdem eine neue Frage an den Client geschickt. Falls es keine Frage mehr gibt, wird ein Fehler geschickt: `ERROR/Ich habe keine Fragen mehr ...` Andernfalls wird der Befehl `QUESTION` zusammen mit der Frage und allen vier Antworten gesendet. Dabei können Sie den Fragetext und die Antworten mit den Methoden `String getQuestion()` und `String[] getAnswers()` des aktuellen Frageobjekts ermitteln.
`ANSWER/nr`	Falls der Server nicht auf die Antwort einer Frage wartet (die letzte Frage wurde bereits beantwortet oder es wurde noch gar keine Frage angefordert), soll ein Fehler gesendet werden: `ERROR/Ich habe noch keine Frage gestellt ...`. Andernfalls soll die vom Client gesendete Antwort mit der Methode `boolean isCorrect(int nr)` des aktuellen Frageobjekts überprüft werden. War die Antwort richtig, bekommt der Spieler 15 Punkte und es wird das Wort `RIGHT` an den Client gesendet. War die Antwort falsch, verliert der Spieler 13 Punkte und es wird `WRONG` zusammen mit der richtigen Antwort an den Client gesendet.
Ungültiges Kommando	Sendet einen Fehler an den Client: `ERROR/Wie bitte?!`

Gruppe 2 (Client): Ihre Aufgabe besteht darin, eine Swinganwendung zu schreiben, die anstelle des Kommandozeilenclients verwendet werden kann, um das Spiel zu spielen. Die eigentliche Anwendung ist schon fast fertig, Sie müssen nurnoch den Teil schreiben, der das Kommunikationsprotokoll umsetzt.

Nehmen Sie zunächst folgende Änderungen an der Klasse `RemoteConnection` vor:

1. Erweitern Sie die Methode `connect()` so, dass sie eine neue Verbindung mit dem Spieleserver aufbaut. Zieladresse und Portnummer finden Sie in den Instanzattributen `hostname` und `port`.
2. Erzeugen Sie außerdem einen `BufferedReader` und einen `PrintWriter`, welche benutzt werden können, um Daten mit dem Server auszutauschen. Speichern Sie die beiden Objekte in den Attributen `fromServer` und `toServer`.
3. Programmieren Sie die Methode `callServer()` zuende, indem Sie am Ende der

Methode eine Antwortzeile des Servers einlesen und diese in einem String-Array zurückgeben. Dabei müssen Sie den empfangenen Text bei jedem Schrägstrich trennen.

Programmieren Sie nun die Spielelogik der Klasse `GameController` fertig:

1. Am Ende der Methode `beginGame()` soll zweimal die Methode `callServer()` des `connection`-Objekts aufgerufen werden, um die Befehle `NEXT QUESTION` und `SCORE` an den Server zu schicken. Das Ergebnis der Aufrufe soll jeweils an die Methode `executeResponse()` übergeben werden, um das Hauptfenster aufzufrischen. Dabei müssen alle auftretenden Exceptions abgefangen und mit der Methode `Main.showException()` angezeigt werden.

2. In der Methode `answer()` sollen die Befehle `ANSWER`, `NEXT QUESTION` und `SCORE` an den Server geschickt und die jeweiligen Antworten ausgewertet werden. Achten Sie darauf, dass Sie bei `ANSWER` zusäzlich die Antwortnummer als String mitgeben. Auch hier müssen alle Exceptions abgefangen und angezeigt werden.

3. In der Methode `skipQuestion()` sollen analog zu `beginGame()` die Kommandos `NEXT QUESTION` und `SCORE` gesendet und deren Antworten ausgewertet werden. Auftretende Exceptions sollen ebenfalls angezeigt werden.

4. Programmieren Sie die Methode `executeResponse()` mit dem Kommunikationsprotokoll aus. Die Methode nimmt ein String-Array mit einer Antwort des Servers entgegen und aktualisiert daraufhin die Programmoberfläche. Dabei entsprechen der erste Eintrag des Arrays immer dem empfangenen Kommando und alle übrigen Einträge den angehängten Zusatzdaten. Setzen Sie das in der nachfolgenden Tabelle beschriebene Protokoll um.

