Team 4

Lucerne University of
Applied Sciences and Arts

HOCHSCHULE
LUZERN

[Stephan Stofer] [Julien Grüter] [Maurizio Hostettler] [Adrian Willi]

Message Logger

Version 1.0.3

System-Spezifikation

[Stephan Stofer] [Julien Grüter] [Maurizio Hostettler] [Adrian Willi]

Version

Rev.	Datum	Autor	Bemerkungen	Status
1.0.0	19.04.20	Alle	1. Version - Zwischenabgabe	Fertig
1.0.1	21.04.20	Alle	Korrekturen gemäss Feedback aus Zwischenabgabe	Fertig
1.0.2	04.05.20	Alle	Uhrensynchronisation / Zeitstempel	Fertig
1.0.3	18.05.20	Alle	RMI Komponente hinzugefügt	Fertig

Abbildungsverzeichnis

1. Systemübersicht

Es soll eine Logger-Komponente für ein Spiel entwickelt werden. Bei der Logger-Komponente handelt es sich um eine Software-Komponente, die einfach in Bestehende Java-Applikationen eingebunden werden kann. Eine Applikation kann durch einfache Methodenaufrufe Meldungen (Messages) Über die Logger-Komponente dauerhaft aufzeichnen. Die Meldungen aller Applikationen sollen in einemzentralen Logger Server in einem wohldefinierten Format gespeichert werden. Das Systemdesign ist durch den Auftraggeber vorgegeben:

1.1. Kontextdiagramm

Das untenstehende Kontextdiagramm gibt einen guten Überblick, was das System an sich beinhaltet, also was verändert und gestaltet wird, und was zum Kontext dieses Systems gehört. In diesem Fall können ziemlich klare Grenzen gezogen werden. Die Schnittstelle zwischen dem System und dem Anwender stellt die auf einem Computer laufende Game-of-Life Applikation dar, welche während des Spiels mit Hilfe des Systems loggt. Die Vorgaben zur Gestaltung des Systems wurden im Projektauftrag durch die Stakeholder definiert, wobei sich die Anforderungen während der Entwicklung ändern können.

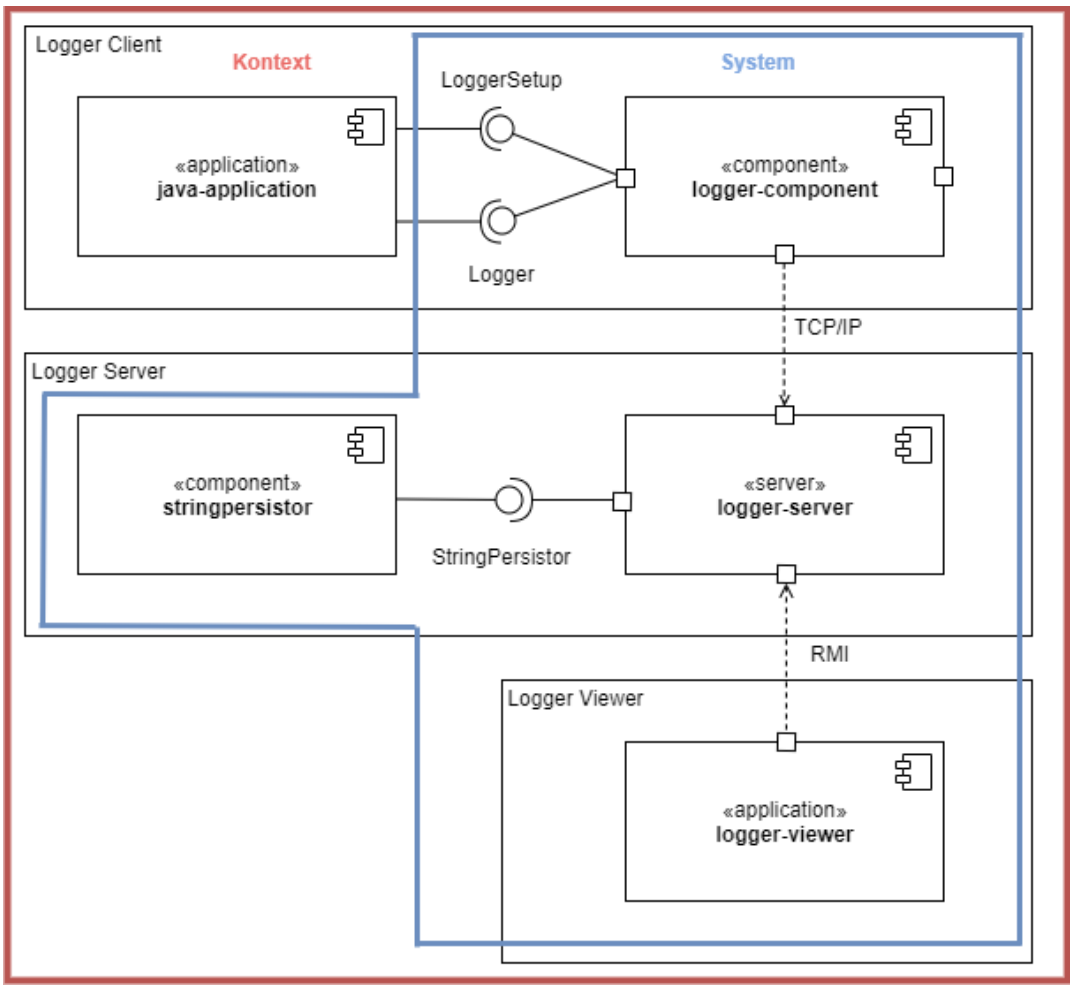

Abbildung 1 Kontextdiagramm

2. Architektur und Designentscheide

Die Architektur wurde grundlegend vom Auftraggeber vorgegeben. Diese können aus dem Dokument Projektauftrag entnommen werden. Einige davon sind auch bereits in Kapitel 1 erwähnt worden.

2.1. Modell(e) und Sichten

Nachfolgend einige Übersichten einzelner Komponenten.

2.1.1. LoggerSetup

1. Die Applikation initiiert ein LoggerSetup.
2. Die Applikation liest den FQDN der Logger Komponente aus der "config.properties" Datei.
3. Die Applikation erstellt via die Factory Methode ein LoggerSetup.
4. Da LoggerSetup lädt die nötigen Konfigurationsparameter aus dem "logger.properties"

alt IOException
Falls das LoggerSetup die "logger.properties" nicht findet, wird eine IO Exception geworfen und an die Applikation weitergegen.

alt Exception
Fall beim Initialisieren der Logger Komponente ein unerwartetes Verhalten auftritt, wird eine Exception geworfen.

