Message Oriented Middleware

Programmentwurf

vorgelegt am 03. September 2024

Sommersemester 2024

CAS / DHBW Mosbach

Kurs T3M70306

Fortgeschrittene Konzepte Programmierung

von

Nils Hepp und Florian Böhm

1. Einleitung

[fl028/little\_mom: Assignment project Advanced concepts programming DHBW CAS SoSe24 (github.com)](https://github.com/fl028/little_mom)

In dieser Arbeit wurde eine Message Oriented Middleware (MOM) sowie deren Clients (Publisher und Subscriber) entwickelt. Die Publisher (P) senden dem Message Broker (MB) Nachrichten zu dem jeweiligen Thema. Der MB verteile die Nachrichten an die Subscriber (S), welche sich zuvor bei dem MB zu deren Wunschthemen anmelden müssen. Der Systemaufbau ist in Abbildung 1 dargestellt. Alle Programmteile laufen auf einem System, jedoch auf unterschiedlichen Threads. Hierfür mussten Synchronisation-Mechanismen implementiert werden. Der Programmentwurf wurde in der Programmiersprache Java implementiert.

A diagram of a diagram

Description automatically generated

Abbildung 1: Struktureller Aufbau des Systems

**Angewendete Paradigmen**

In der Arbeit wurden folgende Programmierparadigmen verwendet:

* Konsistente Formatierung des Codes
* Sinnvolle Bezeichner. Strukturiertheit, Klarheit, Ausführbarkeit Lesbarkeit
* Keep it simple
* Verwendung der kennenglernte Konstrukte aus der Vorlesung
* Englisch

1. Hauptteil

In dem Kapitel wird der Aufbau des Programmentwurfs kurz beschrieben.

**Aufbau**

Der strukturelle Aufbau des Programmes ist in Abbildung 2 dargestellt. Die auszuführende Datei ist „MOMSimulation.java“ (bzw. „MOMSimulation.jar“). Diese Klasse enthält die Main-Methode. In ihr werden die Objekte der weiteren Klassen sowie deren Threads erzeugt und verwendet. Zudem können hier über die konstanten „SIMULATION\_DURATION\_MS“ und „MAX\_MESSAGES“ variiert werden, um das Verhalten (Grund des Terminierens) zu ändern.

Der MB sowie die P und S besitzen jeweils eigene Klassen. Zudem gibt es ein Interface „Topic.java“, welches die Struktur der Themenklassen wie z.B. „FinanceTopic.java“ vorgibt. Alle Topic-Klassen implementieren dieses Interface. Zudem gibt es noch eine Klasse „Message.java“, diese wird versendet[[1]](#footnote-1). Sie enthält ein Topic-Objekt und ein String „content“, welcher die Nachricht enthält.

Ein Bild, das Text, Visitenkarte, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung

Abbildung 2: Strukturelle Aufbau des Programmes

**Klassenübersicht (der Hauptklassen MB, P, S)**

In Tabelle 1 wird eine kurze Übersicht über die Klassen MB, P und S gegeben. Die Klassen implementieren die nötigen Funktionalitäten, um eine Kommunikationsstruktur nach Abbildung 1 zu erhalten. Es wird das „Push – MB – Push“ Prinzip verwendet. Die Publisher ergreifen die initiative und pushen ihre Nachrichten zu dem MB-Objekt. Das MB-Objekt pusht die Nachrichten folgend an die entsprechenden Subscriber.

|  |  |  |
| --- | --- | --- |
| **MessageBroker** | **Publisher**  (implements Runnable) | **Subscriber**  (implements Runnable) |
| **Variablen**:   * Map<Topic, List<Subscribers>> subscribers * List<Message> messageQueue * Map<Topic, Integer> messageCountByTopic * boolean running * int maxMessages * int message | **Variablen**:   * int speed * MessageBroker broker * Topic topic | **Variablen:**   * String name * List<Message> messageQueue * Map<Topic, Integer> messageCountByTopic * Object lock |
| **Methoden:**   * synchronized void publish(Message) * synchronized void subscribe(Topic[], Subscribers) * void dispachMessage() * synchronized void stop() * synchronized int getMessageCount() * Map<Topic, Integer> getMessageCountByTopic() * List<Subscriber> getSubscribers() | **Methoden:**   * Static void SetSpeed(int) * void run() | **Methoden:**   * synchronized void receive (Message) * synchronized Map <Topic, Integer> getMessageCountByTopic * void run() |

Tabelle 1: Übersicht der Klassen

Die beiden Kommunikationspartner (Klasse P und S) implementieren das Interface Runnable und überschreiben deren run Methode (es wurde die Annotation @Override verwendet). Die Klasse MB besitzt ebenfalls eine Art run-Methode, und zwar dispatchMessage.

