Vorlesung: Dr.-Ing. Matthias Becker - xmb@sim.uni-hannover.de
Übung: Philipp Schiffmann - schiffmann@sim.uni-hannover.de
Florian Chudigiewitsch - fch@thi.uni-hannover.de

Assignment 2

Objektorientierung, Dokumentation

Abgabe: **24.04.2018, 23:59 Uhr**

> *The best programs are written so that computing machines can perform them quickly and so that human beings can understand them clearly. A programmer is ideally an essayist who works with traditional aesthetic and literary forms as well as mathematical concepts, to communicate the way that an algorithm works and to convince a reader that the results will be correct.*
>
> — Donald Ervin Knuth, *Selected Papers on Computer Science*

I. Motivation

War das Lernziel in Programmieren I noch, Studierenden die Grundlagen einer Programmiersprache und von algorithmischem Denken zu vermitteln, sollt ihr in Programmieren II auch lernen, die Modellierung des Problems und die Strukturierung selbst zu übernehmen. Es besteht ein Unterschied dazwischen, einen Lückentext auszufüllen und eine komplette Programmarchitektur von Grund auf zu erstellen. Letzteres soll in den Einzelübungen von Programmieren II trainiert werden. Deine Aufgabe wird demzufolge darin bestehen, aus einer natürlichsprachlichen Beschreibung der Aufgabe ein Projekt aufzubauen, das langfristig les- und wartbar bleibt. Dadurch hast du mehr Freiheiten beim Programmieren, es gibt kein absolutes „richtig“ oder „falsch“ mehr. Stattdessen wird sich die Bewertung mehr auf „guter Stil“ oder „schlechter Stil“ fokussieren.

II. Verkehrssimulation

Im Laufe der nächsten vier Assignments sollst du eine einfache Verkehrssimuation erstellen. Wenn sie fertig ist, soll diese Simulation die Frage beantworten, wie hoch die durchschnittliche Zeitersparnis für einen Autofahrer ist, wenn wir in unserem (fiktiven) Straßennetz alle zeitgesteuerten Ampeln durch bedarfsgesteuerte ersetzen. In diesem Assignment wird erst einmal eine sehr einfache Modellierung vorgenommen, die du in den folgenden Assignments sukzessive zu einer komplexeren Simulation erweiterst.
Diese Woche beginnen wir mit einem sehr kleinen Straßennetz, bestehend aus lediglich fünf Knotenpunkten und vier Straßen. Das Straßennetz modellieren wir mit zwei Klassen: `Kreuzung` und `Fahrbahn`. Letztere stellt tatsächlich nur eine einzelne Fahrbahn dar. Auf einer einzigen Fahrbahn können nicht mehrere Autos nebeneinander fahren, und sie kann nur in eine Richtung benutzt werden. Aktuell können Autos also zum Beispiel nur von A nach C fahren, aber nicht von C nach A.

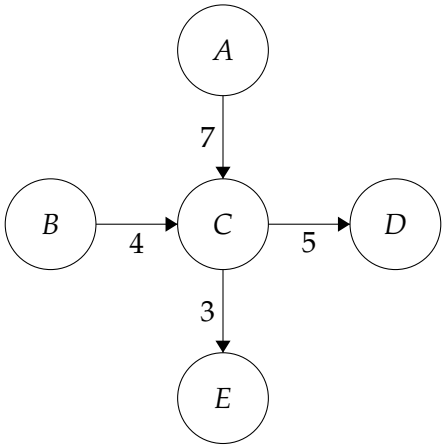

Abbildung 1: Die Zahlen an den Kanten geben die Länge der jeweiligen Straße an, und die Pfeile die Fahrtrichtungen.

Als nächstes definieren wir, wie sich Autos in unserem System verhalten sollen. Grundsätzlich sollen regelmäßig neue Autos zur Simulation hinzugefügt werden, die dann eine zufällig gewählte Route abfahren und anschließend wieder aus dem System entfernt werden. Diese Route soll aus einer Reihe von benachbarten Kreuzungen bestehen, die in dieser Reihenfolge abgefahren werden sollen. Wird also ein Auto an *A* erzeugt und soll nach *D* fahren, erhält es die Route *A*, *C*, *D*.