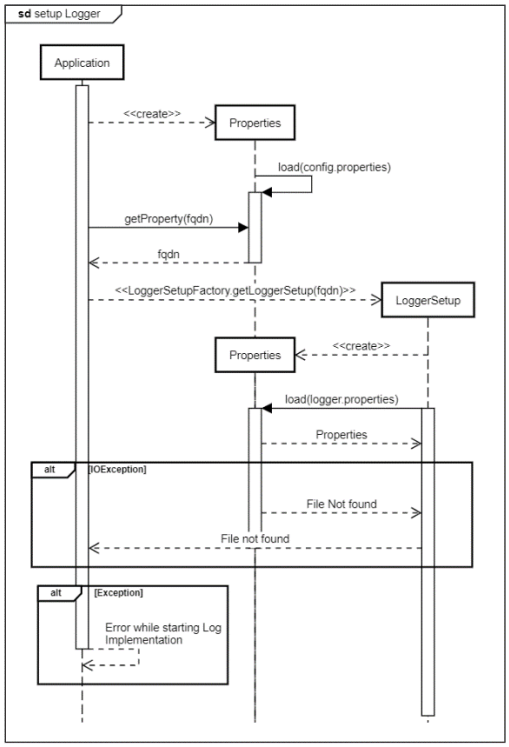

Abbildung 2 Sequenzdiagramm LoggerSetup

2.1.2. Logger

1. Die Applikation initiiert einen Logger mit getLogger(Name). Via dem LoggerSetup
2. Das LoggerSetup retourniert einen Logger

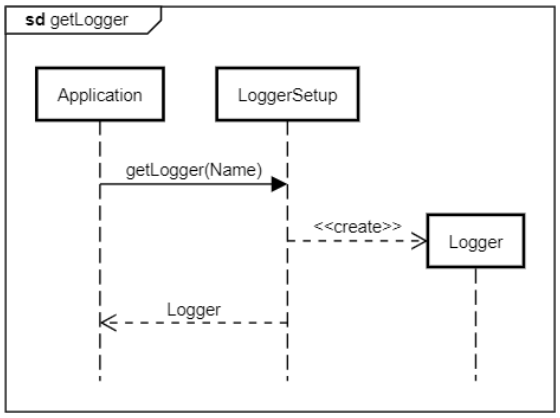

Abbildung 3 Sequenzdiagramm Logger #1

1. Die Applikation sendet eine LogMeldung mit debug(message)
2. Der Logger prüft ob das LogLevel effektiv aufgezeichnet werden soll.
3. Der Logger sendet die Logmeldung an den Log Server.

alt LocalLogger
Falls keine Verbindung zum Log Server besteht, werden die Logs lokal geschrieben.

alt Exception
Falls der Logger keine Verbindung zum Server aufbauen kann wird zusätzlich noch eine Exception geworfen.

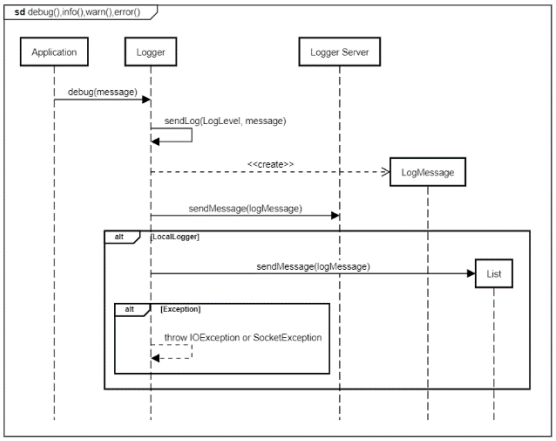

Abbildung 4 Sequenzdiagramm Logger #2

2.1.3. TCP/IP Kommunikation

Für die Kommunikation zwischen LoggerComponent und LoggerServer wird die Socket Kommunikation verwendet. Damit die Daten durch das Netz übertragen werden können, wird auf die Serialisierung zurückgegriffen. Die entsprechenden UML Diagramme sind im Abschnitt 5 (Klassendiagramme) ersichtlich.

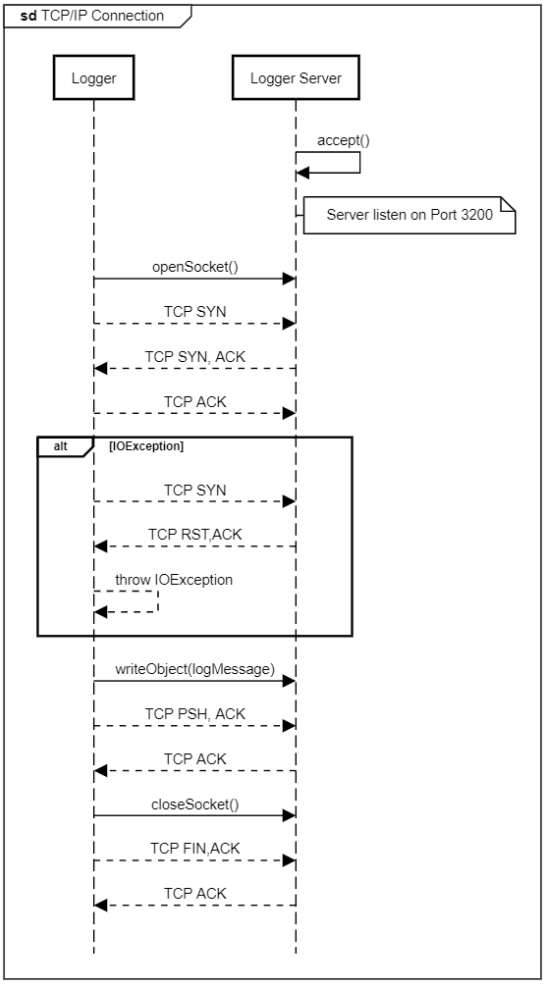

Abbildung 5 Sequenzdiagramm TCP/IP Connection

2.1.4. Logger Viewer

1. LogViewerClient erstellt ein CallbackHandler.
2. LogViewerClient macht ein lookup um den RMI-Service zu finden
3. LogViewerClient registriert sich beim RMI, um alle Logs per Stream zu erhalten.
4. LogViewerService fügt den LogViewerClient in seine Subscriberliste hinzu

Alt Exception:
LogViewerClient kann RMI Service Adresse nicht erreichen.

