Technische Hochschule Rosenheim, Fakultät für Informatik

Vorlesung: Verteilte Verarbeitung SS 2019 Prof. Dr. Gerd Beneken

Praktikum 1 von 3 – Praktikum als Grundlage für ein Kolloquium

Ziel dieses Praktikums

Dieses Praktikum hat zwei Ziele, erstens sollen Sie nach erfolgreichem Abschluss aller Praktikumsaufgaben verteilte Systeme besser verstehen und diese praktisch umsetzen können. Zweitens sollen Sie möglichst viele praktische Erfahrungen beim agilen Programmieren sammeln, damit sie das Praxissemester möglichst erfolgreich überstehen.

Ziel ist es, dass Sie während des Programmierens möglichst umfangreich Feedback zu Ihren Ergebnissen erhalten und zwar von dem betreuenden Professor und auch von Ihren KommilitonInnen. Je mehr Feedback Sie erhalten, desto mehr und besser lernen Sie. Ich hoffe, dass dadurch auch Ihre Motivation steigt, sich in das Thema Programmierung zu vertiefen.

Alle Aufgaben sind jeweils über einen Zeitraum von etwa einem Monat zu bearbeiten. Das Semester wird in vier große Praktika unterteilt. Diese sind jeweils im Laufe des Semesters (weit vor dem Termin des Kolloquiums abzugeben).

Ziel der Aufgabe: Reaktive Systeme

Wenn Sie diese Aufgabe erfolgreich abgeschlossen haben, können Sie

- Ein Reaktives System selbst programmieren, durch Bau eines einfachen reaktiven Systems und sie haben verstanden, was ein reaktives System ist.
- Einen endlichen Automaten / eine Mealy-Maschine umsetzen.
- JSON-Strings erstellen und parsen
- Konzepte der synchronen und asynchronen Verarbeitung speziell Producer / Consumer-Pattern und das Executor-Framework.
- Sie haben einfache Netzwerkkommunikation auf der Grundlage von TCP/IP-Sockets umgesetzt.

Der Aufgabentext wird Ihnen völlig unlösbar erscheinen (das ist Absicht). Die Kunst besteht jetzt darin, dieses **viel zu komplizierte Problem** in für Sie machbare Teilschritte zu zerlegen und diese dann, soweit Sie kommen, abzuarbeiten. Also von einem kleinen Teilprogramm zum nächsten und diese dann wieder zu einer Gesamtlösung zu integrieren.

Aufgabe: Wir bauen ein Anwesenheitskontroll-System

Erzeugen Sie Ihr Projekt mit Gradle als Build-Werkzeug!

Automat: Implementieren Sie eine Klasse Mitarbeiter, diese enthält einen endlichen Automaten. Diese Maschine hat wie in Abbildung 1 dargestellt:

- Vier Zustände: Fehler, Anwesend, ImGang und Abwesend
- Ein Eingabealphabet mit zwei Symbole: Links und Rechts
- Zustandsübergangsfunktion, diese bekommt als Parameter einen Zustand und ein Zeichen. Die Zustandsübergangsfunktion verwendet im Hintergrund bitte eine Zustandsübergangstabelle (als Indizes verwenden Sie Zustand (ordinale) und Zeichen (ordinale)); in den Zellen der Tabelle stehen dann Zustände.

- Man kann Objekte der Klasse Mitarbeiter nach ihrem aktuellen Zustand fragen und man kann dem Objekt ein Symbol (Links, Rechts) übergeben und dann führt dieses intern den Zustandsübergang durch.
- Ein Mitarbeiter hat eine eindeutige ID als String (Fachlich könnte das die MAC-Adresse seines Smartphones sein).

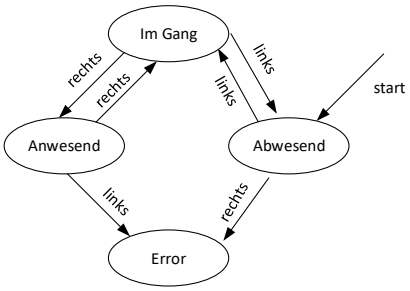

Abbildung 1: Automat für den Mitarbeiter

Entwickeln Sie die Klasse Mitarbeiter testgetrieben: Sichern Sie die Funktionsweise über eine Testsuite in JUnit ab.

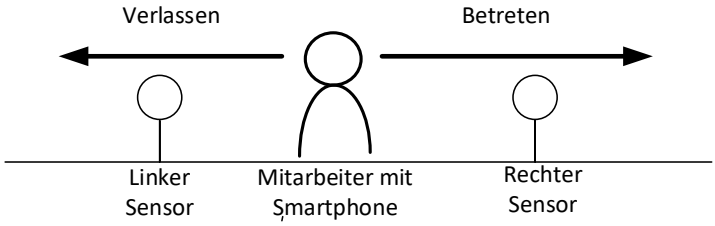

Abbildung 2: Eingangskontrolle

Zur Info: Wir bauen damit eine Eingangskontrolle (vgl. Abbildung 2), diese hat zwei Sensoren Links und Rechts. Geht ein Mitarbeiter von Links nach Rechts betritt er unsere Firma. Geht er von Rechts nach Links verlässt er die Firma. Wir stellen uns die Sensoren als Bluetooth-Empfänger oder als WLAN-Access Point vor, sodass wir die MAC oder Bluetooth-Adresse vom Smartphone des jeweiligen Mitarbeiters an den Sensoren empfangen.

Ereignisse:

Schrebein Sie nun (testgetrieben) eine Klasse Ereignis, diese die Information „Links“ oder „Rechts“ und eine ID als String. Objekte vom Typ Ereignis werden von den Sensoren erzeugt, wenn ein Mitarbeiter mit seinem Smartphone an ihnen vorbeigeht. Die ID ist beispielsweise die MAC-Adresse des WLAN Chips vom Smartphone.

