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 14.12.2023 - 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 **kein neues** Repository. Es wird das gleiche Repository benutzt, das bereits in Hausaufgabe 6 angelegt wurde. 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!

Alle Tests (alt/neu) müssen nach wie vor funktionieren (ggf. angepasst werden), sollte dies nicht der Fall sein, wird mit 0 Punkten bewertet!

Aufgabe 1 - Klassenmodell anpassen (2P)

In der letzten Hausaufgabe wurde das Klassenmodell mit Fulib generiert. Ein Punkt, der während des Programmierens zu Verwirrung und Problemen führen könnte, ist die Benennung der gekauften Autos `cars` des HeadQuarters. Ein HeadQuarter besitzt aufgrund der Vererbung von Location bereits eine Liste namens `cars`, welche die Autos enthält, die sich aktuell „physisch“ am HeadQuarter aufhalten. Die Benennung kommt somit für zwei unterschiedliche Variablen / Anwendungszwecke vor. Um Fehlern vorzubeugen, soll die Liste `cars` im HeadQuarter zu `ownedCars` umbenannt werden. Das aktualisierte Klassendiagramm ist in Abbildung 1 zu sehen, die entsprechende Stelle wurde rot markiert.

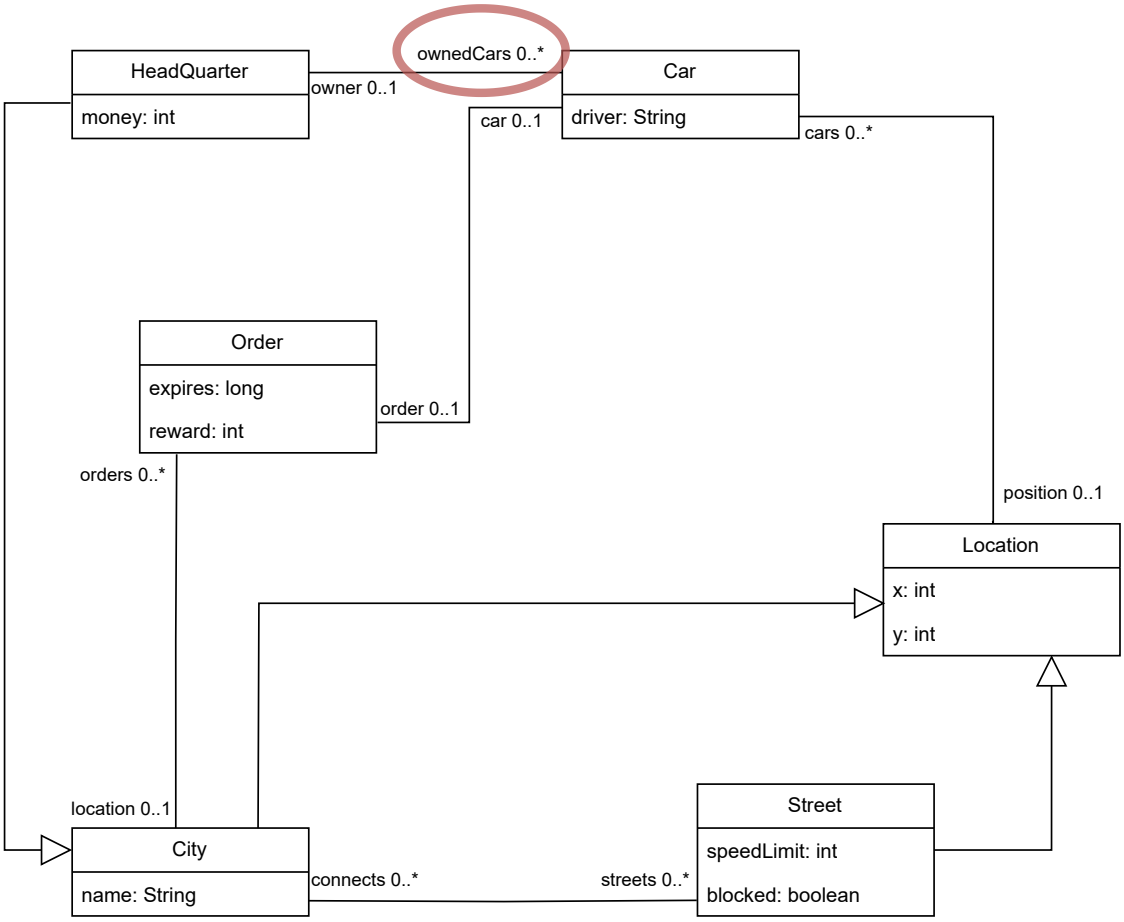

Abbildung 1: „TinyTransport”-Klassendiagramm

Nachdem das GenModel angepasst wurde, muss das Klassenmodell erneut generiert werden.

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

Achte darauf, das Repository der aktuellen Hausaufgabe zu verwenden.

Aufgabe 2 - JavaFX (27P)

In dieser Hausaufgabe wird das Oberflächenframework JavaFX verwendet.

Vorgegebene Klassen

Um einen einheitlichen Startpunkt zur Implementierung der JavaFx-Applikation zu schaffen, wurden folgende Java-Klassen bereitgestellt. Füge diese in das entsprechende Package unter src/main/java in dein Projekt ein.

- `de.uniks.pmws2324.tiny.Main` [1]
- `de.uniks.pmws2324.tiny.App` [2]
- `de.uniks.pmws2324.tiny.controller.Controller` [3]
- `de.uniks.pmws2324.tiny.controller.GameController` [4]

Verknüpfung von FXML und Controller

Verknüpfe den GameController mit dem UI der Game.fxml wie in der Übung gezeigt, sodass die mit `@FXML` annotierten UI-Elemente genutzt werden können.