Empfangenes Kommando	**Reaktion des Clients**

`NO MORE QUESTIONS`	Ruft die Methode `this.window.println()` auf, um folgenden Text im Hauptfenster auszugeben. Ebenso wird die Methode `enableAnswerButtons(false)` des Fensters aufgerufen, um die Spielknöpfe zu deaktivieren:
	
	`Günther: Tja, das war's dann. Damit ist die heutige Sendung schon vorbei. Ich wünsche Ihnen noch einen schönen Abend. Herzlichst, Ihr Günther Rauch!`
	
	`*Tusch* *Applaus*`
	`*Noch mehr Applaus*`
	`*Abspann*`

`YOUR SCORE`	Liest den aktuellen Punktestand aus der Serverantwort aus und übergibt diesen an die Methode `setScore(int)` des Hauptfensters.

Empfangenes Kommando	Reaktion des Clients

`QUESTION`	Speichert die empfangene Serverantwort im Attribut `currentQuestion` und ruft die Methode `enableAnswerButtons(true)` des Hauptfensters auf, um die Spielknöpfe zu aktivieren. Zusätzlich sollen die empfangene Frage und die möglichen Antworten im Hauptfenster ausgegeben werden. Zum Beispiel:
	`Wie nennt man den Aufbau und die Gestaltung eines Tanzes?`
	
	`(1) Choreographie`
	`(2) Cinematographie`
	`(3) Tautologie`
	`(4) Regie`

`RIGHT`	Ruft die Methode `setCorrect(true)` des Hauptfensters auf und gibt folgenden Text aus: `Günther: Halten Sie sich fest. Aber -- das war richtig!`

`WRONG`	Ruft die Methode `setCorrect(false)` des Hauptfensters auf und gibt folgenden Text aus:
	`Günther: Tut mir Leid, Herr Bommel. Das war falsch.`
	`«Antwort» wäre richtig gewesen.`

`ERROR`	Gibt folgenden Text zusammen mit der empfangenen Fehlermeldung im Spielfenster aus: `Günther (leicht irritiert): «Fehlermeldung»`
Unbekanntes Kommando	Wirft eine `IOException` mit folgendem Fehlertext: `Protokollfehler (Unbekanntes Kommando): «Empfangenes Kommando»`

Probieren Sie es aus: Lassen Sie erst Client und Server auf demselben Rechner laufen. Wenn dies funktioniert, führen Sie die beiden Programme auf unterschiedlichen Computern Ihrer Gruppe aus. Sie müssen hierfür nur den Wert `localhost` beim Aufruf des Clients durch die IP-Adresse des Rechners ersetzen, auf welchem der Server läuft. Probieren Sie auch, sich mit mehreren Rechnern und unterschiedlichen Clients gleichzeitig mit dem Server zu verbinden.

Aufgabe 2: Gruppen 3 und 4 **(Virtual Hero)**

Einmal als berühmter Rockgitarisst gefeiert werden. Diesen Traum können Sie jetzt wahr werden lassen, ohne selbst Gitarre spielen zu können. Denn Sie haben beschlossen, einfach einen netzwerkfähigen, virtuellen Gitarissten ähnlich den früher so beliebten virtuellen Haustieren zu programmieren. Im Gegensatz zu füher schlüpft der Spieler jedoch nicht in die Rolle eines Tierbesitzers, sondern in die Rolle eines entnervten Band Leaders, der sich aktiv um seinen Gitarissten kümmern muss. Das heißt, er muss dafür sorgen, dass der Gitarisst regelmäßig übt und die Gelegenheit bekommt, Auftritte mit großartigen Solos zu spielen. Außerdem muss er auf die Gesundheit seines Gitarrenheldes achten, damit dieser sich nicht zu Tode spielt. Zumindest nicht so lange, wie sich durch die Musik der Band hohe Einnahmen erzielen lassen.

