Lucerne University of
Applied Sciences and Arts
HOCHSCHULE
LUZERN

HOCHSCHULE LUZERN - INFORMATIK

Message-Logger System-Spezifikation

Verteilte Systeme und Komponenten

Amstutz Oliver, Brun Joel, Hunkeler Sandro, Leimgruber Dominik

28. Oktober 2020

Rev.	Datum	Autor	Bemerkungen	Status
0.1	05. Okt. 2020	Dominik Leimgruber	Grundstruktur erstellt	Fertig
0.2	10. Okt. 2020	Dominik Leimgruber	System- und Klassenübersicht dokumentiert	Fertig
0.3	13. Okt. 2020	Dominik Leimgruber	Logger-Komponente dokumentiert	Fertig
0.4	25. Okt. 2020	Dominik Leimgruber	Logger-Server dokumentiert	Fertig
0.5	26. Okt. 2020	Dominik Leimgruber	Event / Listener-Pattern dokumentiert	Fertig
0.6	26. Okt. 2020	Sandro Hunkeler	Daten (Mengengerüst & Strukturen), LoggerSetup und StringPersistorFile dokumentiert	Fertig
0.7	27. Okt. 2020	Dominik Leimgruber	Dokumentation der externen Schnittstellen	Fertig
0.8	28. Okt. 2020	Joel Brun	ClientSocket dokumentiert, TCP/IP Schnittstelle beschrieben	Fertig

ABBILDUNGSVERZEICHNIS

TABELLENVERZEICHNIS

INHALTSVERZEICHNIS

1 SYSTEMÜBERSICHT

1.1 Grobe Systemübersicht

Es soll eine bereits bestehende Java Game-Applikation mit einem voll funktionalen Logger erweitert werden. Dieser soll die Log-Einträge auf der Applikation-Seite entgegennehmen und danach über TCP/IP jene Events an einen Server senden. Der Server soll all diese Pakete entgegennehmen und in einem vordefinierten Format in eine dafür definierte Datei schreiben. Das System soll folgendermassen dargestellt werden:

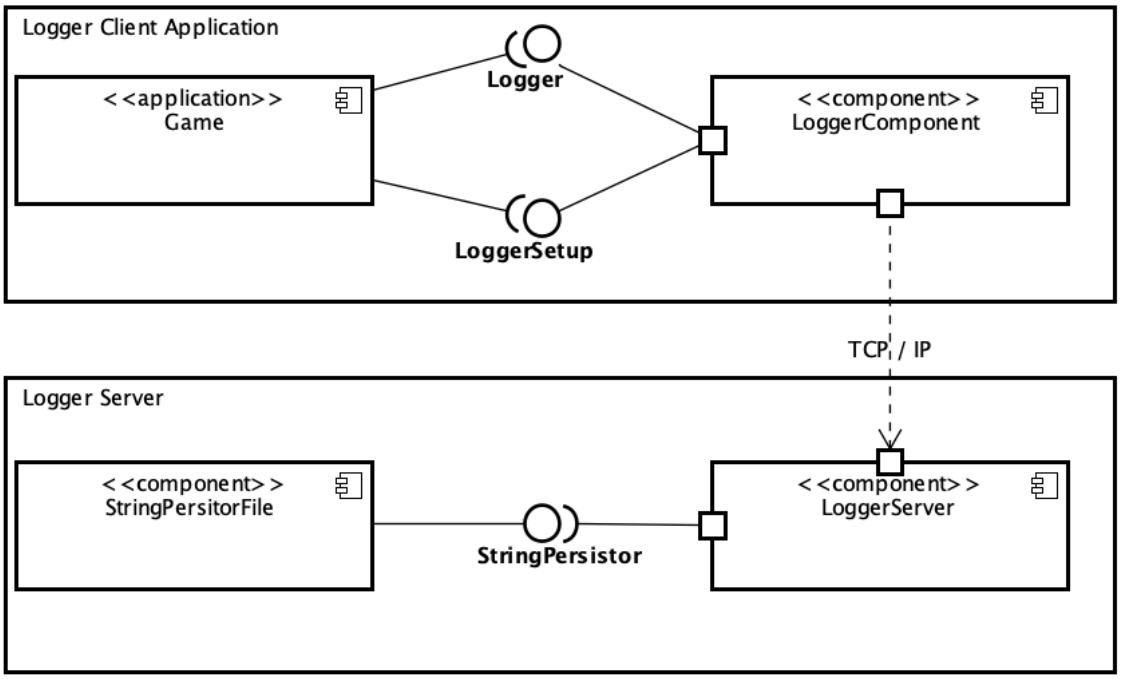

Abbildung 1.1: Komponentendiagram

2 Architektur und Designentscheide

2.1 Modell(e) und Sichten

2.1.1 Klassenüberischt

Die Klasse SimpleLoggerSetup implementiert das LoggerSetup-Interface und ist aufgrund dessen in der Lage ein Logger vom Typ des Logger-Interfaces zu konfigurieren. Der Logger wird durch eine Factory-Methode erstellt und an den Aufrufer zurückgeliefert. Der Logger selbst bietet nun die Möglichkeit die definierten Level zu loggen und besitzt eine Referenz auf den sogenannten Client-Socket, welcher durch den LoggerSetup erstellt wurde. Dieser Client-Socket stellt eine Verbindung über TCP/IP zum Server-Socket her. Somit ist es dem Logger möglich Log-Einträge an einen Server zu übermitteln. Der Logger-Server wird durch das Event Listener-Pattern über neue Log-Einträge informiert und kann diese verarbeiten. Zu guter Letzt werden die Log-Einträge über das StringPersistor-Interface und die StrinPersistorFile-Klasse in eine dafür vorgesehene Datei geschrieben.

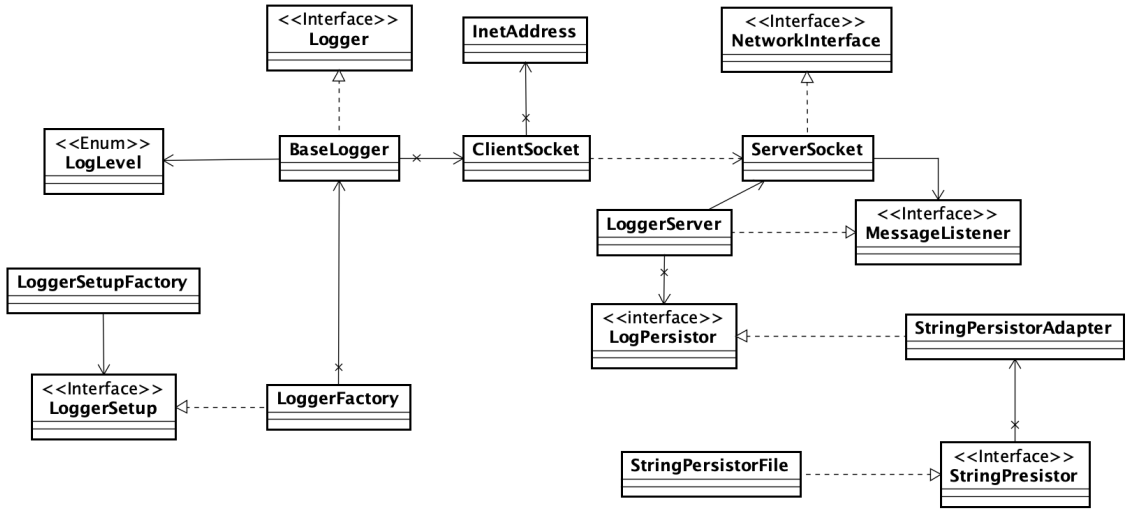

Abbildung 2.1: Klassendiagram

2.1.2 LoggerSetup

Die Klasse LoggerFactory implementiert das zur Verfügung gestellte Interface LoggerSetup. Aufgrund dessen besitz die Klasse alle Methoden, um als LoggerSetup fungieren zu können. In der öffentlichen Methode createClientSocket() wird ein ClientSocket instanziiert, welcher eine Verbindung zum LoggerServer über die im LoggerFactory angegebene IP und Port aufbaut. Die LoggerSetupFactory Klasse wird dazu verwendet, das LoggerSetup mittels createLoggerSetup() über einen String und später auch über eine Konfigurationsdatei zu erstellen.

