Team: 9, Tobias Heitmann, Moritz Spindelhirn

Aufgabenaufteilung: Die Aufgabe wurde gemeinschaftlich bearbeitet.

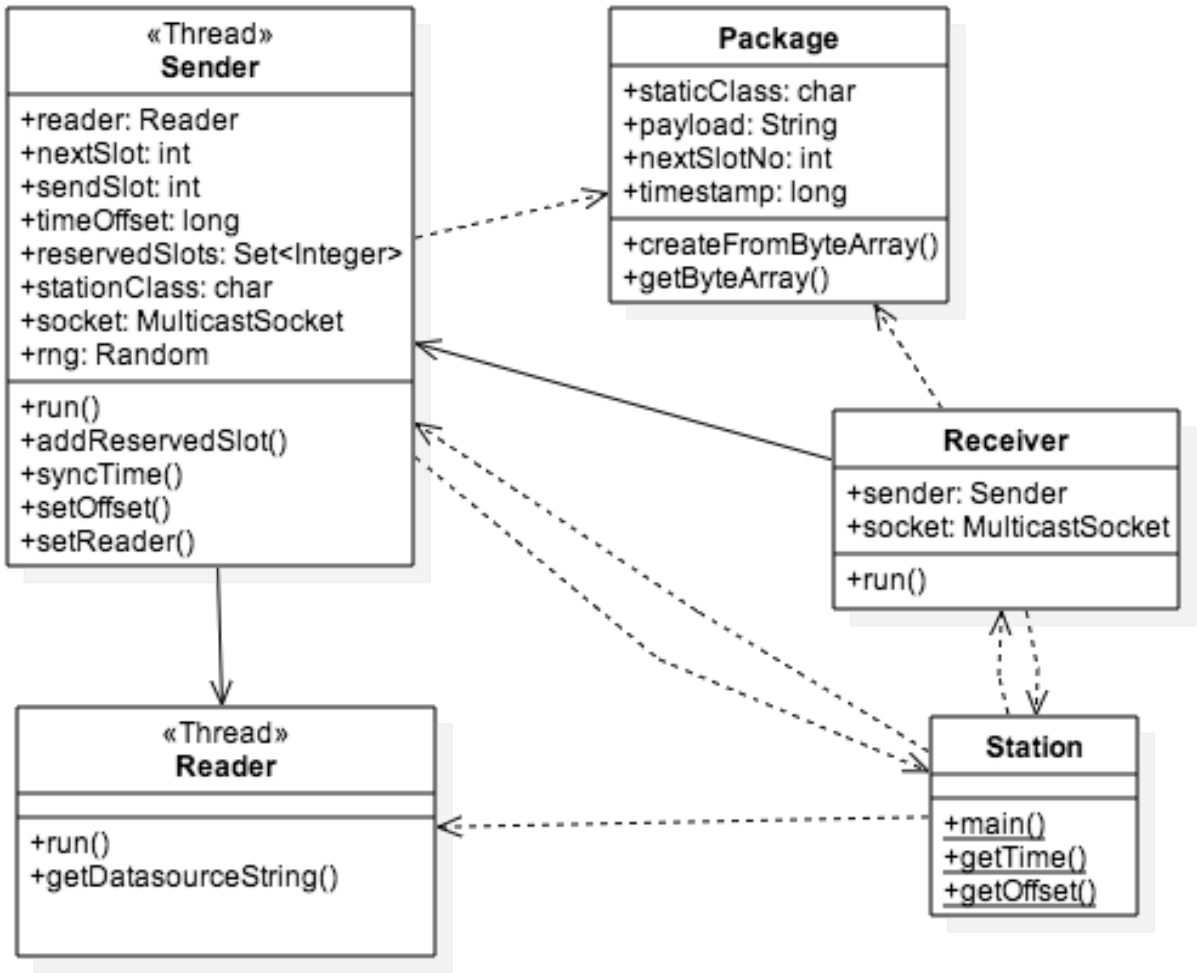
Quellenangaben: Es wurde kein Fremdcode übernommen.

Bearbeitungszeitraum:

21.06.2014: 5 Stunden
22.06.2014: 1 Stunde
23.06.2014: 2,5 Stunden
25.06.2014: 3 Stunden
04.07.2014: 2,5 Stunden

Aktueller Stand: Fertig bis auf einen unerklärlichen Bug bei dem der Sender ohne erklärbaren Grund einen Slot zu früh oder zu spät sendet. Dieser Bug folgt keinem Muster und tritt willkürlich auf.

Bestandteile



Zur besseren Aufteilung der Zuständigkeiten haben wir unser Programm in fünf Klassen unterteilt. Es handelt sich hierbei um die Klassen Station, Reader, Sender, Package und Receiver. Bei allen Klassen handelt es sich um richtige Java-Klassen in eigenen Dateien. Die Klassen Receiver, Sender und Reader sind eigene Threads und sind so in der Lage auf die äußeren Einflüsse zu reagieren.

Reader

Der Reader ist eine recht simple Klasse, deren einziger Zweck darin besteht die letzte Ausgabe der DataSource dem Programm zur Verfügung zu stellen. Beim Reader handelt es sich um einen eigenständigen Thread.