Da Sie bisher noch keine besseren Techniken kennengelernt haben und vor allem, weil Sie später den Unterschied zu anderen Techniken der verteilten Programmierung verstehen sollen, soll das Spiel

mit einfachen Streams und Sockets umgesetzt werden[2]. Dabei übernimmt der Server die Rolle des virtuellen Gitarristen und der Client die Rolle seines Managers. Ein typischer Programmablauf sieht daher wie folgt aus:

Server auf Port 4000 starten:

```
$ cd Server
$ java Main 4000
Bereit!
```

Client mit Server verbinden:

```
$ cd Client
$ java Main localhost 4000

Punkte: 0, Zeit: 0, Energie: 100

[1] Üben
[2] Spielen
[3] Applaus
[4] Bier trinken
[W] Abwarten

[E] Spiel beenden
[S] Server herunterfahren

Ihre Managemententscheidung: 1

Mark übt bis die Finger glühen!

Punkte: 12, Zeit: 2, Energie: 92

[1] Üben
[2] Spielen
[3] Applaus
[4] Bier trinken
[W] Abwarten

[E] Spiel beenden
[S] Server herunterfahren

Ihre Auswahl: 4
Anzahl Managemententscheidung: 3

Mark ist beschäftigt. Er trinkt gerade Duff Bier!
Da kommt gerade der Duff Man herein und macht die Musik an.
```

[2] Vgl. Webprogrammierung, Kapitel 1: Streams und Sockets

```
Diese Party wird wohl länger dauern ...

Punkte: 12, Zeit: 5, Energie: 117

[1] Üben
[2] Spielen
[3] Applaus
[4] Bier trinken
[W] Abwarten

[E] Spiel beenden
[S] Server herunterfahren

Ihre Managemententscheidung: _
```

In diesem Fall würden Client und Server folgende Daten austauschen, wobei zunächst der Server das Wort `WELCOME` zusammen mit dem Namen der Spielfigur an den Client schickt, um seine Empfangsbereitschaft zu signalisieren.

Server	`WELCOME/Mark`
Client	`STATUS`
Server	`CURRENT STATUS/0/100/N`
Client	`PRACTICE`
Server	`PRACTICING`
Client	`STATUS`
Server	`CURRENT STATUS/12/2/92/N`
Client	`DRINK BEER/3`
Server	`I LIKE DUFF`
Client	`STATUS`
Server	`CURRENT STATUS/12/5/117/N`

Jeder Befehl besteht aus genau einer Zeile und kann daher mit den Methoden `println()` und `readLine()` des entsprechenden Datenstroms gesendet und empfangen werden. Falls neben dem eigentlichen Befehlswort weitere Daten übertragen werden sollen, werden diese durch einen Schrägstrich getrennt einfach an das Befehlswort angehängt. Dies wird zum Beispiel genutzt, um den Spielstand bestehend aus Punkten, Zeit und Energie an den Client zu schicken. Jener kann dann auf Befehlswort und Zusatzdaten wie folgt zugreifen. Das erste Element des Arrays beinhaltet dann immer das Befehlswort, alle weiteren Elemente beinhalten die optionalen Zusatzparameter.

```
while ((line = fromServer.readLine()) != null) {
    String[] command = line.split("/");
    // ...
}
```

Gruppe 1 (Serveranwendung): Ihre Aufgabe besteht darin, den Serverteil der Anwendung zu schreiben, wobei einige Teile des Servers bereits fertig programmiert sind. Sie müssen daher nur den Teil des Servers programmieren, der das eigentliche Protokoll umsetzt.

Nehmen Sie zunächst folgende Änderungen an der Klasse `Main` vor:

1. Erweitern Sie die Klasse so, dass sie auf Verbindungsanfragen von den Clients reagiert. Sobald eine neue Verbindung aufgebaut wurde, soll ein neuer Thread erzeugt werden, der die eigentliche Kommunikation durchführt.

2. Übergeben Sie dem Thread das `GuitarPlayer`-Objekt, welches bereits in `main()` erzeugt wird. Achten Sie aber darauf, dass jeder Thread sein eigenes Objekt erhällt, da dieses den Spielstand eines Spielers speichert.

