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 23.11.2023 - 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/qvEIO9bg

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 - Test-First-Prinzip (31P)

Ziel dieser Aufgabe ist es, das Test-First-Prinzip einmal selbst durchzuführen. Hierzu ist folgende Aufgabenreihenfolge vorgegeben:

1. Tests designen
2. Tests implementieren
3. Methoden implementieren
4. Fehler dokumentieren

Jeder Schritt wird im Folgenden näher erläutert.

WICHTIG! Commit Message-Vorgaben

Der letzte Commit zu **jeder Teilaufgabe** soll mit der Commit Message „Finish Step <Nr>“ versehen sein.

Falls nach diesen Commits eine Aufgabe erneut bearbeitet wird, z. B. um einen Fehler zu korrigieren, soll die jeweilige Commit Message einfach noch einmal als letzte Commit Message verwendet werden.

Ohne separate Commits für jede Teilaufgabe können wir die Arbeit nicht nachvollziehen, wodurch diese automatisch mit **0 Punkten** bewertet wird

Also nochmal: Das Ignorieren dieser Vorgabe wird mit 0 Punkten bewertet!

Vorbereitung

In deinem Repository ist folgendes Klassenmodell vorgegeben:

- Die Klasse `City` erbt von Location und repräsentiert eine Stadt.
- Die Klasse `Street` erbt von Location und repräsentiert eine Straße. Die Liste `connectedCities` enthält die Städte, die durch das Straßen-Objekt verbunden werden. Es darf davon ausgegangen werden, dass eine Straße immer **genau zwei Städte** miteinander verbindet. Der Boolean `isBlocked` gibt an, ob die Straße blockiert ist.
- Die Klasse `Location` ist die Elternklasse für Klassen, die Orte repräsentieren.
- Die Klasse `Order` repräsentiert einen Auftrag.

Außerdem ist die Klasse `GameService` gegeben.

Schritt 1: Tests designen

In der ersten Teilaufgabe sollen die Tests und Methoden in den jeweiligen Klassen erstellt, **aber noch nicht implementiert** werden. Ziel des ersten Schrittes ist es, die zu testende Struktur aufzubauen. Nach Bearbeitung hat das Projekt **keine Compiler-Fehler** und die erstellten **Tests schlagen nicht fehl**.

Schritt 1.1: Neue Methoden im GameService

Erstelle unter src/main/java im Package de.uniks.se.transport in der Klasse GameService die folgenden Methoden:

```
public GameService initializeMap() { return this; }
public Street connectCities(City c1, City c2) { return null; }
public void blockStreets() {}
public ArrayList<Location> getPath(City start, City goal) { return null; }
```

Schritt 1.2: Tests

Erstelle unter src/test/java im Package de.uniks.se.transport die Test-Klasse GameServiceTest mit folgenden Methoden:

```
@Test public void testConnectCities() {}
@Test public void testBlockStreets() {}
@Test public void testGetPathSameCity() {
    // get path when c1 and c2 are the same city
}
@Test public void testGetPathWithOneStep() {
    // get path where there exists a street between c1 and c2
}
@Test public void testGetPathWithTwoSteps() {
    // get path where there exists a path
    // from c1 to c2 with another city between them
    // (no direct connection!)
}
@Test public void testGetPathNoConnection() {
    // try to get path between two cities that aren't connected at all
}
@Test public void testGetPathBlocked() {
    // try to get a path where there usually is a path,
    // but currently isn't due to blocked streets
}
```

Achte darauf, dass jede Methode mit @Test annotiert ist.

Committe und pushe die Änderungen auf den main-Branch, bevor du mit der nächsten Teilaufgabe fortfährst. Halte dich an die Vorgaben zur Commit-Message, um zu eindeutig benennen, dass die aktuelle Teilaufgabe (Step 1) abgeschlossen ist.

Schritt 2: Tests implementieren

In der zweiten Teilaufgabe sollen die Tests zu den Methoden aus der vorangegangenen Teilaufgabe implementiert werden. Die Methoden im GameService werden hier noch **nicht** implementiert. Folgende Verhaltensweise ist von den Methoden zu erwarten:

initializeMap()

Nach Aufruf der Methode sollen die `cities` und `streets` des `GameService` folgendermaßen initialisiert werden:

Es gibt die Städte

- Dortmund
- Kassel
- Berlin
- Frankfurt
- München

Es gibt Straßen zwischen den Städten

- Dortmund und Kassel
- Kassel und Berlin
- Dortmund und Frankfurt
- Kassel und Frankfurt

connectCities(City c1, City c2)

Nach dem Aufruf der Methode sollen die Städte `c1` und `c2` über eine neu erstellte Straße verbunden sein. Die Straße wird den `streets` hinzugefügt. Soll als Hilfsmethode beim Initialisieren genutzt werden.

blockStreets()

Nach dem Aufruf der Methode sollen die Straßen zwischen

- Dortmund und Kassel
- Frankfurt und Kassel

blockiert sein.

getPath(City start, City goal)

Es soll ein Weg zwischen der Stadt `start` und der Stadt `goal` zurückgegeben werden. Der Pfad wird als Arrayliste zurückgegeben, die abwechselnd Stadt- und Straßenobjekte beinhaltet, um den Pfad von `start` (erster Eintrag) bis `goal` (letzter Eintrag) zu repräsentieren.