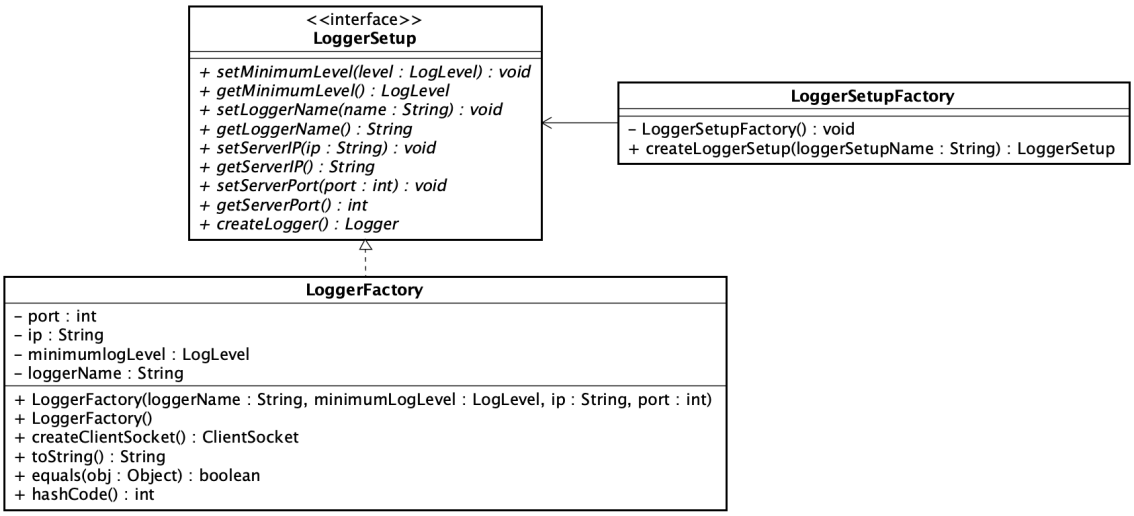

Abbildung 2.2: LoggerFactory Klassendiagram

2.1.3 Logger

Die Klasse BaseLogger implementiert das zur Verfügung gestellte Interface Logger. Aufgrund dessen besitzt die Klasse alle Methoden, um als Logger fungieren zu können. In der privaten Methode maybeLog() wird eine Überprüfung durchgeführt, welche mit Hilfe des minimalen Log-Levels entscheidet, ob eine Nachricht geloggt werden soll oder nicht. Soll eine Nachricht geloggt werden, wird in der Methode createLogMessage() ein Objekt der Klasse LogMessage instantiert und alle notwendigen Informationen diesem Objekt beigefügt. Schliesslich wird die Nachricht dem ClientSocket zum Senden übergeben. Da sie dort serialisiert wird, implementiert die LogMessage-Klasse das Serializable-Interface.