Programmieren Sie nun die Threadklasse:

1. Nehmen Sie im Konstruktur einen Clientsocket sowie ein Objekt der Klasse `GuitarPlayer` entgegen und speichern Sie beide Objekte in privaten Instanzattributen des Threads.

2. Besorgen Sie sich ebenfalls im Konstruktor einen `BufferedReader` und einen `PrintWriter`, um mit dem Client zu kommunizieren.

3. Programmieren Sie die `run()`-Methode mit dem Kommunikationsprotokoll aus. Dabei müssen Sie als erstes das Wort `WELCOME` zusammen mit dem Namen der Spielfigur an den Client senden, um den Beginn des Spiels zu signalisieren. Anschließend lesen Sie in einer Endlosschleife immer eine Zeile vom Client ein und werten diese aus, wobei Sie als Antwort ebenfalls eine Zeile senden müssen. Setzen Sie dabei das in der nachfolgenden Tabelle beschriebene Protokoll um.

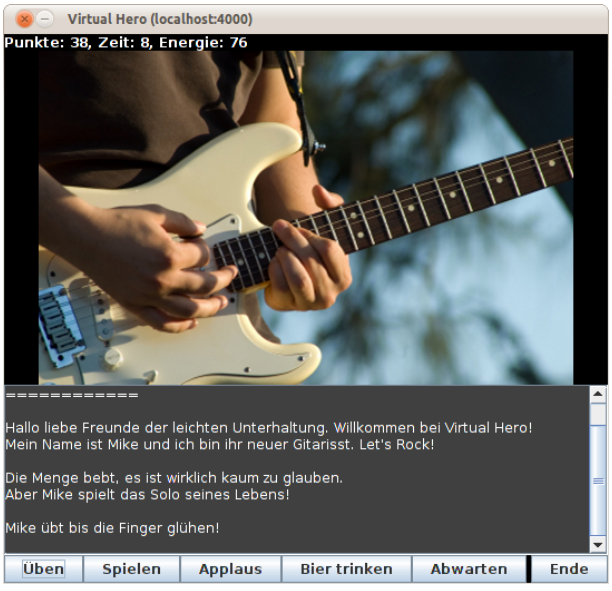
Empfangenes Kommando	**Reaktion des Servers**
Verbindungsaufbau	Antwortet mit `WELCOME/name`, um das Spiel zu beginnen.
`QUIT`	Sendet das Wort `FINISHED` an den Client und beendet die Kommunikation, indem die Methode `close()` des Clientsockets aufgerufen wird.
`SHUTDOWN`	Wie `QUIT`, zusäzlich wird aber die Variable `Main.quit` auf `true` gesetzt, um den Serverprozess zu beenden.

`STATUS`	Sendet das Kommando `CURRENT STATUS` zusammen mit dem Punktestand, vergangener Zeit und Energie der Spielfigur an den Client. Zusätzlich müssen Sie dein `D` senden, falls die Spielfigur tot ist. Wenn Sie schläft, senden Sie stattdessen ein `Y` und sonst ein `N`.
	
	Wenn die Figur schläft oder wach ist, sind das alle zu sendenden Argumente. Wenn die Figur tot ist, kommt noch ein Argument mit der Todesursache hinzu. Beispiele:
	
	`CURRENT STATUS/32/16/64/Y`
	`CURRENT STATUS/42/21/84/D/Er wusste zuviel ...`
	
	Die benötigten Werte erhalten Sie von folgenden Methoden des `GuitarPlayer`-Objekts: `int getScore()`, `int getTime()`, `int getStrength()`, `boolean isSleeping()`, `boolean isAlive()`, `String getDeathCause()`