Station

Die Station ist die main/starter Klasse des Programms. Sie enthalt die main-Funktion, die zum starten des Programms benötigt wird. In ihr werden alle nötigen Objekte initialisiert und mit einander bekannt gemacht, außerdem stellt die Station zwei wichtige Hilfsfunktionen zur Verfügung. Hierbei handelt es sich einmal um eine Methode, die die aktuelle Zeit ausgibt und um eine, die den Offset eines mitgelieferten Timestamps zur aktuellen Stationszeit ermittelt.

Package

Beim Package handelt es sich um eine Klasse, die dazu da ist einen einkommenden Bytestrom in ein Packet umzuwandeln oder Packetinformationen in ein Bytestrom umzuwandeln. Außerdem stellt sie getter und setter für die Packetinformationen zur Verfügung.

Receiver

Der Receiver ist läuft in einem eigenem Thread ab. Er lauscht auf dem MulticastSocket nach Nachrichten von anderen Stationen. Diese Nachrichten werden dann zu einem Package umgewandelt und weiterverarbeitet. Der in dem Packet vorgegebene ReservedSlot wird im reservedSlots Set des Senders vermerkt. Wenn es sich bei der Quelle des Pakets um eine Station des Typs A handelt wird hier auch die Anpassung der Systemzeit gestartet.

Sender

Die Aufgabe des Senders besteht darin, die Pakete im vorgegebenen Der Sender besteht zum Hauptteil aus zwei Threads, der eine wartet immer genau ein Frame ab um dann einen anderen zu starten, der die Slots abwartet und dort überprüft, ob er bereits senden kann und, ob der Slot, den er als Slot angeben will, wo er darauf sendet nach wie vor frei ist, um eventuell einen neuen Slot auszuwählen.

