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/ws2223/programming-and-modelling/` zu berücksichtigen.

Abgabefrist ist der 01.12.2022 - 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/n-8XoDkO`

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!

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.

Aufgabe 1 - FulibWorkflows (10P)

In dieser Veranstaltung sind Nutzer-Szenarien ein tragender Bestandteil für die Modellierung. Bisher musste aus den Szenarien händisch Objektdiagramme abgeleitet und daraus ein Klassendiagramm erstellt werden. Im Fachgebiet wird aktuell ein Tool entwickelt, welches sich auf die Methodik des Event Stormings fokussiert und daraus automatisiert Objekt- und Klassendiagramme generiert.

In dieser Aufgabe soll das Tool über den Online-Editor der fulib.org-Plattform getestet werden.

`https://fulib.org/workflows`

Erstelle aus dem folgenden Szenario einen Workflow mit fulibWorkflows.

Title: New workplace

Start:	The city of Kassel has multiple public buildings. The University of Kassel and the town hall are public buildings and are located in Kassel. Sebastian and Natascha are working at the University of Kassel. Sebastian owns a black and a white car. Natascha owns a silver car. All cars are located in Kassel. Max has chosen Kassel as his hometown but currently has no workplace. Max owns a blue car which is currently located in the city of Hannover. The City of Kassel therefore has the inhabitants Sebastian, Natascha and Max.
Action:	Max signs a contract at the University of Kassel.
End:	Max is now an employee at the University of Kassel.

Das generierte Klassendiagramm soll aus 4 Klassen und 5 Assoziationen bestehen.

Lege den Inhalt des Online-Editors in der Datei workflow.es.yaml in dem Ordner src/gen/resources ab.

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

Achte darauf, das Repository der aktuellen Hausaufgabe zu verwenden.

Aufgabe 2 - Fulib (18P)

In dieser Aufgabe wird das vollständige Datenmodell des Spiels ”PMon” mit Fulib generiert. Fulib bietet die Grundlage von fulib.org und 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.

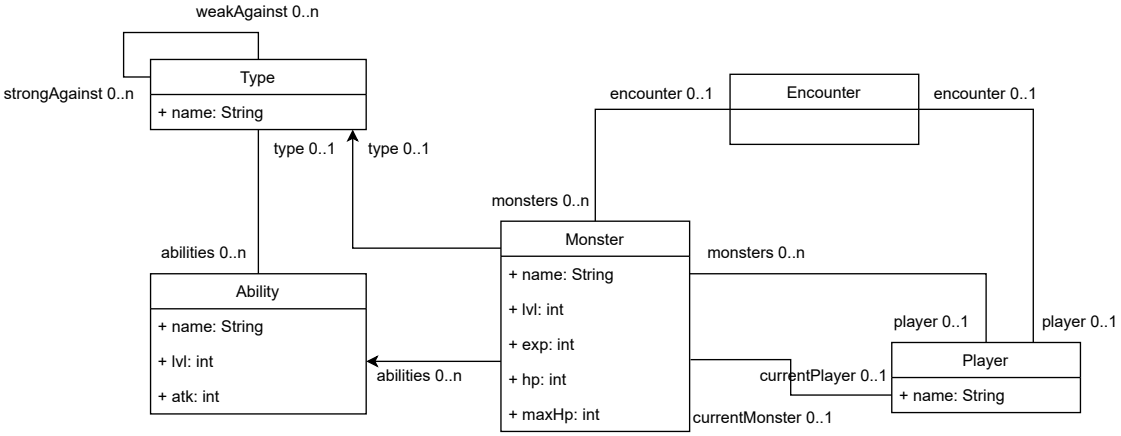

Abbildung 1: „PMon”-Klassendiagramm

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

Achte darauf, das Repository der aktuellen Hausaufgabe zu verwenden.

Aufgabe 3 - Initialisierung (32P)

In dieser Aufgabe soll der Startzustand von PMon in den Klassen Constants und RandomGenerator implementiert werden.

Constants

Zunächst benötigen wir fest definierte Types und Abilities. Diese werden in Constants (zu finden unter src/main/java/de/uniks/pmws2223) definiert. Lege die sechs Typen aus Abbildung 2 als Konstanten an. Verknüpfe sie mit Strong/Weak-Links gemäß der Abbildung in einem static-Block.

Lege nun Konstanten für zwei Abilities vom Typ Normal und jeweils eine Ability für jeden anderen Typ an. Jede Ability soll Level 1 und einen Angriffswert (atk) zwischen 5-10 haben.

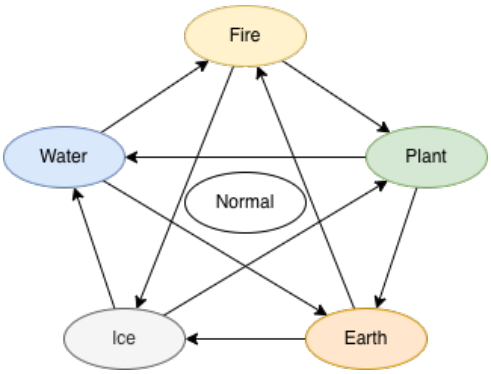

Abbildung 2: Types, Strong und Weak

RandomGenerator

In RandomGenerator (src/main/java/de/uniks/pmws2223/service) wird in der Methode createPlayer anhand des Namens ein Spieler mit Starter-Monsters angelegt. Befolge dazu die Kommentare in der Methode.

Danach wird mit der Methode createEncounter ein Encounter angelegt und mit Monstern gefüllt. Die Kommentare geben auch hier die Vorgaben.

RandomGeneratorTest

Schreibe nun zwei Test-Methoden für die oben genannten Methoden in RandomGeneratorTest (zu finden unter src/test/java/de/uniks/pmws2223/service). Jeder Test soll die zugehörige Methode mit Beispielparametern aufrufen und das Ergebnis mit Asserts prüfen. Es genügt, die explizit gesetzten Werte des erstellten Player (name, currentMonster) bzw. Encounter (player) sowie die Anzahl und Namen der Monster zu überprüfen.

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