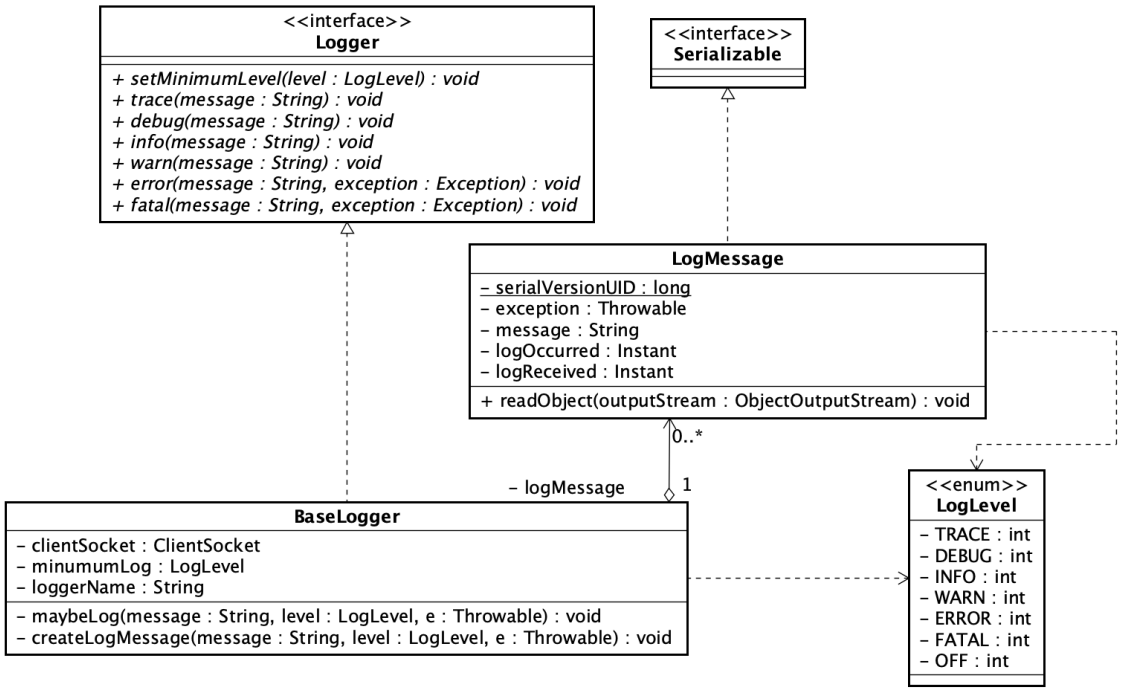

Abbildung 2.3: Logger Klassendiagramm

2.1.4 ClientSocket

Der ClientSocket ist zuständig für die Kommunikation mit dem Logger-Server. Diese Kommunikation läuft mithilfe einer Socket Instanz über TCP/IP ab. Die Methode sendData() öffnet die Socket-Verbindung und serialisiert die mitgegebene LogMessage, welche anschliessend gesendet wird. Zum Schluss wird die Socket-Verbindung wieder geschlossen.

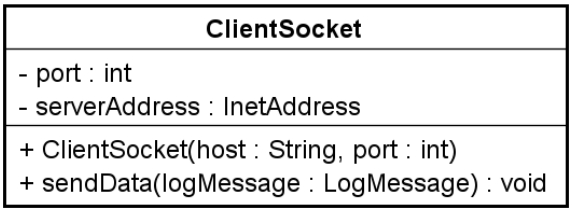

Abbildung 2.4: ClientSocket Klassendiagramm

2.1.5 Logger-Server

Der Logger-Server besteht aus zwei Klassen, dem Logger-Server und dem LoggerServerSocket. Der Logger-Server ist lediglich für die Weiterleitung der LogMessages an den StringPersistor zuständig und implementiert zusätzlich die main-Methode, um den Startpunkt des Servers zu definieren. Die Klasse LoggerServerSocket implementiert die benötigte Logik zur Entgegennahme der an den Server gesendeten Log-Messages. Die einzelnen Messages werden in einzelnen Threads parallel abgearbeitet und an den LoggerServer übergeben.

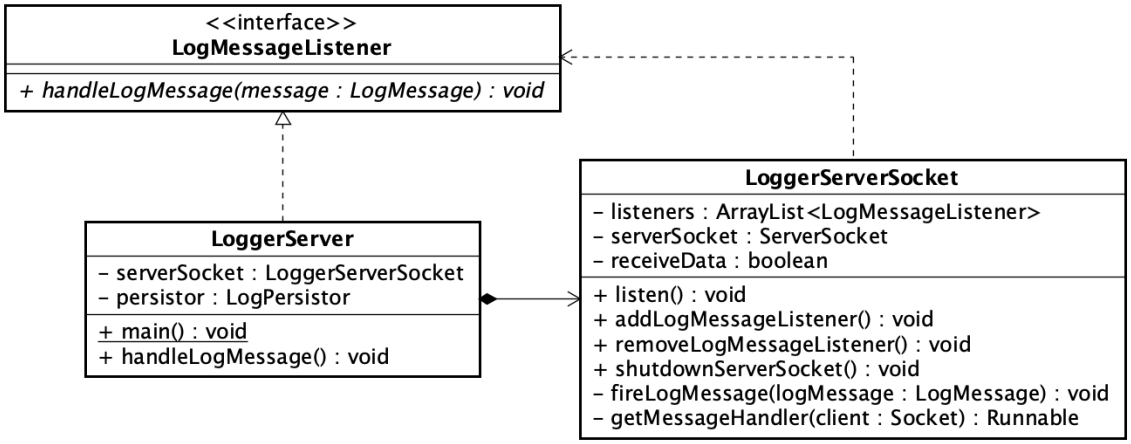

Abbildung 2.5: Logger-Server Klassendiagramm

2.1.6 StringPersistor

Die Klasse StringPersistorFile implementiert das zur Verfügung gestellte Interface StringPersistor. Aufgrund dessen besitzt die Klasse alle Methoden, um als StringPersistor fungieren zu können. Mit der Methode save() welcher über den Adapter zur Verfügung steht wird die LogMessage in einen String umgewandelt und mit dem entsprechenden Zeitstempel ins File geschrieben. Die Messages werden über einen printWriter in das File geschrieben. Über die Methode setFile() wird das Textfile erstellt und über die Methode get() können diese Files wieder zurückgegeben werden.