Mitarbeiterverwaltung:

Schreiben sie nun (testgetrieben) eine Klasse Mitarbeiterverwaltung, ein Objekt (Singleton) dieser Klasse verwaltet Mitarbeiter-Objekte. Die Mitarbeiterverwaltung hat eine Methode notify(Event e). Über die ID in dem Event wird das Mitarbeiter-Objekt gefunden und diesem wird dann die Information „Links“ oder „Rechts“ übergeben. Existiert kein Mitarbeiter-Objekt mit der entsprechenden ID wird es neu erzeugt.

Logging und Errorhandling:

Bauen Sie nun Logging in die Mitarbeiterverwaltung ein. Sie erstellen zwei Log-Dateien. Die erste enthält allgemeine Betriebsinformationen ihrer Mitarbeiter verwaltung. Die zweite Datei protokolliert die oben genannten Events. Ziel ist des, diese datei später mit einem anderen Werkzeug zu analysieren.

Überlegen Sie sich, wie die Mitarbeiterverwaltung sinnvoll mit Fehlern umgehen kann. Die Software läuft ja später vermutlich z.B. auf einem RaspberryPi am Eingang, ohne eine grafische Oberfläche. Zu Fehlern gehören der Error-Zustand eines Mitarbeiters, oder diverse Exceptions beim Betrieb der Mitarbeiter-Verwaltung und Expeptions die aus der nachfolgenden Netzwerkkommunikation kommen.

Server mit Netzwerkanschluss:

Bauen Sie ein Hauptprogramm, das eine Endlosschleife enthält, dies wird unser Server. In der Endlosschleife (eigener Thread) wird jeweils ein Objekt der Klasse Ereignis aus einer BlockingQueue gelesen und dem Mitarbeiter-Verwaltungsobjekt übergeben.

Schreiben Sie nun eine Methode, die in einem eigenen Thread ausgeführt wird und den ServerSocket (TCP/IP) bereitstellt. Bei jedem Verbindungsversuch zu diesem Socket sollte ein weiterer Thread mit dem dann entstandenen Socket erzeugt werden (bitte verwenden Sie dazu das Executor Framework). Dieser Thread liest aus dem Socket die ankommenden Daten aus. Er wandelt diese Daten in ein Objekt der Klasse Ereignis um und schreibt die Ereignis-Objekte in die oben genannte BlockingQueue. Das Hauptprogramm und der „Empfangs-Thread“ kommunizieren also über diese Blocking Queue. Die hier empfangenen Daten werden von den beiden Sensoren über eine TCP/IP Verbindung gesendet. Die Daten werden als JSON-String über den Socket gesendet, z.B.
{ "type": "LINKS", "id": "00-14-22-01-23-45" }

Achten Sie bitte im jedem Fall darauf, dass alle allokierten Ressourcen auch im Fehlerfall wieder freigegeben werden.

Achten Sie weiterhin darauf, dass sie ihren Server sauber herunterfahren können müssen, möglichst ohne Datenverlust.

Sensoren als Clients:

Schreiben Sie eine Klasse Sensor (mit Hauptprogramm), diese kann als linker oder rechter Sensor instanziiert werden. Ihre Objekte senden als „Client“ an den gerade beschriebenen Server über Sockets Daten (die Ereignisse als JSON-String von oben). Die Daten werden ebenfalls in einer Log-Datei protokolliert. Schreiben Sie den Sensor zunächst so, dass dieser in regelmäßigen Abständen zufällige Testdaten sendet. Testen Sie mit zwei Sensoren und halbwegs sinnvoll erzeugten oder von der Konsole eingelesenen Testdaten den Server.

Konfiguration von Außen:

Sorgen Sie nun dafür, dass die zwei Programme „Server“ und „Client“ (zweimal Starten) von außen über Kommandozeilenparaemter, Java-Properties oder auf andere Weise konfigurierbar sind. Konfigurierbar sind beispielsweise: Sensor-Art (links, rechts), IP-Adressen und Portnummern. Programmieren Sie dazu eventuell ein Interface IConfiguration, verschiedene Implementierungen dazu, lesen die Konfigurationsdaten aus unterschiedlichen Quellen aus.

Härten:

Testen Sie ihre Software nun auf Robustheit. Was passiert, wenn der Server Blödsinn geschickt kriegt, oder ein Client abstürzt? Was passiert wenn ein Client hängen bleibt und die Verbindung offen lässt. Versuchen Sie während der Tests ihre Software so robust wie möglich gegen Fehler zu machen.

Bitte Beachten:

- Saubere Package-Struktur wählen
- Variablen, Klassen und Methoden so benennen, dass Fremde SCHNELL ihren Code verstehen
- Kommentare so wählen, dass Fremde SCHNELL verstehen, was sie sich gedacht haben
- Code so formatieren, dass Fremde SCHNELL verstehen, wie der Code funktioniert, also keine zu tiefen if-Kaskaden. keine zu langen Methoden, keine zu großen Klassen. Wenn der Code zu kompliziert wird: gnadenloses Refactoring!
- Möglichst große Teile des Quelltextes Testgetrieben entwicklen
- Immer wieder mit TestCoverage überpfüfen, ob ihnen ein Testfall fehlt
- Die Java-Coding-Conventions einhalten (Klassennamen groß, Methoden und Variablen klein)
- Keine Magic-Numbers im Code sondern eher konfigurierbare Parameter oder Konstante
- Achten Sie beim Logging darauf, dass die entstehende Datei auch ausgewertet werden muss.