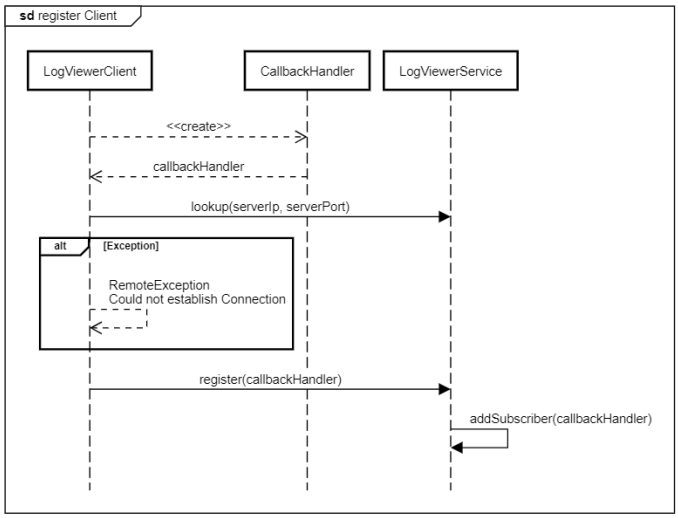

Abbildung 6 LogViewerClient Registration

1. LogViewerClient deregistiert sich beim LogViewerService
2. LogViewerService entfernt den LogViewerClient von seiner Subscriberliste.

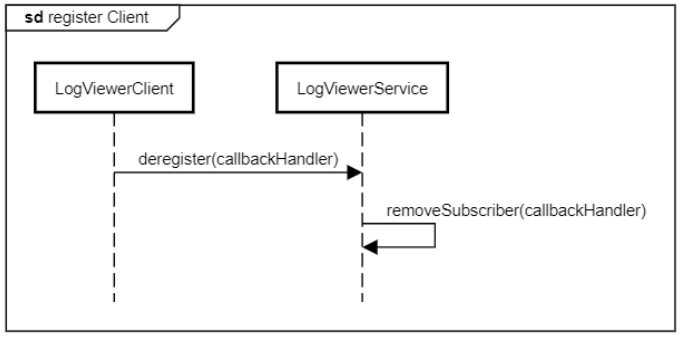

Abbildung 7 LogViewerClient Deregistration

1. LoggerServer registriert sich beim MessageHandler um über neue Logs informiert zu werden.
2. MessageHandler fügt den LoggerServer als Subscriber hinzu.
3. MessageHandler informiert den LoggerServer über neue Logs
4. LoggerServer informiert den LogViewerService über neue Logs.
5. LogViewerService sendet die Logs zu allen registrierten Clients.

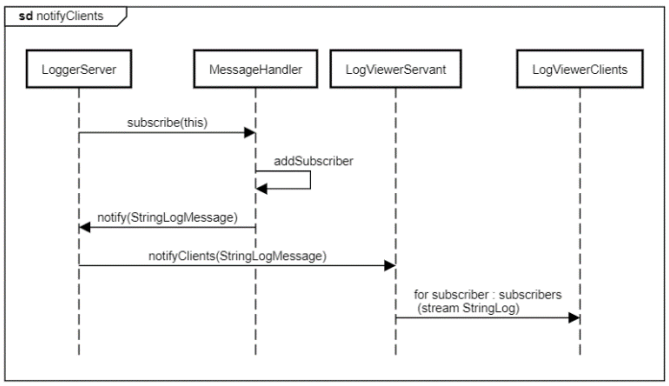

Abbildung 8 LogViewerClient Notification

2.2. Daten (Mengengerüst & Strukturen)

2.2.1. Struktur

Die Daten entsprechen einzelnen Logmeldungen, welche während der Laufzeit geschrieben werden. Die Struktur der Logmeldungen sind aus den Anforderungen abgeleitet:

Log [**Message**, **Log Level**, **Application ID, Time of Log**, **Time Server Received Log**]

Message: Die Logmeldung wird als Text

Log Level: Es bestehen die folgenden Level:

- OFF- Der Logger schreibt keine Logs zum Server
- ERROR - Unerwartetes Systemverhalten, welches zum Absturz führt.
- WARNING - Unerwartetes Systemverhalten, welches nicht zum Absturz führt.
- INFO – Erwartete, Anwenderzentrische Abläufe innerhalb der Applikation
- DEBUG - Wird für Analysezwecke der Entwickler benötigt.

Application ID: Eindeutige ID einer Applikation während der Laufzeit.

Time of Log: Entspricht dem Zeitpunkt der Erstellung des Logs.

Time Server Received Log: Zeitpunkt des Logempfangs auf Serverseite.

Die Klasse PersitedString dient als effektiven Träger der Log-Information zum Speichern und Auslesen von Logmeldungen durch den StringPersistor.

Zeitgangaben: Diese sind generell im Format UTC instant. (YYYY-MM-DDThh:mm:ss.ms

2.2.2. Mengengerüst

Es gibt kein Mengengerüst für die Logger Komponente. Es wird best effort angenommen und muss bei der individuellen Implementation getestet werden.

2.3. Entwurfsentscheide

2.3.1. Datenformat LogMessages

Die LogMessage-Klasse implementiert das Serializable-Interface um die Nachrichten per TCP/IP zwischen Client und Server über das Netzwerk auszutauschen. Wir verwenden das Interface als Vanilla Setup.

2.3.2. Adapter-Pattern (GoF)

Gemäss den Anforderungen des Auftraggebers musste das Adapter Pattern nach der GoF (Gang of Four) verwendet werden. Es beschreibt eine Adapter-Klasse, welche die Kooperation ansonsten inkompatibler Klassen und Interfaces ermöglichen. Dies geschieht in Java durch Erweiterung einer Klasse und/oder Implementierung eines Interfaces.

Das Pattern wird implementiert, um das Handling mit der LogDatei zu vereinfachen. Um die Funktion sicherzustellen, muss im Adapter die äquivalente Implementation des StringPersistor gegeben sein. Für die Übertragung der LogMessage zum StringPersistorFile sicherzustellen, wird das Adapter-Modell (StringPersistorAdapter) verwendet.