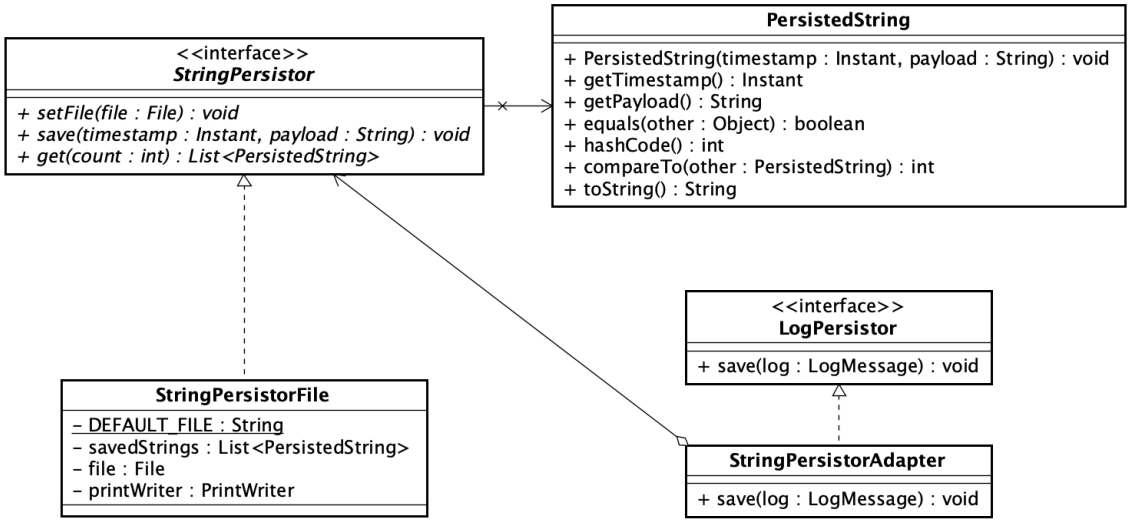

Abbildung 2.6: StringPersistor Klassendiagramm

2.2 Daten (Mengengerüst & Strukturen)

Zur Festhaltung der Daten werden sogenannte LogMessages generiert. Wie dies genau geschieht ist in der Abbildung 2.3 ersichtlich.

2.2.1 Mengengerüst

Da es sich um eine Logger Komponente handelt, die an diversen Stellen eingesetzt wird, kann das Mengengerüst nicht klar definiert werden.

2.2.2 Strukturen

Die Struktur der LogMessages, welche auch über den Adapter in Textfiles gespeichert werden sieht wie folgt aus:

- LogLevel, definiert welches Level dem Log entspricht:
  - Off
  - Trace
  - Debug
  - Info
  - Warn
  - Error
  - Fatal
- Message, gibt mittels eines Strings eine Meldung mit.
- LogOccured, gibt den Zeitstempel mit, wann das Log erstellt wurde.
- LogReceived, gibt den Zeitstempel mit, wann das Log auf dem LoggerServer empfangen wurde.

2.3 Entwurfsentscheide

2.3.1 Event / Listener Pattern

Das Event/Listener-Pattern wurde in Bezug auf den Logger-Server verwendet. Da der Logger-Server einen LoggerServerSocket beinhaltet, dieser jedoch dem Logger-Server die neuen Messages mitteilen muss, entsteht eine zirkuläre Beziehung zwischen diesen beiden Objekten. Um diese zirkuläre Beziehung aufzulösen und dadurch die Kopplung zu minimieren wird das Event / Listener-Pattern eingesetzt. Dadurch kommuniziert der LoggerServerSocket nur noch über das definierte Interface mit dem LoggerServer. Die komplette LoggerServer-Komponente ist unter Abbilung 2.5 ersichtlich.