Folge nun den Anweisungen der mit `// TODO` markierten Kommentare in der `render`-Methode, um die gewünschten Informationen als Text anzuzeigen und mithilfe von PropertyChangeListenern zu updaten etc.

Karte zeichnen

Implementiere die `drawMap`-Methode, um Städte, Straßen, Autos und Orders anzuzeigen. Halte dich an folgende Vorgaben:

- Städte sollen als gelbe Quadrate angezeigt werden. An geeigneter Stelle soll zusätzlich der Name der Stadt angezeigt werden. Der HeadQuarter soll besonders hervorgehoben werden.
- Die Straßen sollen als Linien eingezeichnet werden.
- Autos sollen als rote Kreise an ihrer aktuellen Location eingezeichnet werden.
- Orders sollen als kleine blaue Kreise an geeigneter Stelle an ihrer jeweiligen Stadt eingezeichnet werden.

Städte anklicken und Order anzeigen

Implementiere die `handleMouseClick`-Methode im GameController, um zu überprüfen, ob eine Stadt mit einem verfügbaren Auftrag angeklickt wurde. Im positiven Fall sollen die entsprechenden UI-Elemente, welche den ausgewählten Auftrag anzeigen, mit den Daten des

[1] https://github.com/sekassel/pmws2324-files/blob/main/HA07/Main.java
[2] https://github.com/sekassel/pmws2324-files/blob/main/HA07/App.java
[3] https://github.com/sekassel/pmws2324-files/blob/main/HA07/Controller.java
[4] https://github.com/sekassel/pmws2324-files/blob/main/HA07/GameController.java

Auftrags in sinnvollem Format befüllt werden. Wurde keine Stadt mit gültigem Auftrag angeklickt, soll entsprechend nichts angezeigt werden. Die vorgegebene Variable `selectedOrder` soll genutzt werden, um sich den ausgewählten Order zu merken. (Diese wird später bei der Implementierung der `handleAcceptOrder`-Methode gebraucht.)

Der Button zum Annehmen eines Auftrags soll die `handleAcceptOrder`-Methode ausführen (dies muss in der `render`-Methode festgelegt werden). Die Implementierung der `handleAcceptOrder`-Methode des GameControllers folgt in Aufgabe 3. Hat der User zuvor einen gültigen Auftrag ausgewählt, können mithilfe des GameServices die entsprechenden Methoden zum Ausführen eines Auftrags ausgeführt werden, diese sind ebenfalls Inhalt der Aufgabe 3.

