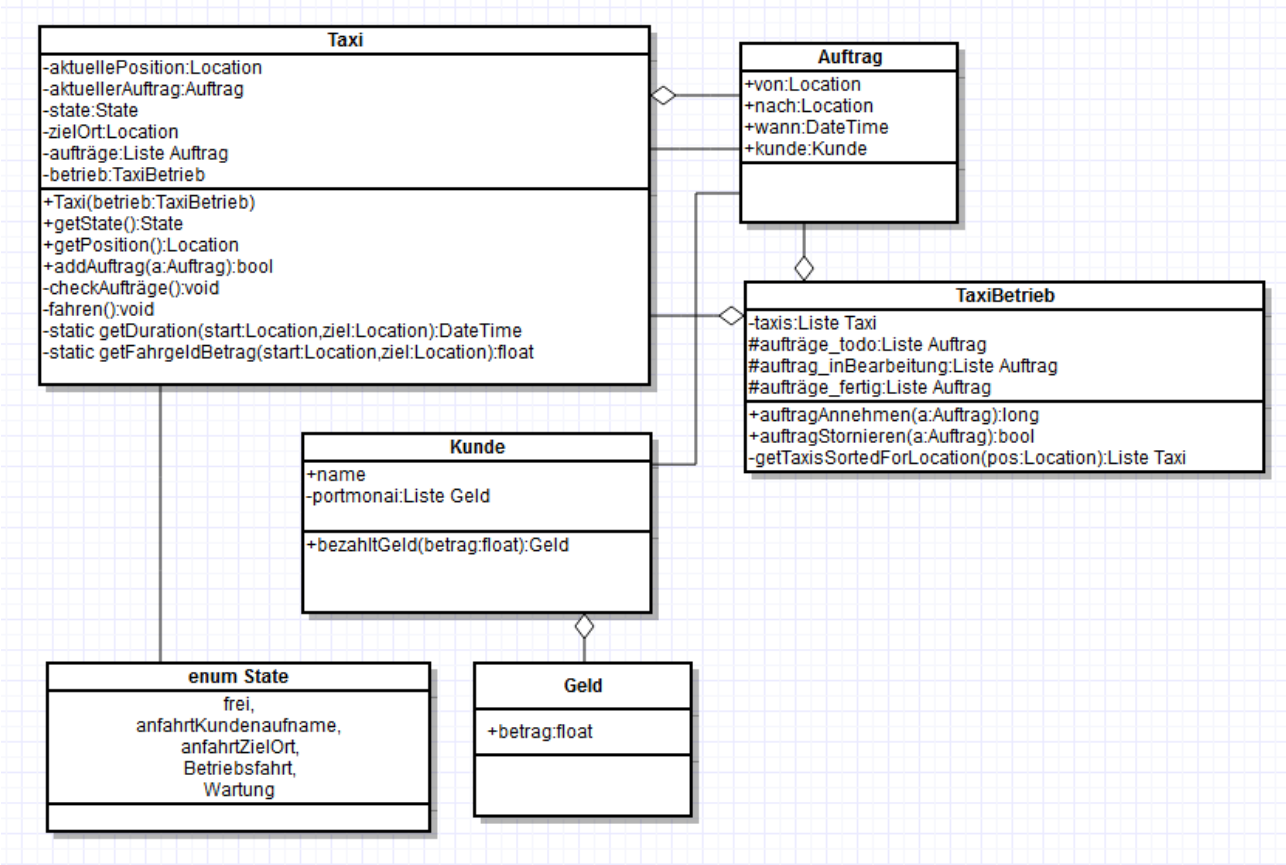
Ein Uber-Taxi-Unternehmen besitzt 3 autonome Taxis, welche gleichzeitig im Einsatz sind. Taxis haben eine aktuelle Position, und besitzen den Status: frei (F), anfahrtKundenaufname (KA), anfahrtZielOrt (Z), Betriebsfahrt zu einem Sammelpunkt (B), in Wartung (W). Die Verwaltung der Taxis wird mit Hilfe eines Softwaresystems abgebildet.

Eine Struktur als Klassendiagramm ist gegeben:



Methoden der Klasse TaxiBetrieb

auftragAnnehmen ein Kunde meldet sich beim Taxibetrieb und teilt einen Auftrag mit. Dieser Auftrag wird einem Taxi übertragen, welches den Auftrag-Start-Ort (von) zum Zeitpunkt des Auftrags (wann) am nahesten sein wird / ist, und verfügbar sein wird / ist. Ist kein Taxi verfügbar, wird der Auftrag nicht angenommen (es wird -1 returned, ansonsten die Wartezeit, bis das Taxi da ist)

auftragStornieren ein Auftrag kann vom Kunden storniert werden, wenn das Taxi, welches den Auftrag bekommen hat, den Kunden noch nicht aufgenommen hat (im Falle der Stornierung wird ein true returned).

getTaxisSortedForLocation gibt eine Liste aller Taxis zurück geordnet nach der Entfernung zur angegebenen Position (das naheste Taxi zuerst)

Methoden der Klasse Taxi

getState gibt den aktuellen Zustand des Taxis zurück

getPosition gibt den aktuellen Ort des Taxis zurück

addAuftrag fügt einen Auftrag in die Liste der Aufträge ein. Diese Liste wird nach Einfügen immer sortiert nach dem wann-Attribut der Aufträge (der zeitlich-naheste Auftrag steht immer zuerst in der

Liste, das sortieren kann mit ::sort() erreicht werden). Ein Auftrag kann abgelehnt werden, wenn das Taxi zum wann-zeitpunkt des Auftrags bereits einen Auftrag durchführen werden wird (laut seiner Auftragsliste).

checkAufträge es wird regelmäßig (alle 60s) überprüft, ob der nächste Auftrag begonnen werden muss.

fahren wird ausgeführt im Rahmen eines Auftrags, oder einer Betriebsfahrt. Nimmt auch das Fahrgeld ein und sorgt für die Aktualisierung der auftrags-listen der Klasse TaxiBetrieb.

Die Dauer einer Fahrt kann mit der Methode *getDuration* ermittelt werden, das Fahrgeld wird mit der Methode *getFahrgeldBetrag* berechnet

Methoden der Klasse Kunde

bezahltGeld der Kunde bezahlt das Fahrgeld

Exercise Zustandsdiagramm & Pseudocode

Erstellen Sie ein Action-State (Zustandsdiagram) mit welchem die Zustände eines Taxis und die (im Rahmen der Aufgabenstellung sinnvollen) Aktionen dargestellt werden, welche die Zustände verändern.

Erstellen Sie das Verhalten als Pseudocode für die methode *auftragAnnhemen* der Klasse TaxiBetrieb.

Exercise Sequenzdiagramm

Erstellen Sie das Sequenzdiagramm, mit welchem die Kommunikation zwischen den benötigten Objekten abgebildet wird, für folgendes Szenario:

Uhrzeit	k1:Kunde	k2:Kunde	t1:Taxi	t2:Taxi	t3:Taxi
10:00			frei, im HBF	frei, an der DHBW	frei, am Schloss
10:05	bucht eine Fahrt vom HBF zur DHBW für 10:30 (Fahrzeit 15 min)				
10:10					
10:15					
10:20					das Taxi hat einen Defekt und fällt für 20 min aus
10:25					
10:30					
10:35		möchte sofort vom HBF zur DHWB fahren			
10:40					
10:45		storniert die Fahrt zur DHBW, da er ein freies Taxi eines anderen Unternehmens findet			