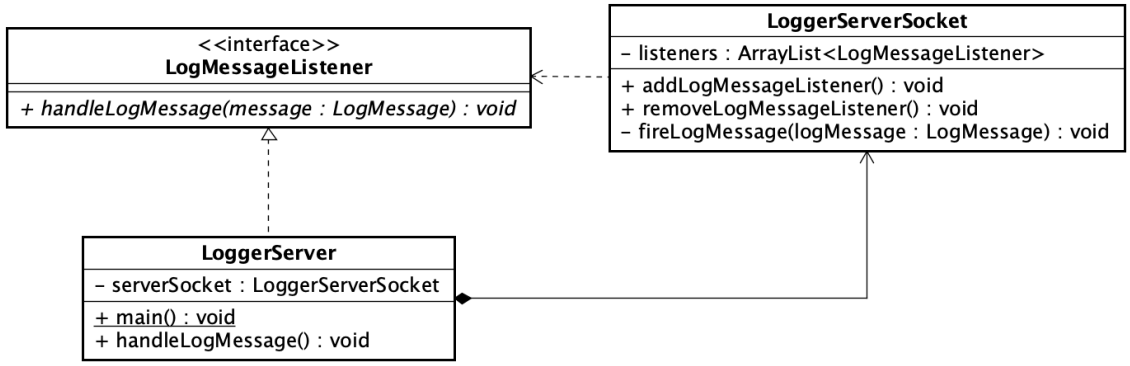

Abbildung 2.7: Event / Listener Pattern

2.3.2 Adapterpattern

Das Adapter Entwurfsmuster ist auf die String Persistor Schnittstelle angewendet. Dadurch kann die Komplexität der String Adapter Schnittstelle auf die benötigten Funktionen reduziert werden. Desweitern ist die neue Log Persistor Schnittstelle anwenderfreundlicher im Kontext des Loggers. Die LogPersistor Schnittstelle wird in Abbildung **??** dargestellt.

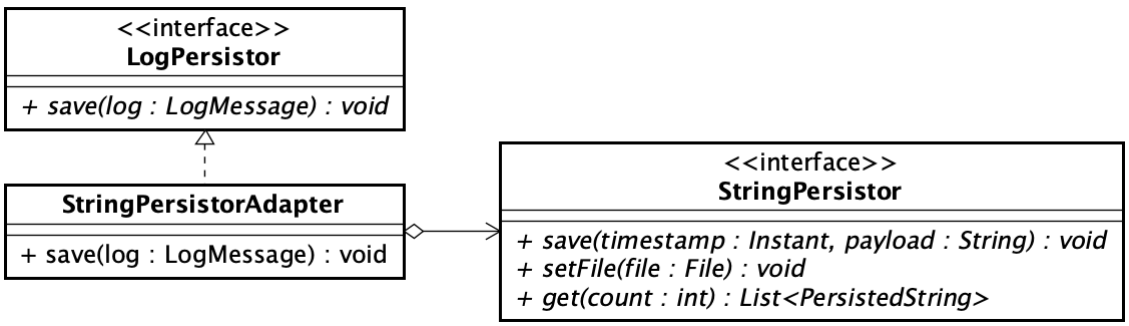

Abbildung 2.8: Adapterpattern

3 SCHNITTSTELLEN

3.1 Externe Schnittstellen

Die folgenden externen Schnittstellen sind im Projekt enthalten:

1. Logger
2. LoggerSetup
3. LogLevel
4. StringPersitor

3.1.1 Logger

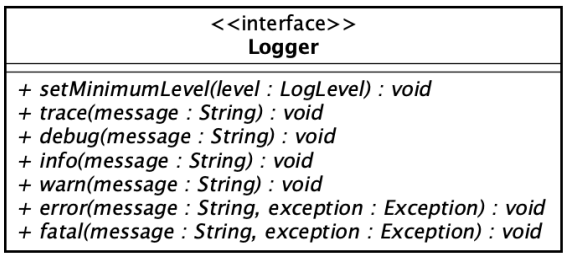

Abbildung 3.1: LoggerInterface Klassendiagram

Das Logger-Interface, über welches verschiedenste Applikationen Log-Einträge erfassen können enthält insgesamt acht Methoden. Für die einzelnen Log-Levels existieren eigene Methoden. Des Weiteren enthält das Logger-Interface eine Methode zur Setzung des minimalen Log-Levels.