> Sollte es keinen Weg von `start` nach `goal` geben, wird `null` zurückgegeben.
>
> Ist eine Straße blockiert (`isBlocked`), darf diese nicht als nutzbare Straße gewertet werden.

Implementiere die Tests im GameServiceTest. Für den initialen Aufbau der Objektstruktur in den Tests für die Wegfindung darf die initializeMap-Methode des GameService benutzt werden.

Des Weiteren kann es sinnvoll sein, den Ablauf der einzelnen Testmethoden und deren zu testenden Eigenschaften zunächst mittels Kommentaren im Quelltext zu planen. Dies ist allerdings kein Muss.

Es dürfen zusätzliche Hilfsmethoden in der Test-Klasse GameServiceTest implementiert werden.

Nach Bearbeitung dieser Teilaufgabe sollten einige der Tests bei der Ausführung fehlschlagen, da keine Logik in den Methoden implementiert wurde. Einige Tests bleiben allerdings weiterhin grün, da sie z. B. testen, dass keine Veränderung stattfindet.

Committe und pushe die Änderungen auf den main-Branch, bevor du mit der nächsten Teilaufgabe fortfährst. Halte dich an die Vorgaben zur Commit-Message, um zu eindeutig benennen, dass die aktuelle Teilaufgabe (Step 2) abgeschlossen ist.

Schritt 3: Methoden implementieren

In der dritten Teilaufgabe sollen die Methodenrümpfe der Klasse GameService implementiert werden. Diese folgen den schriftlichen Beschreibungen aus der vorangegangenen Teilaufgabe.

Es dürfen zusätzliche Hilfsmethoden in der Klasse GameService implementiert werden.

Nach Bearbeitung dieser Teilaufgabe sollten mehrere oder alle Tests bei der Ausführung erfolgreich durchlaufen. Sollte dies nicht der Fall sein, korrigiert diese Fehler erst in der folgenden Teilaufgabe!

Committe und pushe die Änderungen auf den main-Branch, bevor du mit der nächsten Teilaufgabe fortfährst. Halte dich an die Vorgaben zur Commit-Message, um zu eindeutig benennen, dass die aktuelle Teilaufgabe (Step 3) abgeschlossen ist.

Schritt 4: Fehler dokumentieren

In der vierten Teilaufgabe sollen die konzeptionellen Fehler in Tests oder Logik korrigiert werden. Dabei soll bei jedem gefundenen Fehler ein Kommentar erstellt werden, welcher folgende Punkte beleuchtet:

- Wo trat der Fehler auf?
- Was hat den Fehler verursacht?
- War es ein Konzeptionsfehler im Test oder eine fehlerhafte Implementierung?

Schritt 4 ist vollständig abgeschlossen, sobald keine Fehler mehr korrigiert werden müssen und kein Test mehr fehlschlägt.

Sollte es keine Fehler gegeben haben, darf dieser Schritt weggelassen werden.

Committe und pushe die Änderungen abschließend auf den main-Branch. Halte dich an die Vorgaben zur Commit-Message, um eindeutig zu benennen, dass die aktuelle Teilaufgabe (Step 4) abgeschlossen ist.

Bei der Bewertung wird vor allem darauf geachtet, dass die Reihenfolge des Test-First-Prinzips eingehalten wurde.

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.

IntelliJ IDEA

Bei IntelliJ wird zwischen der kostenlosen „Community“ und der kostenpflichtigen „Ultimate“-Version unterschieden. Es ist für Studierende möglich die „Ultimate“-Version kostenlos zu erhalten, dies ist die „Free Educational License“. Für diese Veranstaltung genügt die kostenlose Version.

- Download: https://www.jetbrains.com/idea/download/
- Free Educational Licenses: https://www.jetbrains.com/community/education/#students
- Unterschiede der Versionen: https://www.jetbrains.com/products/compare/?product=idea&product=idea-ce

Test Driven Development

- Agile Alliance zu TDD: https://www.agilealliance.org/glossary/tdd