Empfangenes Kommando	Reaktion des Servers
`WAIT`	Ruft die Methode `waitSomeTime()` der Spielfigur auf und sendet `WAITING` an den Client.
`PRACTICE`	Schickt `WORRY` zusammen mit einem mürrischen Kommentar an den Client, wenn der Gitarrist gerade schläft. Sonst wird die Methode `practice()` der Spielfigur aufgerufen und `PRACTICING` an den Client gesendet.
`PLAY`	Schickt ebenfalls `WORRY` und einen Kommentar an den Client, wenn die Spielfigur gerade schläft. Andernfalls wird die Methode `singAndPlay()` der Spielfigur aufgerufen und `SOLO OF MY LIFE` an den Client gesendet.
`APPLAUSE`	Ruft die Methode `String receiveApplause()` der Spielfigur auf. Falls die Methode einen Leerstring zurückgibt, soll `THANKS` an den Client gesendet werden, sonst `WORRY/antwortstring`.
`DRINK BEER/anzahl`	Falls die Spielfigur gerade schläft, wird `WORRY` zusammen mit einem ablehnenden Kommentar an den Client gesendet. Wenn die Figur wach ist, wird die Methode `drinkBeer(int amountBottles)` der Spielfigur mit der empfangenen Anzahl Flaschen aufgerufen und `I LIKE DUFF` gesendet.
Ungültiges Kommando	Sendet einen Fehler an den Client: `ERROR/Wie bitte?!`

Gruppe 2 (Client): Ihre Aufgabe besteht darin, eine Swinganwendung zu schreiben, die anstelle des Kommandozeilenclients verwendet werden kann, um das Spiel zu spielen. Die eigentliche Anwendung ist schon fast fertig, Sie müssen nurnoch den Teil schreiben, der das Kommunikationsprotokoll umsetzt.

Nehmen Sie zunächst folgende Änderungen an der Klasse `RemoteConnection` vor:



1. Erweitern Sie die Methode `connect()` so, dass sie eine neue Verbindung mit dem Spieleserver aufbaut. Zieladresse und Portnummer finden Sie in den Instanzattributen `hostname` und `port`.

2. Erzeugen Sie außerdem einen `BufferedReader` und einen `PrintWriter`, welche benutzt werden können, um Daten mit dem Server auszutauschen. Speichern Sie die beiden Objekte in den Attributen `fromServer` und `toServer`.

3. Programmieren Sie die Methode `callServer()` zuende, indem Sie am Ende der Methode eine Antwortzeile des Servers einlesen und diese in einem String-Array zurückgeben. Dabei müssen Sie den empfangenen Text bei jedem Schrägstrich trennen.

Programmieren Sie nun die Spielelogik der Klasse `GameController` fertig:

1. Innerhalb der Methode `doApplause()` soll zweimal die Methode `callServer()` des `connection`-Objekts aufgerufen werden, um die Befehle `APPLAUSE` und `STATUS` an den Server zu schicken. Das Ergebnis der Aufrufe soll jeweils an die Methode `executeResponse()` übergeben werden, um das Hauptfenster aufzufrischen. Dabei müssen alle auftretenden Exceptions abgefangen und mit der Methode `Main.showException()` angezeigt werden.

2. In der Methode `doDrinkBeer()` sollen die Befehle `DRINK BEER` und `STATUS` an den Server geschickt und die jeweiligen Antworten ausgewertet werden. Achten Sie darauf, dass Sie bei `DRINK BEER` zusäzlich die Anzahl Bierflaschen als String mitgeben. Auch hier müssen alle Exceptions abgefangen und angezeigt werden.

3. Analog zu den vorherigen Methoden sollen in `doWait()` die Kommandos `WAIT` und `STATUS` gesendet und deren Antworten ausgewertet werden. Auftretende Exceptions sollen ebenfalls angezeigt werden.

4. Programmieren Sie die Methode `executeResponse()` mit dem Kommunikationsprotokoll aus. Die Methode nimmt ein String-Array mit einer Antwort des Servers entgegen und aktualisiert daraufhin die Programmoberfläche. Dabei entsprechen der erste Eintrag des Arrays immer dem empfangenen Kommando und alle übrigen Einträge den angehängten Zusatzdaten. Setzen Sie das in der nachfolgenden Tabelle beschriebene Protokoll um. Den Namen der Spielfigur finden Sie im Attribut `this.playerName`.

