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 07.12.2023 - 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/kl_i4rdv

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

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

Projekte, deren GUI nicht mit FXML-Dateien umgesetzt sind, werden mit 0 Punkten bewertet!

Vorgegebenes Java-Projekt

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

Zukünftige Abgaben

Das oben genannte Repository ist der Startpunkt für die Anwendung, die im weiteren Verlauf dieser Veranstaltung entwickelt werden soll. Das Repository wird also fortan nicht mehr für jede Hausaufgabe gewechselt, sondern für kommende Abgaben weiterverwendet.

Konventionen zur Benennung

Halte dich beim Programmieren in dieser und allen folgenden Hausaufgaben an die nachfolgenden Naming Conventions:

https://www.oracle.com/java/technologies/javase/codeconventions-namingconventions.html

Beachte, dass Code und Kommentare in Englisch verfasst werden sollen. Verstöße führen zu Punktabzug.

Aufgabe 1 - Fulib (18P)

In dieser Aufgabe wird das vollständige Datenmodell des Spiels "TinyTransport" mit Fulib generiert. Fulib kann in bestehenden Projekten genutzt werden, um Datenmodelle zu definieren.

Verwende die bereits existierende Klasse GenModel, um das Klassendiagramm aus Abbildung 1 zu definieren sowie zu generieren.

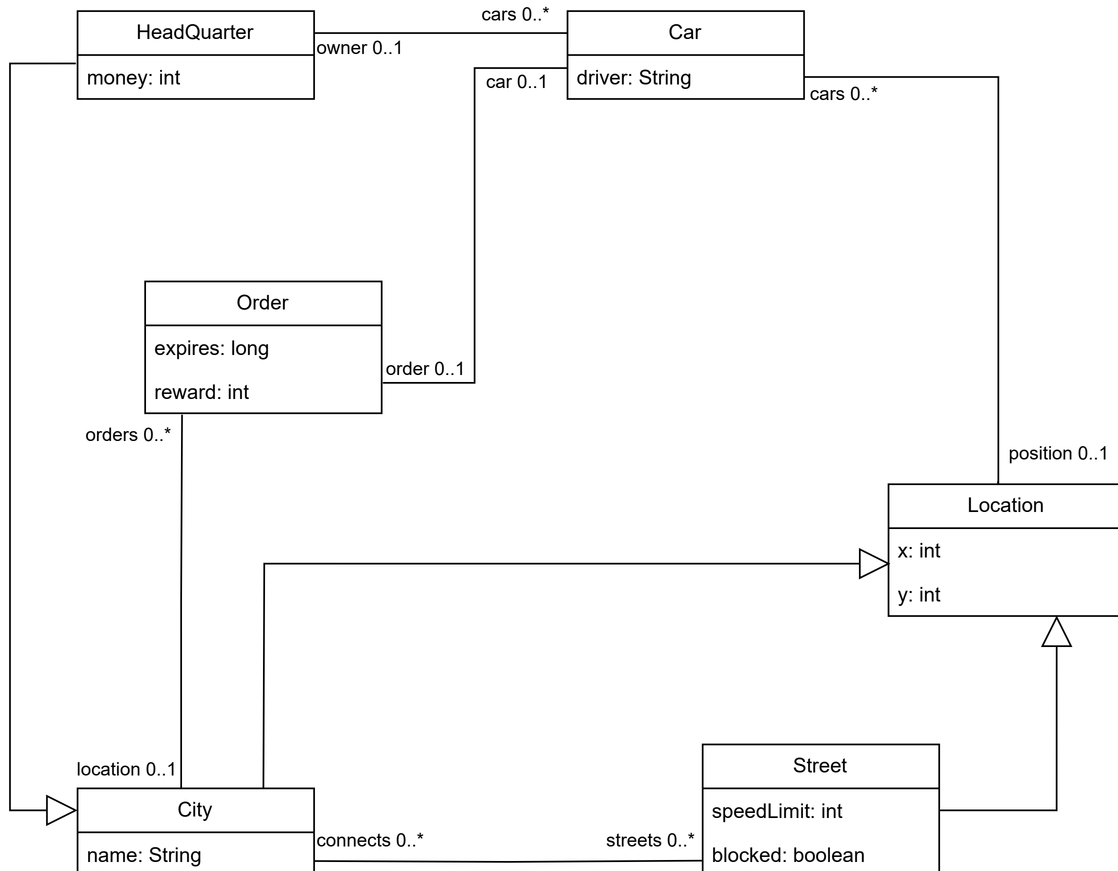

Abbildung 1: „TinyTransport"-Klassendiagramm

Committe und pushe die Änderungen abschließend auf den main-Branch.

Achte darauf, das Repository der aktuellen Hausaufgabe zu verwenden.

Aufgabe 2 - FXML (16P)

In dieser Aufgabe soll die Grundlage zur Umsetzung der in der vorherigen Hausaufgabe erstellten Wireframes geschaffen werden.