3.1.2 LoggerSetup

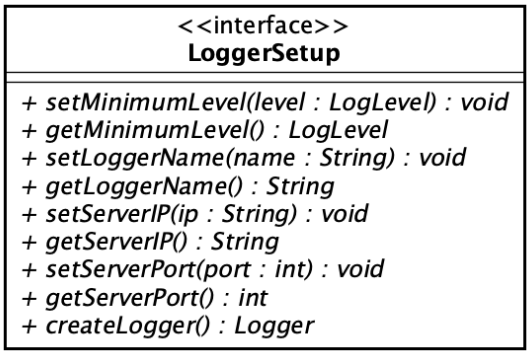

Abbildung 3.2: LoggerSetup Klassendiagram

Das LoggerSetup-Interface enthält neun Methoden, welche für die Erstellung eines voll funktionsfähigen Loggers benötigt werden. So muss definiert werden, wie die Server-IP-Adresse und dessen Port gesetzt werden kann. Des Weiteren kann bereits definiert werden, welcher minimale Log-Level der Logger zu Beginn besitzt. Der Logger-Name soll ebenfalls über eine Setter-Methode gesetzt werden können. Die wichtigste Methode ist die Factory-Methode createLogger, welche einen Logger vom Interfacetyp Logger zurückgeben soll.

3.1.3 LogLevel

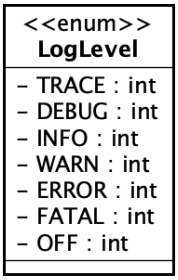

Abbildung 3.3: Log-Level Klassendiagram

Die Log-Level Enumeration definiert die möglichen Log-Level, durch die klare Definition der Log-Level ist ein Austausch der Logger-Komponente möglich. Die Enumeration enthält ebenfalls einen numerischen Wert, um die Überprüfung des minimalen Log-Level möglichst einfach und performant zu gestalten.

3.1.4 StringPersistor

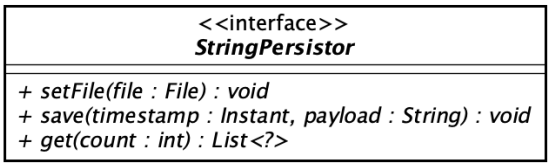

Abbildung 3.4: StringPersistor-Interface Klassendiagram

Das StringPersistor-Interface gibt Methoden zur Speicherung von Strings in eine Datei vor. Das Format dieser Datei ist jedoch nicht durch das Interface vorgegeben. Mit der setFile()-Methode soll definiert werden, in welche Datei die Strings gespeichert werden sollen. Die tatsächliche save-Methode verlangt einen Instant-Zeitstempel und den String der gespeichert werden soll. So soll jede String-Speicherung einer Zeit zugeordnet werden können. Da es sich um ein StringPersistor-Interface handelt, existiert auch eine Methode, die eine Liste von Strings wieder zurückgeben soll.

3.2 Interne Schnittstellen

Die folgenden internen Schnittstellen wurden durch das Projektteam definiert:

1. LogMessage
2. LogPersistor
3. TCP / IP
4. client.properties
5. server.properties

3.2.1 LogMessage

3.2.2 LogPersistor

3.2.3 TCP / IP

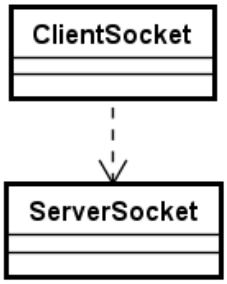

Abbildung 3.5: TCP/IP Schnittstelle

Das TCP/IP Protokoll wird verwendet, um die Kommunikation ziwschen dem ClientSocket und dem ServerSocket herzustellen. Der ServerSocket wartet in seiner listen() Methode auf einen ClientSocket und nimmt dessen Verbindung an.

3.2.4 client.properties

3.2.5 server.properties

4 Environment-Anforderungen

4.1 Hardware

Die Logger-Komponente braucht Zugriff auf ein Netzwerkinterface, um mit dem Server zu kommunizieren. Soll der Server mehrere Logger-Clients bedienen, ist es wichtig, dass der Server genügend Rechenleistung besitzt, um alle LogMessages verarbeiten zu können.

4.2 Software

Die Logger-Komponente und der Server laufen auf allen Betriebssystemen, welche von Java 11 unterstützt werden.

4.3 Java Virtual Machine

Damit Logger-Komponente und Server laufen, muss Java 11 installiert sein.

LITERATUR