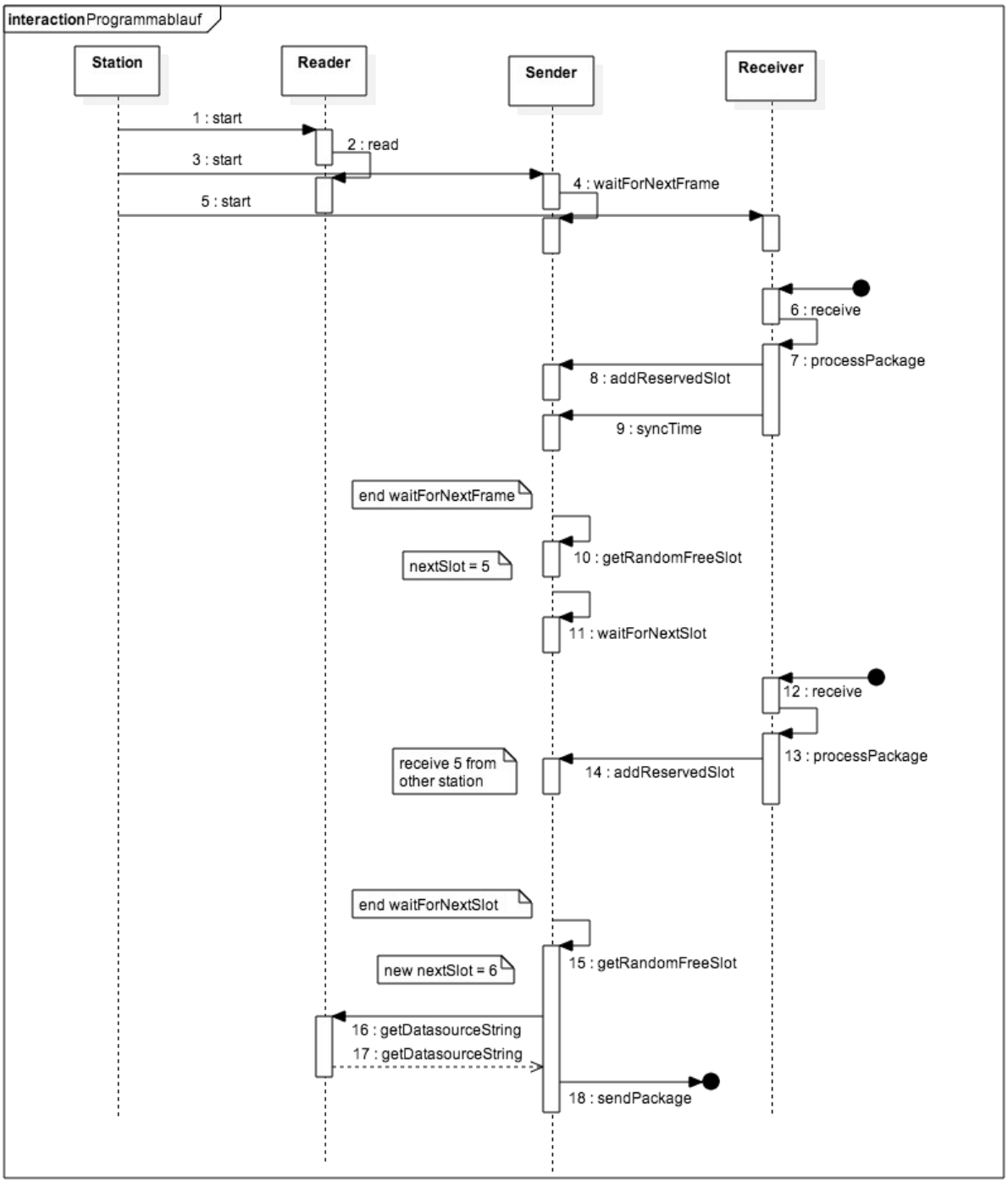
Programmablauf

Allgemein

Nach dem Start der Station erstellt diese als erstes einen Reader, der auf System.in lauscht, dann werden die nötigen Sockets erstellt und initialisiert. Diese Sockets werden dann inklusive anderer Programmparameter und Objektreferenzen an die nun ebenfalls erstellten Receiver und Sender Objekte weitergegeben. Als letzte Aktion der main-Funktion werden Receiver und Sender gestartet.

Der für das Programm wichtige Teil spielt sich nun im Senderobjekt ab. Nachdem dieses von der Station gestartet wurde, wartet es als erstes den nächsten Frame ab.
Darauf wird jedes Frame ein extra Thread gestartet, dessen Aufgabe es ist genau dieses eine Frame zu behandeln und dann ein Paket zu senden, wenn der entsprechende Slot dran ist. Der Hauptthread wartet derweil auf das nächste Frame um dann die vorgemerkten Reservierungen zu löschen und einen neuen Thread für dieses Frame zu starten.

Gleichzeitig läuft der Receiver, der ankommende Datagramme in Packages umwandelt um dann die wichtigen Informationen erhalten zu können. Dazu gehört in welchem Slot diese Station als nächstes senden wird, da dies für den Sender wichtig ist, wird diese Information in das reservedSlots Set des Senders eingetragen. Wenn der Receiver feststellt, dass er ein Paket einer Station der Klasse A erhalten hat, dann startet er eine Methode zum annähern der eigenen Stationszeit an die Zeit des Absenders.


interaction Programmablauf
Station
Reader
Sender
Receiver
1 : start
2 : read
3 : start
4 : waitForNextFrame
5 : start
6 : receive
7 : processPackage
8 : addReservedSlot
9 : syncTime
end waitForNextFrame
nextSlot = 5
10 : getRandomFreeSlot
11 : waitForNextSlot
12 : receive
13 : processPackage
receive 5 from other station
14 : addReservedSlot
end waitForNextSlot
new nextSlot = 6
15 : getRandomFreeSlot
16 : getDatasourceString
17 : getDatasourceString
18 : sendPackage

Lesen der DataSource

Um die Daten der DataSource verwenden zu können, benutzen wir unsere Reader Klasse, diese wartet immer wieder auf neuen Input am System.in. Dieser Input wird dann gespeichert und ist per getter abfragbar. Um Zugriffsprobleme zu verhindern, arbeiten getter und setter für den Speicher des letzten Inputs mit einem ReentrantLock.

Nachrichten anderer Stationen empfangen

Der Receiver, der fürs empfangen der Nachrichten anderer Stationen zuständig ist, wartet aktiv auf neue Datagramme, die dann unter Zuhilfenahme einer statischen Methode der Package Klasse namens createFromByteArray(data) in ein Package umgewandelt werden. Dann wird überprüft welcher Slot als reservedSlot in der Nachricht eingetragen ist, welcher dann dem reservedSlots Set des Senders hinzugefügt wird, damit dieser weiß, dass jener Slot bereits besetzt ist. Auch stößt der Receiver die Annäherung der eigenen Systemzeit an die des Absenders an unter der Bedingung, dass es sich beim Absender um eine Station der Klasse A handelt.

Senden von Nachrichten

Der Sender ist als eigener Thread programmiert, dieser wartet immer das nächste Frame ab um dann einen SlotThread zu starten, welcher dann innerhalb dieses einen Frames guckt, ob er bereits im richtigen Slot zum senden ist und, ob der Slot, den er für das nächste Frame reservieren will, bereits besetzt ist, wenn beides nicht zutrifft wartet er auf das nächste Frame.
Falls der zu reservierende Slot bereits belegt ist, wird eine Liste an noch nicht belegten Slots erstellt und von diesen zufällig ein Slot ausgewählt, welcher nun angepeilt wird.
Wir haben den Ansatz mit zwei Threads gewählt, um dem Problem aus dem Weg zu gehen, dass es eventuell länger brauchen könnte, bis der SenderThread etwas abgearbeitet hat und so immer wieder größere Offsets zu allen anderen Stationen erreichen könnte. Da es aber immer nur einen SlotThread gibt, der arbeitet und dieser nur ein Frame lang arbeitet, wird der SenderThread immer zum Beginn eines Frames einen neuen Thread erstellen, auch wenn der vom vorherigem Frame nach wie vor arbeitet, aber nicht mehr im richtigen Takt ist.