Vorbereitung

Wir verwenden für die Erstellung unserer Oberflächen den SceneBuilder. Lade dir den SceneBuilder unter folgendem Link herunter:

https://gluonhq.com/products/scene-builder/

Wenn du deine eigenen Wireframes nicht umsetzen möchtest, können die von uns zur Verfügung gestellten Wireframes (Abbildung 2 und 3) als Arbeitsgrundlage verwendet werden.

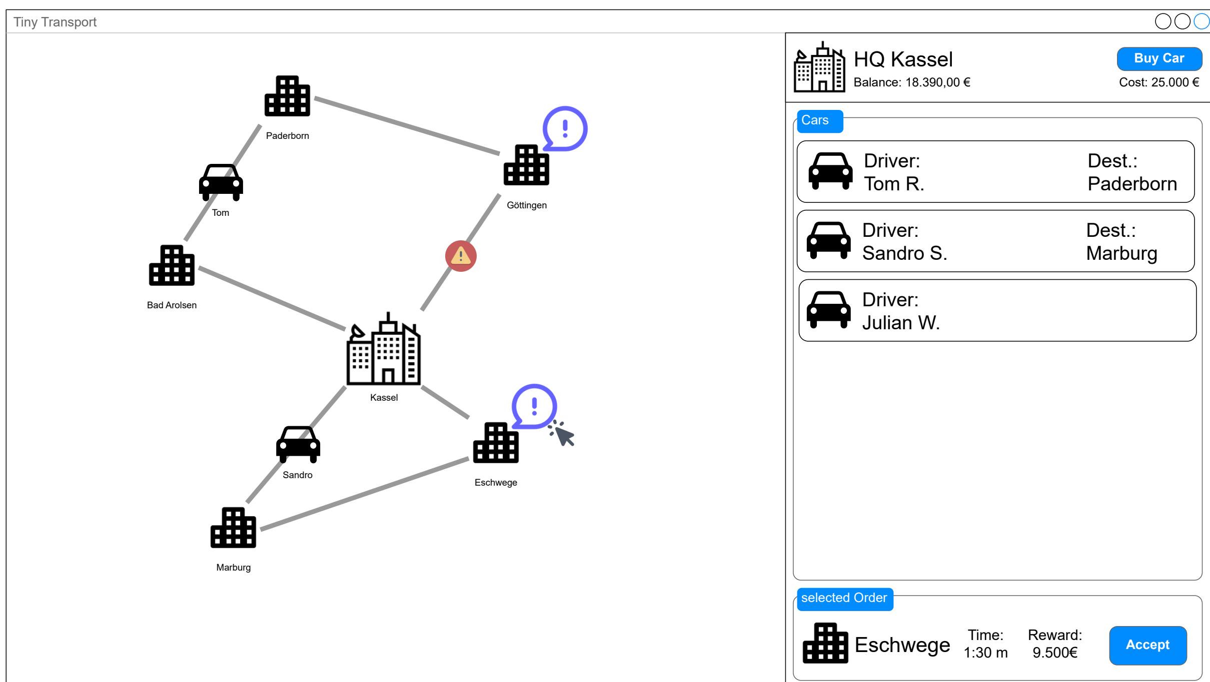

Abbildung 2: Game-Bildschirm

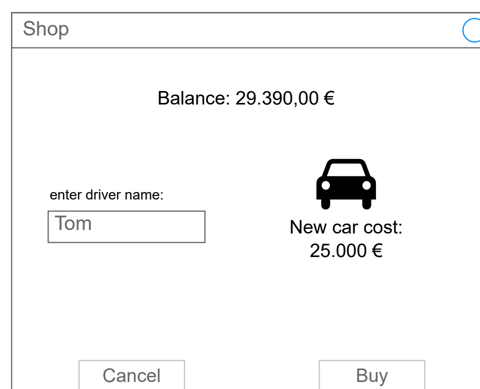

Abbildung 3: Shop-Bildschirm

1. Fxml-Dateien erstellen

Der Scenebuilder speichert seine Dateien im fxml-Format. Für jedes Wireframe muss eine eigene fxml-Datei erstellt werden (Game.fxml und Shop.fxml). Die zwei erstellten fxml-Dateien sind im Modul src/main/resources im Package de.uniks.pmws2324.tiny.view abzulegen. Achte darauf, dass im resources-Ordner die korrekt geschachtelte Ordnerstruktur des erforderten Packages erstellt wurde und nicht nur ein Ordner mit dem Namen „de.uniks.pmws2324.tiny.view“ existiert (dies ist leicht im File Explorer ersichtlich).

2. fx:ids vergeben

Sowohl beim Umsetzen deiner eigenen als auch unserer Wireframes sind **vorgegebene Benennungen** der fx:id bestimmter Komponenten zu berücksichtigen. Achte bei deinen eigenen Wireframes darauf, dass alle unten genannten Elemente vorhanden sind.