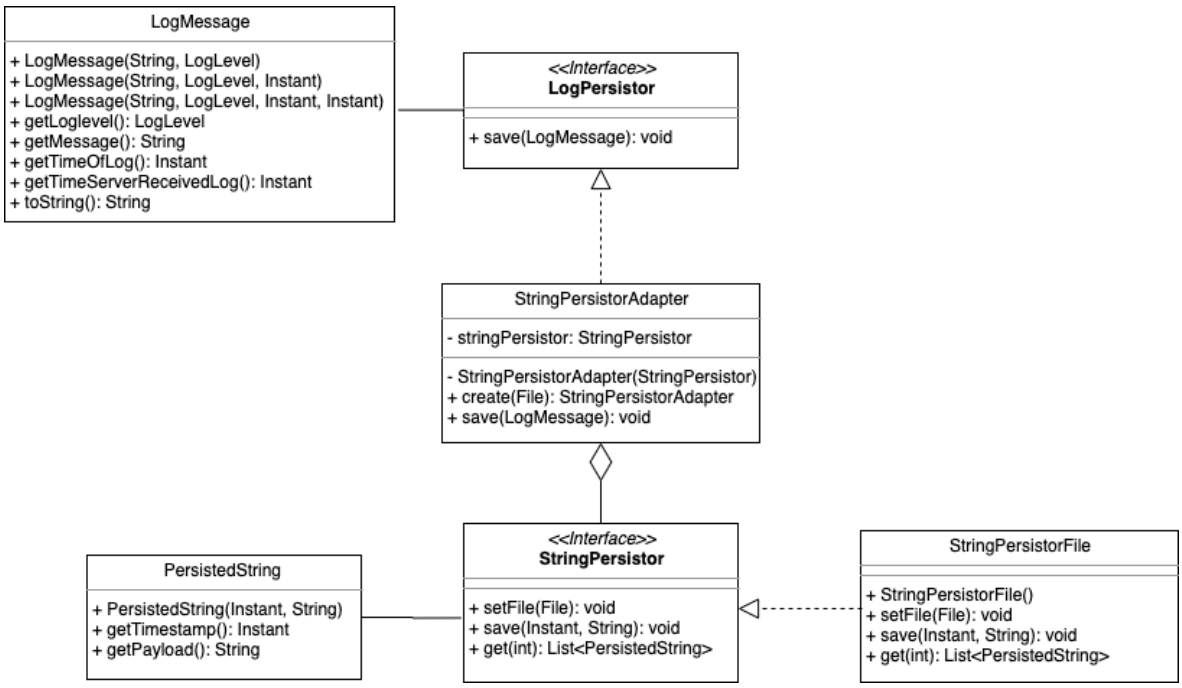

Abbildung 9 StringPersistorAdapter Klassen und Abhängigkeiten

2.3.3. Strategy-Pattern (GoF)

Das Speicherformat für das Textfile soll über verschiedene austauschbare Strategien (GoF-Pattern) leicht angepasst werden können. Wir haben uns dazu entschieden, das Format und allenfalls ein Delimiter aus einem System-Property beim LoggerServer auszulesen und dem StringPersistorAdapter in der Methode «create» als Parameter zu übergeben. Dieser erzeugt ein StringPersistorFile, welcher wiederum die gewünschte Strategy verwendet. Das Property-File wurde bewusst beim LoggerServer platziert, da dies erst dort verwendet wird und an einem anderen Ort nur zu einer höheren Kopplung zwischen Komponenten führen würde, die nicht gewünscht ist.

Für die Umsetzung haben wir uns für ein Interface mit zwei Methoden «serialize» und «deserialize» entschieden. Serialize erwartet ein PersitedString Objekt und kann ins gewünschte Fileformat serialisieren und übergibt es dem Bufferwriter zum Persistieren. Beim Auslesen des Files werden PersistedString Objekte zurückgegeben. Der grosse Vorteil an diesem Entscheid ist, dass die bestehende StringPersistorFile Klasse nur minimal angepasst werden muss, die Code Duplikation massiv verhindert werden kann und trotzdem die Variabilität und dadurch der Grundgedanke des Strategy Patterns erhalten bleibt.

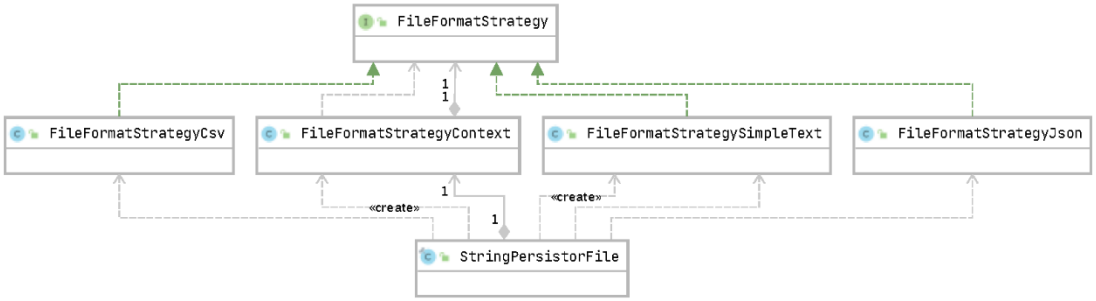

Abbildung 10 FileFormatStrategy Klassen und Abhängigkeiten

2.3.4. Konfigurationsdatei

Es werden drei Konfigurationsdateien genutzt. Beim Starten der Applikation werden die nötigen Konfigurationen aus den Dateien ausgelesen und angewendet. Spezieller ist die Konfigurationsdatei "loggerServer.properties", da sie beim Start des Servers gelesen wird und nicht direkt beim Start des Games.

Konfigurationsdatei	**Anwendung**
config.properties	Definiert welcher Logger für die Applikation genutzt werden soll.
logger.properties	Enthält Einstellungen für den Logger. • Server IP-Adresse oder FQDN des Logger Servers • Server Port des Logger Servers • Log Level
loggerServer.properties	Definiert Format mit welchem in das File geschrieben werden soll.

Als Beispiel zum Aufbau einer Konfigurationsdate dienen die folgenden Zeilen.

```
# logger.properties
logLevel=DEBUG
dstIpAddress=127.0.0.1
dstPort=3200
```

2.4. Logische Uhren / Zeitsynchronisation

Ein Einsatz im Projekt von logischen Uhren oder Zeitsynchronisation wäre zwischen den Komponenten Server – Client und Server – Viewer denkbar, jedoch nur bedingt sinnvoll.

Der Hauptvorteil, der sich durch eine Implementation einer synchronisierten Zeit zwischen Server und Client ergibt, ist eine lückenlose Betrachtung der Logs auf dem Server. Da jedoch

bereits die Serverzeit und die Clientzeit unabhängig von der Synchronisation in jedem Log ersichtlich sind, lohnt sich dieser Aufwand in Relation zum Nutzen nicht.