Empfangenes Kommando	**Reaktion des Clients**
`WAITING`	Ruft die Methode `this.window.setImage("default")` auf, um das Hintergrundbild des Hauptfensters zu wechseln. Ebenso wird die Methode `println()` aufgerufen, um folgenden Text auszugeben: `Die Zeit vergeht ...`
`PRACTICING`	Wechselt zum Hintergrundbild `practice` und gibt folgenden Text im Hauptfenster aus: `≪Spielername≫ übt bis die Finger glühen!`

`SOLO OF MY LIFE`	Wechselt zum Hintergrundbild `solo` und gibt folgenden Text aus:
	`Die Menge bebt, es ist wirklich kaum zu glauben.`
	`Aber ≪Spielername≫ spielt das Solo seines Lebens!`

`I LIKE DUFF`	Wechselt zum Hintergrundbild `beer` und gibt folgenden Text aus:
	`≪Spielername≫ ist beschäftigt. Er trinkt gerade Duff Bier!`
	`Da kommt gerade der Duff Man herein und macht die Musik an.`
	`Diese Party wird wohl länger dauern ...`

`CURRENT STATUS`	Ermittelt den aktuellen Spielstand bestehend aus Punkten, Zeit, Energie und Status aus der Serverantwort und übergibt ihn an folgende Methode des Spielfensters:
	`setScore(String playerName, int score, int time,`
	`int strength, String status)`

Empfangenes Kommando	Reaktion des Clients

	Wenn die Spielfigur schläft, ist der Status auf `Y` gesetzt. In diesem Fall soll das Hintergrundbild `asleep` angezeigt werden. Ist die Figur hingegen tot, ist der Status `D` und es soll das Bild `dead` erscheinen. In diesem Fall beinhaltet die Serverantwort als letztes Argument noch die Todesursache, die wie folgt dargestellt werden soll:
	`~~~~~`
	`Es ist fürchterlich, aber ≪Spielfigur≫ ist tot.`
	`≪Todesursache≫`
	`~~~~~`
	
	Bei allen anderen Stati ändert sich der Hintergrund nicht. Achten Sie außerdem darauf, dass der Text nicht nochmal ausgegeben wird, wenn das Programm wieder eine Statusantwort empfängt. Zusätzlich müssen Sie einmalig die Methode `enableGameButtons(false)` des Hauptfensters aufrufen, um die Spielknöpfe nach dem Ende des Spiels zu deaktivieren.

`THANKS`	Gibt eine kurze Bedankung im Hauptfenster aus. Das Hintergrundbild bleibt bestehen und wird nicht gewechselt. `≪Spielername≫: Danke, danke! Kommen Sie bald wieder.`
`WORRY`	Gibt einen kurzen, verärgerten Kommentar der Spielfigur aus. Der Kommentar wird vom Server geliefert und soll wie folgt angezeigt werden. Genau wie bei einer Bedankung ändert sich das Hintergrundbild nicht. `≪Spielername≫ (verärgert): ≪Empfangener Zusatztext≫`
`ERROR`	Gibt folgenden Text zusammen mit der empfangenen Fehlermeldung im Spielfenster aus: `≪Spielername≫ (leicht irritiert): ≪Fehlermeldung≫`
Unbekanntes Kommando	Wirft eine `IOException` mit folgendem Fehlertext: `Protokollfehler (Unbekanntes Kommando): ≪Empfangenes Kommando≫`

Probieren Sie es aus: Lassen Sie erst Client und Server auf demselben Rechner laufen. Wenn dies funktioniert, führen Sie die beiden Programme auf unterschiedlichen Computern Ihrer Gruppe aus. Sie müssen hierfür nur den Wert `localhost` beim Aufruf des Clients durch die IP-Adresse des Rechners ersetzen, auf welchem der Server läuft. Probieren Sie auch, sich mit mehreren Rechnern und unterschiedlichen Clients gleichzeitig mit dem Server zu verbinden.