Die Hausaufgaben müssen von jedem Studierenden einzeln bearbeitet und abgegeben werden. Für die Hausaufgabe sind die aktuellen Informationen vom Blog https://seblog.cs.uni-kassel.de/ws2324/programming-and-modelling/ zu berücksichtigen.

Abgabefrist ist der 30.11.2023 - 23:59 Uhr

Abgabe

Wir benutzen für die Abgabe der Hausaufgaben Git. Jedes Repository ist nur für den Studierenden selbst sowie für die Betreuer und Korrektoren sichtbar.

Für die Hausaufgabe benötigst du **ein neues** Repository.

Dieses kann über folgenden Link erstellt werden, falls nicht bereits geschehen:

https://classroom.github.com/a/MAtaZZbV

Nicht oder zu spät gepushte (Teil-)Abgaben werden mit 0 Punkten bewertet!

Vorbereitung

Zur Bearbeitung der Hausaufgabe sollte eine Entwicklungsumgebung verwendet werden. Wir empfehlen aufgrund der Nachvollziehbarkeit die Verwendung von IntelliJ (siehe Aufgabenblatt 3). Die Abgabe muss als **lauffähiges** Projekt abgegeben werden.

Abgaben, die nicht lauffähig sind, werden mit 0 Punkten bewertet!

Vorgegebenes Java-Projekt

Dein Repository enthält bereits ein Java-Projekt, das mit IntelliJ bearbeitet werden kann.

Aufgabe 1 - Wireframes (16P)

Ziel dieser Aufgabe ist es, das Prototyping in Form von Wireframes zu üben. Erstellt aus den Anforderungen zu den Oberflächen GameScreen und ShopScreen jeweils ein Wireframe. Es darf sich an den Wireframes der Vorlesung und Übung orientiert werden.

Zur Erstellung von Wireframes kann die Webanwendung diagrams.net verwendet werden.

GameScreen

Im GameScreen soll dem User in einem großen Bereich die Spielkarte angezeigt werden. Die Spielkarte enthält folgende Elemente:

- Städte mit ihren Namen
- der Headquarter hebt sich von den anderen Städten in irgendeiner Weise ab
- Straßen
- Autos, die sich auf einer Straße oder bei einer Stadt befinden
- bei Städten, an denen es Aufträge gibt, muss dieser durch eine Markierung an der Stadt angezeigt werden
- es muss erkennbar sein, dass eine Straße blockiert ist

In einem Bereich neben der Karte soll dem User die Übersicht über Aufträge, Autos und weitere Informationen angezeigt werden. Diese Übersicht enthält folgende Elemente:

- verfügbare Aufträge mit Ablaufdatum, Belohnung und Zielort
- Autos des Spielers mit Namen der Fahrer
- es muss erkennbar sein, welche Autos bereits für Aufträge eingesetzt wurden und welche Autos gerade keinem Auftrag zugeteilt sind
- es muss eine Möglichkeit geben, einem Auftrag ein freies Auto zuzuteilen
- aktuelles Guthaben des Spielers
- ein Button, über den man ein weiteres Auto kaufen kann und eine Info darüber, wie viel das Auto kostet

ShopScreen

Der ShopScreen ist eine kleine Ansicht, die nach dem Klick auf den Kaufen-Button angezeigt werden soll. Die Ansicht enthält folgende Elemente:

- eine Aufforderung, den Namen des Fahrers für das neue Auto einzutragen
- ein Textfeld zum Eintragen des Namens
- ein Button zum Bestätigen des Kaufs
- ein Button zum Abbrechen

Lege die erstellten **.pdf**-Dateien „GameScreen.pdf“ und „ShopScreen.pdf“ in einem Ordner mit dem Namen „task1“ in deinem Repository ab. Handgezeichnete Abgaben sind nicht erlaubt. Committe und pushe die Änderung abschließend auf den main-Branch.

Die Wireframes müssen jeweils als einseitige PDF-Datei (.pdf) abgegeben werden.

Bei der Bewertung wird vor allem darauf geachtet, dass auf den Wireframes alle benötigten Informationen und Eingabemöglichkeiten erkennbar sind.

Achte darauf, das Repository der aktuellen Hausaufgabe zu verwenden.

Aufgabe 2 - Referentielle Integrität testen (8P)

Die zu implementierende Funktionalität aus Aufgabe 3 soll getestet werden. Diese Test-Aufgabe kann unter Verwendung des Test-First-Prinzips gelöst werden, weshalb sie an zweiter Stelle des Aufgabenblattes steht. Die tatsächliche Lösungsreihenfolge der Aufgaben ist dir überlassen.

Erstelle unter src/test/java im Package de.uniks.ws2324.tiny.model eine Klasse `ModelTest`. Folgende Methoden des Klassenmodells sollen in eigenen Testmethoden getestet werden:

- Methode `setCar` aus der Klasse `Order`
- Methode `withStreets` aus der Klasse `City`
- Methode `withoutStreets` aus der Klasse `City`

setCarTest

Dieser Test soll sicherstellen, dass nach dem Aufruf von `setCar` eine **gegenseitige Verbindung** zwischen Car- und Order-Objekt besteht.

Wenn mit `setCar` eine neue Verbindung hergestellt wird, darf das alte Car-Objekt **keine Verbindung** mehr zum Order-Objekt haben.

Um einem Order-Objekt **kein** Car-Objekt zuzuordnen, wird `setCar` mit `null` aufgerufen. Hierbei muss ebenfalls eine mögliche Rückrichtung zum alten Car-Objekt **aufgelöst** werden.

withStreetsTest

Dieser Test soll sicherstellen, dass nach dem Aufruf von `withStreets` eine **gegenseitige Verbindung** zwischen Street- und City-Objekt besteht.