Der Einsatz von einer eigenen Zeitsynchronisation auf Server und Client würde zusätzliche Komplexität ins Projekt bringen, die auch bei Netzausfällen oder einem gestarteten Client ohne Serverzugriff klar durchdacht werden müsste. Zusätzlich findet bei uns keine Kommunikation vom Server zum Client statt, also ist die Problematik nochmals geringer, das Probleme nur in eine Richtung auftreten könnten.

Der Einsatz einer Logischen Uhr (Lamport Zeitstempel) wäre vom Aufwand her jedoch am besten tragbar. Der Einsatz im Projekt ist jedoch nicht vorgesehen, wie bereits oben erläutert wurde. Aus den Anforderungen geht heraus, dass die Logs “human readable” sein müssen. Denkt man sich die Situation von Lamport Zeitstempeln, die unter Umständen Sprünge machen, ist der Effekt auf eine Analyse der Logs tendenziell eher verwirrend als hilfreich. Durch die Identifier der Clients können diese isoliert betrachtet werden und die sequentielle Reihenfolge der Logs ist bereits gegeben.

Auch werden die Logs und somit die Zeitstempel nicht technisch weiterverwendet wodurch eine perfekte Abstimmung für konsumierende Systeme aktuell nicht nötig ist.

3. Schnittstellen

3.1. Externe Schnittstellen

Die externen Schnittstellen sind durch den Auftraggeber und durch ein Interface-Komitee vorgegeben. Die Dokumentation der Interfaces (Syntax, Semantik, Protokoll und Nichtfunktionale Eigenschaften) ist in der JavaDoc der jeweiligen Interfaces vorhanden.

Folgende externe Interfaces werden verwendet:

Interface	**Version**
Logger	1.0.0
LoggerSetup	1.0.0
StringPersistor API	5.0.3

3.2. Wichtige interne Schnittstellen

LogPersistor

Beim LogPersistor handelt es sich um die Schnittstelle, welche die StringPersistor-Schnittstelle für den Logger adaptiert (Adapter Pattern, siehe Abschnitt 2.3.2).

3.2.1. Struktur der Logdatei

Die LogMessage Klasse hat mehrere Konstruktoren, da die Nachricht beim Eintreffen auf dem Server noch mit einem Zeitstempel mit der Ankunftszeit ergänzt wird. Diese Anforderung sowie die weitern notwendigen Bestandteile der LogMessage werden im Projektauftrag explizit definiert und im Abschnitt 2.1.1 genau erläutert. Die Nachricht wird auf dem Client gespeichert, falls keine Verbindung mit dem Server hergestellt werden kann. Nachfolgend wird erläutert, wie die LogMessage auf dem Client sowie dem Server abgespeichert werden.

Client

Eine LogMessage-Instanz wird mithilfe eines Formatter formatiert. Als erstes kommt der Zeitstempel der Erstellung gefolgt von einem vertikalen Trennstrich «Pipe» (|). Anschliessend kommen das LogLevel sowie die eigentliche Nachricht. Es wird ein Formatter verwendet, damit die LogMessage-Instanzen zu einem späteren Zeitpunkt einfacher ausgelesen und an den Server gesendet werden können.

```
2020-04-18T14:43:39.4392731002 | [ERROR] Logging level ERROR
2020-04-18T14:43:41.528462Z | [WARN] Logging level WARN
2020-04-18T14:43:43.533462Z | [INFO] Logging level INFO
2020-04-18T14:43:45.5436018002 | [DEBUG] Logging level DEBUG
2020-04-18T14:43:47.7007552002 | [DEBUG] Start UI
2020-04-18T14:43:49.7077615002 | [DEBUG] Grid created
2020-04-18T14:43:51.7346545002 | [DEBUG] Game of Live as DropTarget
2020-04-18T14:43:53.7523311002 | [DEBUG] Start construction controls
2020-04-18T14:43:55.7718645002 | [DEBUG] End construction controls
2020-04-18T14:43:57.7768995002 | [DEBUG] End UI
2020-04-18T14:44:00.1271465002 | [DEBUG] Grid cleared
2020-04-18T14:44:05.1975218002 | [DEBUG] Start/Stop button clicked
2020-04-18T14:44:07.9186381002 | [DEBUG] Start/Stop button clicked
```

Abbildung 11 Logs clientseitig

Server

Um eine LogMessage-Instanz auf dem Server zu speichern, wird die toString-Methode der LogMessage Klasse verwendet. Der erste Zeitstempel sowie der vertikale Trennstrich «Pipe» sind vom StringPersistorFile, welcher ins File schreibt. Danach folgt die effektive LogMessage-Instanz im folgenden Format:

Log[message: die Message, logLevel: das LogLevel der Message, timeOfLog: der Zeitstempel der Erstellung, timeServerReceivedLog: der Zeitstempel der Ankunft der Message beim Server]

```
2020-04-18T15:17:52.3634093002 | Log[ message: Logging level DEBUG, logLevel: DEBUG, timeOfLog: 2020-04-18T15:17:52.2794196002, timeServerReceivedLog: 2020-04-18T15:17:52.2894129002 ]
2020-04-18T15:17:52.3634093002 | Log[ message: Logging level INFO, logLevel: INFO, timeOfLog: 2020-04-18T15:17:52.2774138002, timeServerReceivedLog: 2020-04-18T15:17:52.2904107002 ]
2020-04-18T15:17:52.3634093002 | Log[ message: Logging level WARN, logLevel: WARN, timeOfLog: 2020-04-18T15:17:52.2754125002, timeServerReceivedLog: 2020-04-18T15:17:52.2904107002 ]
2020-04-18T15:17:52.3624156002 | Log[ message: Logging level ERROR, logLevel: ERROR, timeOfLog: 2020-04-18T15:17:52.2584109002, timeServerReceivedLog: 2020-04-18T15:17:52.2894129002 ]
2020-04-18T15:17:52.452412Z | Log[ message: Start UI, logLevel: DEBUG, timeOfLog: 2020-04-18T15:17:52.4504113002, timeServerReceivedLog: 2020-04-18T15:17:52.452412Z ]
2020-04-18T15:17:52.4584109002 | Log[ message: Grid created, logLevel: DEBUG, timeOfLog: 2020-04-18T15:17:52.4554131002, timeServerReceivedLog: 2020-04-18T15:17:52.4584109002 ]
2020-04-18T15:17:52.4664126002 | Log[ message: Game of Live as DropTarget, logLevel: DEBUG, timeOfLog: 2020-04-18T15:17:52.4624116002, timeServerReceivedLog: 2020-04-18T15:17:52.4664126
2020-04-18T15:17:52.4914116002 | Log[ message: Start construction controls, logLevel: DEBUG, timeOfLog: 2020-04-18T15:17:52.4874112002, timeServerReceivedLog: 2020-04-18T15:17:52.4914116
2020-04-18T15:17:52.5034135002 | Log[ message: End construction controls, logLevel: DEBUG, timeOfLog: 2020-04-18T15:17:52.4974148002, timeServerReceivedLog: 2020-04-18T15:17:52.5024128
2020-04-18T15:17:52.5044112002 | Log[ message: End UI, logLevel: DEBUG, timeOfLog: 2020-04-18T15:17:52.4994119002, timeServerReceivedLog: 2020-04-18T15:17:52.5044112002 ]
2020-04-18T15:17:52.8374109002 | Log[ message: Grid cleared, logLevel: DEBUG, timeOfLog: 2020-04-18T15:17:52.8354088002, timeServerReceivedLog: 2020-04-18T15:17:52.8364094002 ]
2020-04-18T15:17:55.0336428002 | Log[ message: Next button clicked, logLevel: DEBUG, timeOfLog: 2020-04-18T15:17:55.0326402002, timeServerReceivedLog: 2020-04-18T15:17:55.0336428002 ]
2020-04-18T15:17:56.3666448002 | Log[ message: Start/Stop button clicked, logLevel: DEBUG, timeOfLog: 2020-04-18T15:17:56.3646447002, timeServerReceivedLog: 2020-04-18T15:17:56.3666448
2020-04-18T15:17:57.4066414002 | Log[ message: Start/Stop button clicked, logLevel: DEBUG, timeOfLog: 2020-04-18T15:17:57.4046418002, timeServerReceivedLog: 2020-04-18T15:17:57.4066414
```

Abbildung 12 Logs serverseitig

Speicherort der Datei

Die LogDatei wird als ".txt" Datei auf dem Client im Home-Verzeichnis des angemeldeten Users in einem erstellten Ordner Namens "vsk_group04_FS20" im Format "yyyy.MM.dd.HH.mm" abgespeichert, welches dem Zeitpunkt der Erstellung der Datei entspricht. Beim Start des Servers wird sofort im Home-Verzeichnis des angemeldeten Users ein Ordner namens "vsk_group04_FS20_Server" und dort die LogDatei im Format "yyyy.MM.dd" erstellt.

Für Windows Rechner entspricht dies: `C:\Users\%username%\vsk_group04_FS20\`
Für Unix Systeme hingegen `/home/$USER/vsk_group04_FS20/`

Client

Beim Starten des Games wird ein Socket erstellt und versucht eine Verbindung zum LoggerServer aufzubauen. Falls dies nicht möglich ist, werden die LogMessages lokal beim Client geloggt und gesendet, sobald eine Verbindung zum LoggerServer besteht.

Server

Der LoggerServer seinerseits hört auf einem vordefinierten Port auf Anfragen und erzeugt LogMessageHandler, welche dann Verbindung mit dem Client aufnehmen und schlussendlich die LogMessages persistieren. Dabei werden mehrere Threads verwendet, um ein hohes Aufkommen von Verbindungen managen zu können.

4. Environment-Anforderungen

Der Logger ist in der Programmiersprache Java geschrieben und unterliegt deshalb den Mindestanforderungen von Java.
Zur Verwendung kamen Java in der Version 11, Maven 3.3.9 und JUnit 5.6.0 Entsprechend müssen kompatible Systeme verwendet werden um das Produkt gegebenenfalls weiter zu entwickeln.

4.1. Hardware Anforderungen

CPU: Pentium 2 266 MHz Prozessor

RAM: 128MB

Speicher: 200MB

4.2. Betriebssystemanforderungen

Windows

- Windows 10 (8u51 and above)
- Windows 8.x (Desktop)
- Windows 7 SP1
- Windows Vista SP2
- Windows Server 2008 R2 SP1 (64-bit)
- Windows Server 2012 and 2012 R2 (64-bit)

Mac OS X

- Intel-based Mac running Mac OS X 10.8.3+, 10.9+
- Administrator privileges for installation
- 64-bit browser
- A 64-bit browser (Safari, for example) is required to run Oracle Java on Mac.

Linux

- Oracle Linux 5.5+1
- Oracle Linux 6.x (32-bit), 6.x (64-bit)2
- Oracle Linux 7.x (64-bit)2 (8u20 and above)
- Red Hat Enterprise Linux 5.5+1, 6.x (32-bit), 6.x (64-bit)2
- Red Hat Enterprise Linux 7.x (64-bit)2 (8u20 and above)
- Suse Linux Enterprise Server 10 SP2+, 11.x
- Suse Linux Enterprise Server 12.x (64-bit)2 (8u31 and above)
- Ubuntu Linux 12.04 LTS, 13.x
- Ubuntu Linux 14.x (8u25 and above)
- Ubuntu Linux 15.04 (8u45 and above)
- Ubuntu Linux 15.10 (8u65 and above)

- Browsers: Firefox

4.3. Systemanforderungen

Netzwerk

- Offener TCP-Zielport, Standard = 3200
- FQDN des LoggerServer muss dem DNS bekannt sein, Standard = localhost