Game

Im Game-Bildschirm werden folgende fx:id vergeben:

- hqNameLabel für den Namen des HQs im Info-Bereich
- balanceLabel für den aktuellen Kontostand im Info-Bereich
- carCostLabel für die Anschaffungskosten des nächsten Autos
- shopButton für den Button zum Aufruf des Shops im Info-Bereich
- mapCanvas für den Canvas, auf dem wir später die Map mit Städten etc. einzeichnen
- orderTownLabel für den Stadtnamen des ausgewählten Auftrags
- orderTimeLabel für den Text, der die übrige Zeit des ausgewählten Auftrags enthält
- orderRewardLabel für den Text, der die Belohnung enthält
- orderAcceptButton für den Button, über den ein Auftrag angenommen werden kann

Die Blöcke, in denen Autos dargestellt werden, dürfen zunächst kopiert werden. Später werden wir diese in eine weitere Fxml-Datei auslagern. Beim Kopieren ist jedoch wichtig, dass die **fx:ids jeweils nur einmal vergeben werden**, andernfalls ist die Fxml-Datei fehlerhaft und kann nicht geladen werden. Kopiere daher **erst** deine Blöcke und vergib **danach** für den ersten davon die nachfolgenden fx:ids.

- carDriverLabel für den Text, der den Namen des Fahrers enthält
- carDestinationLabel für den Text, der die Zielstadt enthält oder leer bleibt, wenn das Auto keinen Auftrag hat

Shop

Im Shop-Bildschirm werden folgende fx:id vergeben:

- shopBalanceLabel für den aktuellen Kontostand
- shopCarCostLabel für die Anschaffungskosten des nächsten Autos

- nameInput für die Text-Eingabe für den Namen des Fahrers
- buyButton für den Button, über den der Kauf abgeschlossen werden kann
- cancelButton für den Button, über den der Kauf abgebrochen werden kann

Committe und pushe die neuen Dateien abschließend auf den main-Branch.

Achte darauf, das Repository der aktuellen Hausaufgabe zu verwenden.

Aufgabe 3 - Modell initialisieren (13P)

In dieser Aufgabe soll der Startzustand von TinyTransport in der Klasse GameService implementiert werden.

initGame

Zunächst benötigen wir fest definierte Städte und Straßen. Konstanten, die bei der Initialisierung genutzt werden sollen, sind bereits in Constants definiert. Lege die Städte und Straßen aus Abbildung 2 unter Verwendung der Konstanten an und verknüpfe sie mit `connects`-Links gemäß der Abbildung in einer Methode namens `initGame` im GameService. Es sollen noch keine Straßen blockiert sein.

Beachte, dass der HeadQuarter zwar eine Stadt ist, aber als HeadQuarter-Objekt angelegt werden soll.

Die Methode `connectCities` aus den vergangenen Hausaufgaben darf übernommen und genutzt werden.

Lege in `initGame` außerdem ein Car-Objekt an, welches beim HeadQuarter positioniert ist, als Owner ebenfalls den HeadQuarter hat und dessen Fahrerin „Alice“ heißt.

Erstelle in `initGame` außerdem zwei neue Orders mithilfe der Methode `generateOrder`.

generateOrder

Die Methode soll an einer zufällig gewählten Stadt aus der `cities`-Liste ein neues Order-Objekt verknüpfen, welches als `reward` und `expires` jeweils einen zufälligen Wert hat. Die Zufallswerte sollen mithilfe des `rnGenerators` und der vorgegebenen Grenzwerte aus der Constants-Klasse erzeugt werden.

GameServiceTest

Schreibe nun eine Test-Methode `initGameTest` in GameServiceTest (zu finden unter src/test/java/de/uniks/pmws2324/tiny/service). Der Test soll die zugehörige Methode aufrufen und das Ergebnis mit Asserts prüfen. Es genügt, die Verbindungen der Straßen und Städte, die explizit gesetzten Werte des Autos und das Vorhandensein der zwei Orders mit Attributen des gewünschten Wertebereichs zu prüfen.

Weitere Hilfsmethoden sind erlaubt.

Committe und pushe die Änderungen abschließend auf den main-Branch.

Bei der Bewertung wird vor allem auf die vollständige Umsetzung der Spielsituation geachtet.

Achte darauf, das Repository der aktuellen Hausaufgabe zu verwenden.

Anhang

Es folgt eine Auflistung hilfreicher Webseiten und weiterer 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.

fulib

- fulib.org: `https://fulib.org/`
- fulib-Dokumentation: `https://fulib.org/docs/fulib/README.md`
- fulib-GenModel-Beispiel:
  `https://fulib.org/docs/fulib/quickstart/1-defining-class-model.md`

Scene Builder

`https://gluonhq.com/products/scene-builder/`