**Multithreading und Datenfluss**

Wie in Abbildung 1 dargestellt, existieren mehrere Instanzen der P und S Klassen, jedoch nur eine Instanz der MB-Klasse. Die dispatchMessage Methode des MBs sowie die run-Methoden der P und S Instanzen werden in Neben-Threads ausgeführt. Der Hauptthread wird die Zeit „SIMULATION\_DURATION\_MS“ schlafen gelegt und unterbricht dann danach die anderen Threads, um das Programm zu terminieren (vorausgesetzt Nachrichtenanzahl < MAX\_MESSAGES).

Der Message Broker besitzt eine Map mit Themen und Listen an Subscribern, um die entsprechenden Nachrichten an den entsprechenden Subscriber pushen zu können. Die Subscriber besitzen keine Referenz auf das Message Broker Objekt, der Message Broker kennt also die Subscriber, die Subscriber den Message Broker aber nicht. Den Publishern wird im Konstruktor das eine existierende Message Broker Objekt übergeben. Der Publisher kennt also den Message Broker, der Message Broker den Publisher aber nicht. Das liegt am Kommunikationsfluss. Der Publisher initiiert und pusht die Nachricht zu dem Message Broker und dieser zu dem Subscriber. Es handelt sich um eine Simplex Verbindung.

In Abbildung 3 ist der Datenfluss sowie die Aufteilung in verschiedene Threads, die gemeinsam genutzten Variablen und Datenbehälter sowie der verwendeten Lock-Objekte schematisch dargestellt. Alle P Instanzen verwenden das im Konstruktor übergebenen MB-Objekt als Lock in der run-Methode. In deren run-Methode wird die publish Methode des übergebenen MB-Objektes ausgerufen, wodurch die Daten übertragen werden. In einer ersten Version wurde die run-Methode der S Klasse nicht verwendet. Da dann aber das MB und das S Objekt immer auf dem gleichen Thread laufen würde, wurde das Programm verändert. Der MB pusht die Message in eine Warteschlange (Aufruf der S.receive Methode), in der run-Methode der S Klasse wird die Warteschlange zyklisch abgefragt und geleert. Somit ist ein wahres Multi-Threading implementiert.

Ein Bild, das Text, Screenshot, Schrift, Diagramm enthält.

Automatisch generierte Beschreibung

Abbildung 3: Datenfluss zwischen den Objekten sowie eine schematische Darstellung der Threads, gemeinsam genutzten Variablen und Lock-Objekten.

**Terminieren des Programmes**

Wie bereits beschrieben gibt es zwei Möglichkeiten das Programm ordnungsgemäß terminieren zu lassen. Zum einem, wenn die maximale Zeitdauer überschritten wird oder wenn die maximale Anzahl der Nachrichten überschritten wird. Die beiden Parameter sind in der Hauptklasse MOMSimulation festgelegt, sie können variiert werden.

Für den ersten Fall wird der Haupt-Thread die eingestellte Zeit schlafen gelegt. Im Folgenden werden die anderen Threads unterbrochen. Für den zweiten Fall wird in der publish Methode des MB-Objektes ein Zähler hochgezahlt, welcher die Nachrichtenanzahl darstellt. Ist die Nachrichtenanzahl größer als der eingestellte Parameter, werden die Threads terminiert. Hierfür sind in den entsprechend auszuführenden Methoden try catch statements, welche eine InterruptedException abfangen.

**Main Methode**

In der Main Methode wird ein MB-Objekt und jeweils drei P und S Objekte erzeugt und diese wie beschrieben verknüpft. Es werden insgesamt sieben weitere Threads erzeugt, in welchen die run Methoden von P und S sowie die dispatchMessage Methode von MB ausgeführt werden. Im Folgenden wird der Thread schlafen gelegt und wartet auf einer der beiden beschriebenen Abbruchbedingungen. Zuletzt werden alle Threads Ordnungsgemäß terminiert und folgend eine kleine Statistik und Zusammenfassung ausgegeben.

1. Es wird eine Klasse, welche das Interface Message implementier, anstatt beispielsweise ein String-Array versendet, um die Nachricht in einem „Container“ zu halten. [↑](#footnote-ref-1)