Applikationen / Applikationsumgebungen

- Java Runtime Environment 11+
- 7zip

5. Klassendiagramme

5.1. GameOfLife

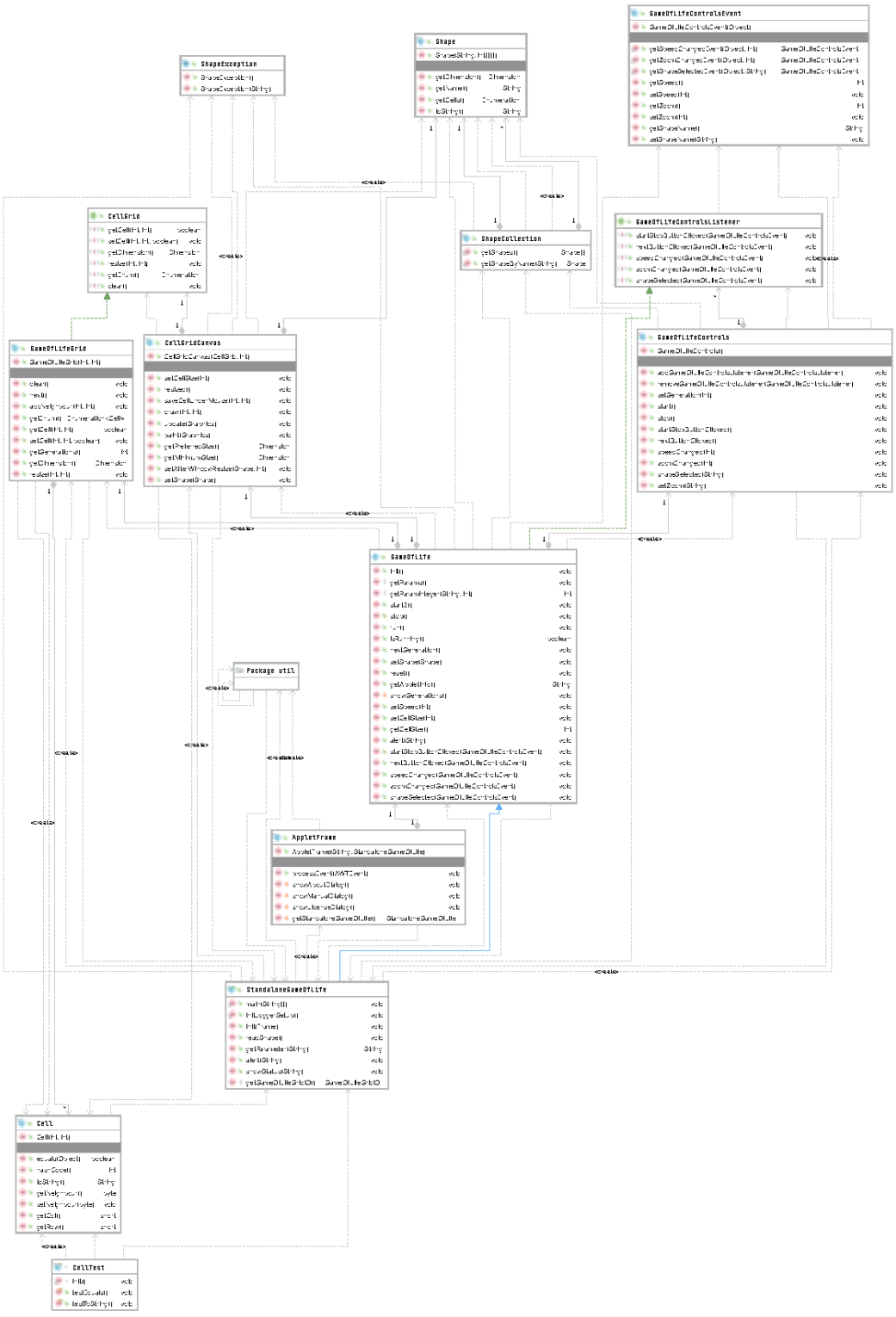

Abbildung 13 Übersicht Klassen der GameOfLife Applikation

5.2. Logger-Common

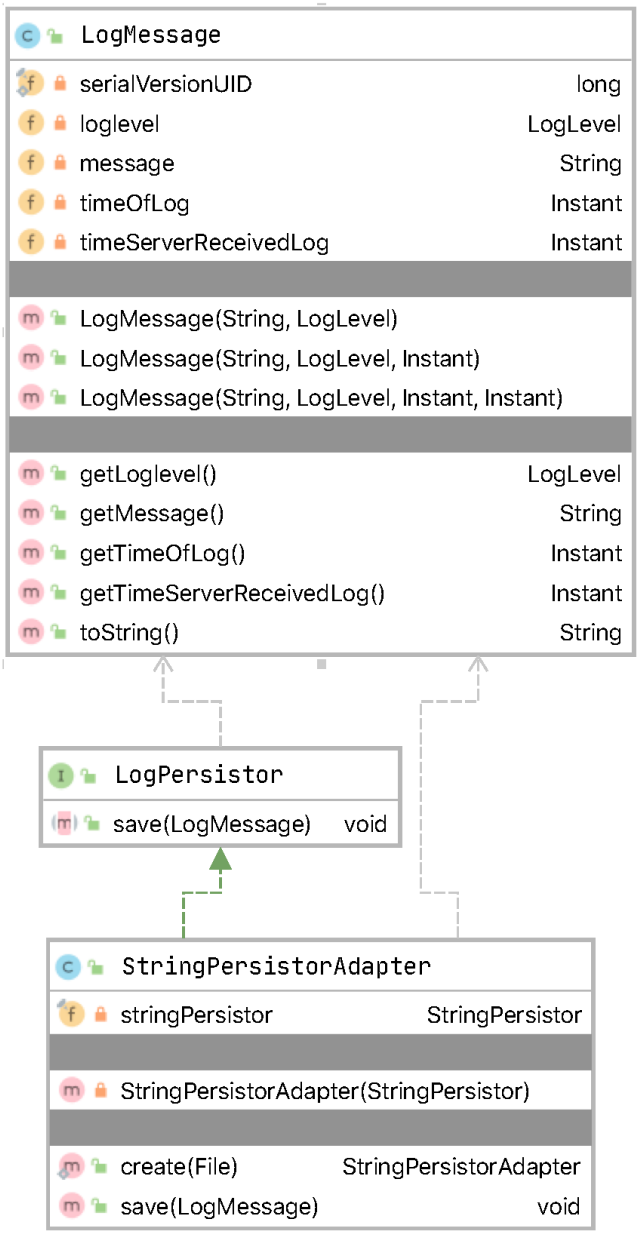

Abbildung 14 Logger-Common Klassen

5.3. Logger-Component