Als nächstes betrachten wir, wie wir das „Fahren“ umsetzen. Dafür setzen wir die Schrittgröße der Simulationszeit einfach auf die Zeit, die ein Auto benötigt, um sich genau eine Autolänge vorwärts zu bewegen. (Alle unsere Autos sind gleich lang) Dadurch lässt sich die Simulation durch folgenden Pseudocode ausführen:

```
until simulation terminates:
    increase total elapsed time counter by 1.
    for each car in the system:
        update car state.
```

Alle Straßenlängen geben wir ebenfalls in Autolängen an. Jetzt können wir uns eine Fahrbahn als eine Reihe von Feldern vorstellen, und *update car state* bedeutet schlicht, dass sich das Auto ein Feld nach vorne bewegt – natürlich nur, solange das nächste Feld frei ist.
Zuletzt vereinfachen wir uns unser Modell noch ein bisschen weiter, indem wir die Implementierung der Ampelanlagen erst nächste Woche betrachten. Wenn ein Auto also an eine Kreuzung gelangt, darf es direkt auf die nächste Fahrbahn weiterfahren, sobald diese frei ist, und muss sich nicht um weitere Verkehrsregeln wie Rechts-vor-Links kümmern. Außerdem bestimmen wir, dass Kreuzungen keine eigenen Felder besitzen. Zwei Autos können also dieselbe Kreuzung im selben Zeitschritt überqueren, solange sie nicht auf die selbe Zielstraße auffahren wollen.

III. Lösungshinweise

Modelliere `Kreuzung`, `Fahrbahn` und `Auto` jeweils in einer eigenen Klasse. Um verschiedene Objekte derselben Klasse beim Debuggen leichter auseinanderhalten zu können, kann es sinnvoll sein, jeder

Klasse ein Attribut `name` zu geben und diese Namen bei der Objekterzeugung von Hand auf eindeutige Werte zu setzen.
Eine Kreuzung hat genau vier ankommende und vier ausgehende Fahrbahnen entsprechend den vier Himmelsrichtungen. Ist die Kreuzung in dieser Himmelsrichtung nicht mit einer anderen Kreuzung verbunden, enthält die entsprechende Variable den Wert `null`.
Eine Fahrbahn hält Referenzen auf die Kreuzungen, an denen sie beginnt und endet, und verwaltet intern die Information, welche ihrer Felder aktuell frei oder durch ein Auto belegt sind. So können andere Autos effizient abfragen, ob sie noch ein Feld vorrücken dürfen.
Ein Auto enthält die Straße und das Feld, auf dem es sich gerade befindet; einen Zähler, der die Lebenszeit des Objektes zählt und in jedem Simulationsschritt erhöht wird; sowie die abzufahrende Route. Diese Route kann als verkettete Liste von Kreuzungen implementiert werden, wobei jeder `RoutenAbschnitt` eine Kreuzung und den nächsten `RoutenAbschnitt` enthält. Aus der Gesamtlänge der Route und der Lebenszeit des Autos können wir dann in der nächsten Woche die an Ampeln gewartete Zeit berechnen.
Schreibe eine Klasse `Simulation`, die die Simulation ausführt. Im Konstruktor kannst du das Straßennetz initalisieren. Außerdem soll die Klasse eine Methode `run` enthalten, die die Simulation entsprechend dem oben aufgeführten Pseudocode durchführt, aber zusätzlich in jedem Zeitschritt den aktuellen Zustand (Position, Lebenszeit) aller aktiven Autos ausgibt.
Verwalte die Liste der aktiven Autos innerhalb der `Simulation`-Klasse. Du kannst ein `array` dafür benutzen und davon ausgehen, dass nie mehr als 10 Autos gleichzeitig im System sein werden.

IV. Aufgaben

1. Gliedere deinen Code in packages, die der offiziellen Namenskonvention folgen:
   `de.uni_hannover.sim.<dein Name>.simulation` für die Klasse `Simulation`,
   `de.uni_hannover.sim.<dein Name>.model` für Kreuzung, Fahrbahn und Auto.

2. Erstelle alle Klassen mitsamt ihren Funktionen und Membervariablen.

3. Die Simulation soll nach 15 Zeitschritten terminieren. Erzeuge zum Zeitpunkt 0 ein Auto mit der Route *A, C, E*, zum Zeitpunkt 1 ein Auto mit der Route *A, C, D* und zum Zeitpunkt 3 ein Auto mit der Route *B, C, E*. Autos, die ihr Ziel erreicht haben, sollen von der Fahrbahn entfernt und und nicht weiter aktualisiert werden.

4. Schreibe eine `main`-Methode, die eine Simulation instanziert und darauf `run` aufruft.

V. Bewertungskriterien

Um dieses Aufgabenblatt zu bestehen,

1. muss dein Code ohne Fehler compilieren.

2. musst du das beschriebene Datenmodell mit Hilfe von Klassen modelliert haben.

3. muss du dir Gedanken darüber gemacht hast, welche Methoden und Attribute du in welche Klasse schreibst.

4. sollte die Simulation grundsätzlich laufen.