Zusätzliche Hilfsmethoden sind erlaubt. Es dürfen weitere Konstanten angelegt werden. Bei der Ausführung und Benutzung des Programms sollten keine Exceptions auftreten.

Nach Bearbeitung der Aufgabe sollen alle `TODO`s aus den Kommentaren entfernt sein.

Bei der Bewertung der Aufgabe wird auf die Einhaltung des MVC-Patterns geachtet.

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

Achte darauf, das Repository der aktuellen Hausaufgabe zu verwenden.

Aufgabe 3 - Order annehmen (11P)

In dieser Aufgabe soll das Annehmen und grundlegende Abarbeiten von Orders umgesetzt werden. Hierzu soll die Methode `handleAcceptOrder` des GameControllers und die Methoden `acceptOrder`, `finishOrder` und `setNewCarPosition` im GameService implementiert werden.

Hinweis: Beachte, dass die Methode `getPath` aus vorhergegangenen Hausaufgaben in den GameService übernommen und evtl. angepasst werden muss, um eine Pfadfindung zu ermöglichen.

GameController

Implementiere die Methode `handleAcceptOrder` und vervollständige die `nextStep`-Methode.

In `handleAcceptOrder` muss die `acceptOrder`-Methode des GameServices mit dem selektierten Auftrag aufgerufen werden, um die Datenmodell-Logik zur Annahme einer Order durchzuführen. Nachdem dies geklappt hat, kann mithilfe des GameServices der Pfad geholt werden, den das Auto fahren wird. An geeigneter Stelle muss dann noch die `nextStep`-Methode verwendet werden, damit sich das Auto tatsächlich Schritt für Schritt auf dem übergebenen Pfad entlang bewegt. Außerdem müssen in `nextStep` auch wieder die mit `// TODO` markierten Kommentare entsprechend implementiert werden.

GameService

Erstelle und implementiere im GameService die Methoden

- `acceptOrder(Order selectedOrder)`
- `finishOrder(Order order)`
- `setNewCarPosition(Car car, Location location)`

Die Methode `acceptOrder` des GameServices soll dem Auto den ausgewählten Auftrag zuweisen. Dies soll nur möglich sein, wenn das Auto bisher keinen Auftrag hat und wenn es einen Weg zur Zielstadt gibt.

Die Methode `finishOrder` des GameServices soll dem HeadQuarter den Reward der Order gutschreiben. Außerdem wird das Auto wieder zum HeadQuarter zurück teleportiert. Die abgearbeitete Order muss von der Stadt und vom Auto entfernt werden.

Die Methode `setNewCarPosition` des GameServices setzt das Auto auf die neue Position.

Weitere Hinweise

Nach Implementierung der Aufgaben soll jederzeit korrekt angezeigt werden, an welchen Städten Orders vorhanden sind, auch nach Annahme und automatischem Abarbeiten von Aufträgen. Die Bewegung des Autos sollte aufgrund der Vorgaben in der `nextStep`-Methode ebenfalls auf der Karte sichtbar sein. Zur Vereinfachung werden das SpeedLimit von Straßen und die Deadline von Aufträgen in dieser Hausaufgabe noch nicht beachtet.

Zusätzliche Hilfsmethoden sind erlaubt. Es dürfen auch weitere Konstanten angelegt werden.

Bei der Ausführung und Benutzung des Programms sollten keine Exceptions auftreten. Nach Bearbeitung der Aufgabe sollen alle `TODO`s aus den Kommentaren entfernt sein. Bei der Bewertung der Aufgabe wird auf die Einhaltung des MVC-Patterns geachtet.

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

Achte darauf, das Repository der aktuellen Hausaufgabe zu verwenden.

Anhang

Zur Verfügung gestellte Dateien

- https://github.com/sekassel/pmws2324-files/tree/main/HA07