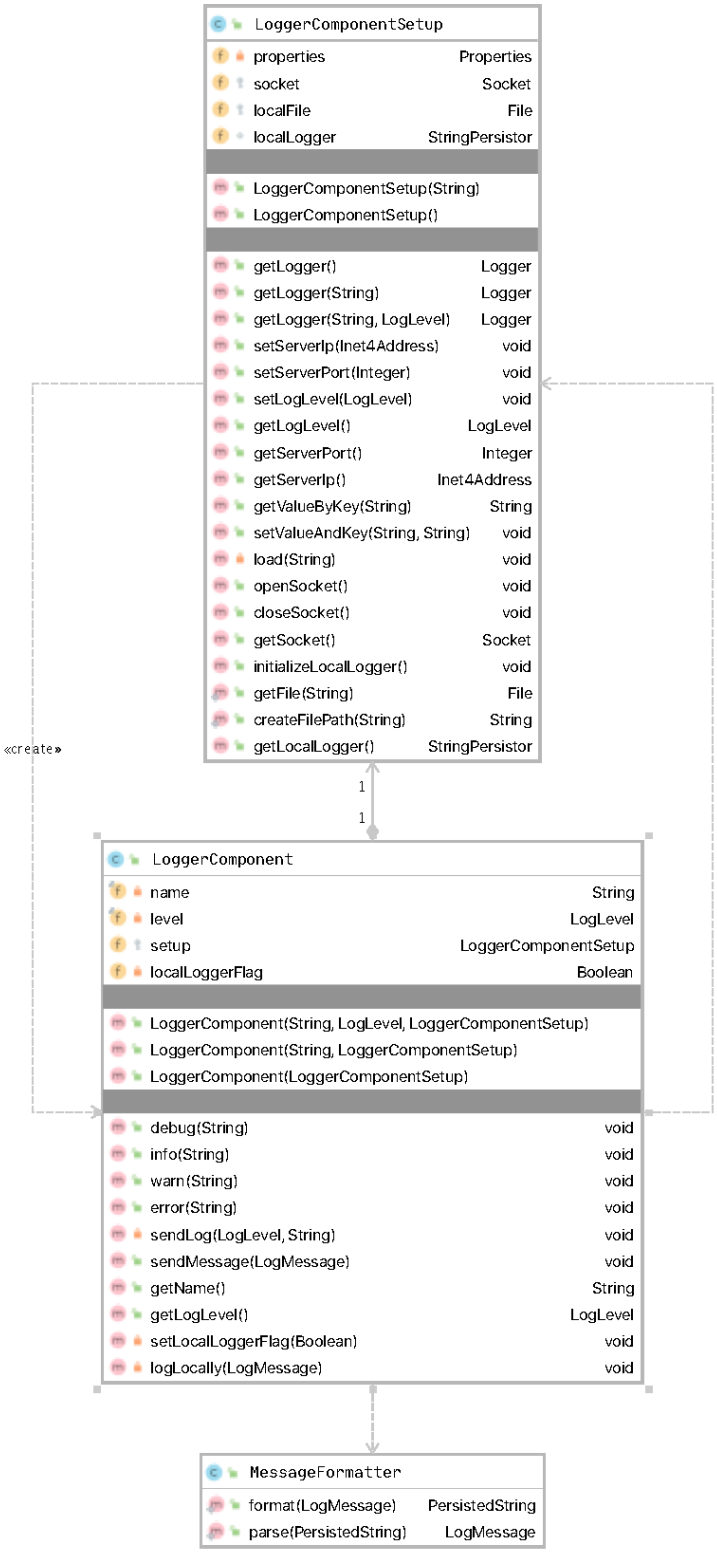

Abbildung 15 Logger-Component Klassen

5.4. Logger-Server

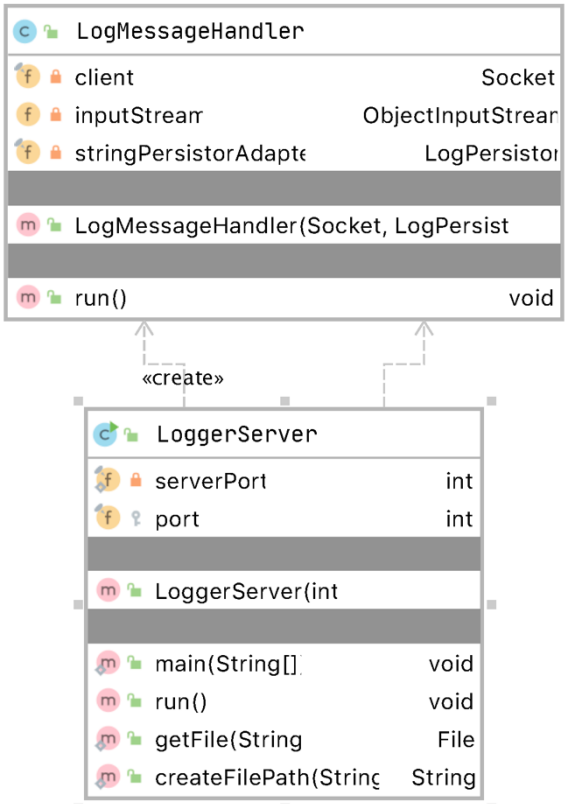

Abbildung 16 Logger-Server Klassen

5.5. StringPersistor

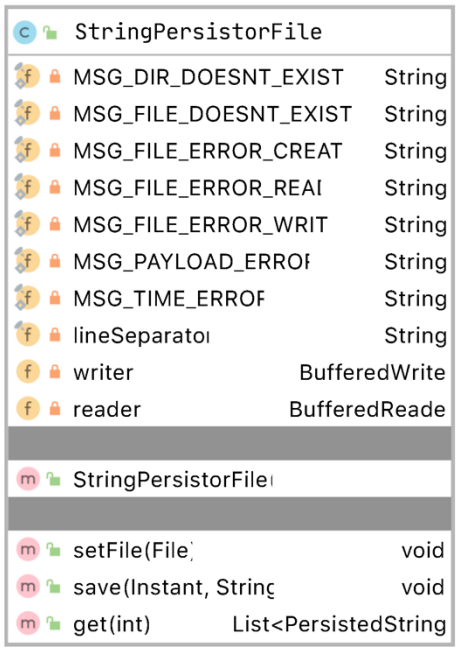

Abbildung 17 StringPersistor Klasse

5.6. Logger-Viewer

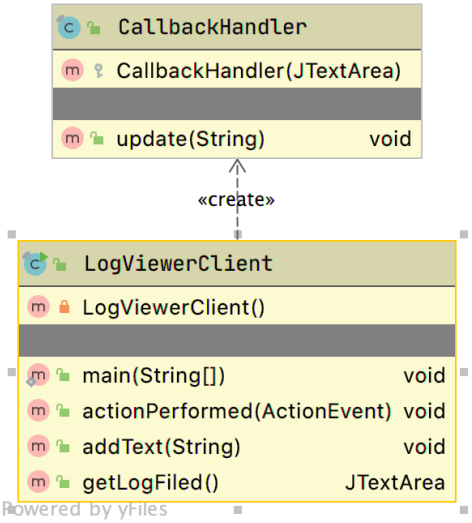

Abbildung 18 LogViewerClient Klasse