Über einen weiteren Aufruf von `withStreets` kann ein weiteres Street-Objekt **hinzugefügt** werden. Das alte Street-Objekt bleibt weiterhin vorhanden.

Wird versucht, das selbe Street-Objekt mehrmals hinzuzufügen, verändert dies nichts.

withoutStreetsTest

Dieser Test soll sicherstellen, dass nach dem Aufruf von `withoutStreetsTest` die **gegenseitige Verbindung** zwischen Street- und City-Objekt **aufgelöst** wurde.

Der Versuch, ein Street-Objekt zu entfernen, welches nicht in der Liste vorhanden ist, hat keine Auswirkung auf die Liste.

Weitere Infos zu den Assoziationsmethoden gibt es in Aufgabe 3.

Es dürfen zusätzliche Tests für weitere Assoziationen hinzugefügt werden.

Bei der Bewertung wird vor allem darauf geachtet, dass geeignete Test-Szenarien aufgebaut und die Assertions sinnvoll eingesetzt werden.

Achte darauf, das Repository der aktuellen Hausaufgabe zu verwenden.

Aufgabe 3 - Klassendiagramm zu Java (17P)

In dieser Aufgabe implementieren wir ein vereinfachtes Klassendiagramm des Spiels "TinyTransport". Hierzu ist das in Abbildung 1 dargestellte Klassendiagramm gegeben.

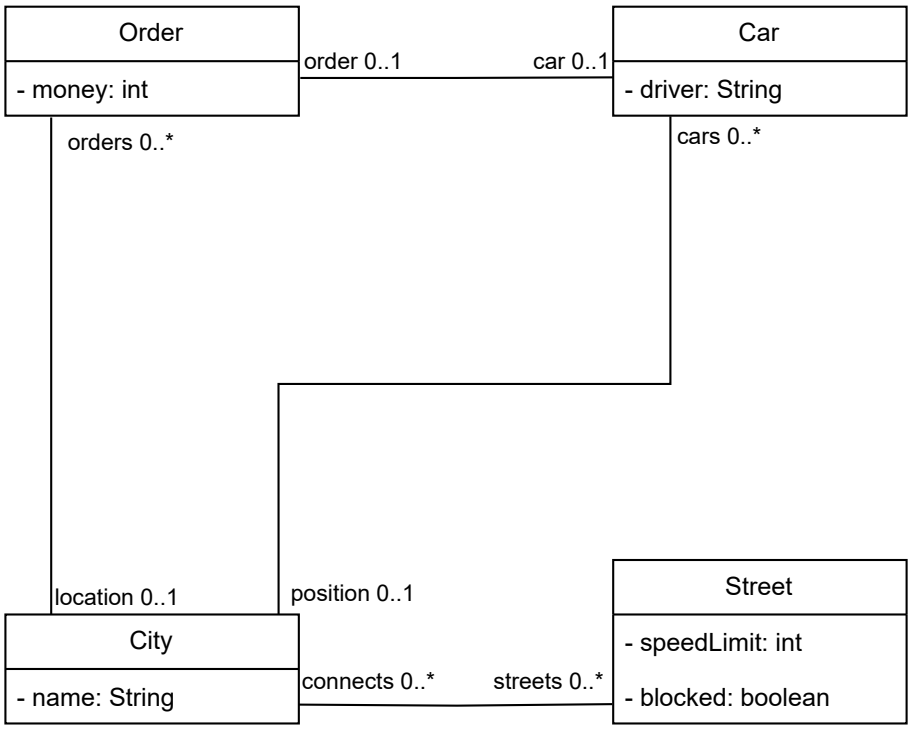

Abbildung 1: TinyTransport-Klassendiagramm (vereinfacht)

Folgende Vorgehensweise wird vorgeschlagen:

1. Erstelle für jede Klasse im Diagramm eine Klasse in einer separaten .java-Datei unter dem Modul src/main/java im zu erstellenden Package de.uniks.ws2324.tiny.model
2. Füge den Klassen keine Konstruktoren hinzu.
3. Füge den Klassen die entsprechenden Attribute hinzu. Achte dabei auf die Zugriffsverkapselung der Attribute mit den Zugriffsmethoden Set und Get aus der Vorlesung!
   - <Class> set<Field>(<Type> <Variable>) und <Type> get<Field>()
4. Implementiere die Assoziationen und stelle die referenzielle Integrität sicher. Dies verlangt folgende Punkte:
   - Füge den Klassen für 0...1-Assoziationen die Zugriffsmethoden <Type> get<Field>() und <Class> set<Field>(<Class> <Variable>) hinzu
   - Füge den Klassen für 0...n-Assoziationen die Zugriffsmethoden List< <Class> > get<Field>() sowie <Class> with<Field>(<Class> <Variable>) und <Class> without<Field>(<Class> <Variable>) hinzu
   - Wähle für die 0...n-Assoziationen eine geeignete Containerklasse (z.B. ArrayList)

Du kannst deine Implementierung mit dem Test aus Aufgabe 2 ausprobieren.

Bei der Bewertung wird vor allem auf die Projektstruktur, Zugriffsverkapselung sowie die korrekte Umsetzung der referenziellen Integrität geachtet.

Achtet darauf, das Repository der aktuellen Hausaufgabe zu verwenden.

Anhang

Es folgt eine Auflistung hilfreicher Webseiten und weiteren Erklärungen zu den Themen dieser Hausaufgabe. Die Links sind als Startpunkt zur selbstständigen Recherche angedacht. Das Durcharbeiten der folgenden Quellen ist kein bewerteter Anteil der Hausaufgaben.

Wireframes

- diagrams.net: https://diagrams.net
- Balsamiq (Desktop-Anwendung): https://balsamiq.com/