Programmentwurf
DnD-Character Manager

Name:	Leon Knorr
Matrikelnummer:	9800840
Abgabedatum:	01. Februar 2018

Inhaltsverzeichnis

1 Einführung

1.1 Übersicht über die Applikation

Die Abgegebene Applikation, „DnD-CharacterManager“, ermöglicht das erstellen und verwalten von Dungeons & Dragons Charactären nach dem 5e Regelwerk. Somit soll sie den bekannten Papiercharacterbogen ablösen und durch eine digitale Version ersetzen. Neben dem persistenten halten von Daten nach dem In-Memory Prinzip moderner Datenbanken, annulliert sie ausserdem das anstrengende Kopfrechnen bei verschiedenen Checks und stellt einen Character Creation wizard bereit, der den Nutzer durch den Charaktererstellungsprozess führt.

1.2 Wie startet man die Applikation?

Die Applikation kann man starten, in dem man die bereitgestellte *.jar Datei in einem Terminal emulator mit dem Befehl `java -jar DnDCharacterManager-jar-with-dependencies` ausführt. Vorraussetzung dafür ist eine valide funktionierende Java installation. Das projekt wurde mittels des `openjdk-18` und einem „language level“ von 17 erstellt. Bitte verwenden sie zum ausführen eine ähnliche Version des jdks, da es ansonsten zu Problemen und inkompatibilität kommen kann.

1.3 Wie testet man die Applikation?

Die Unit Tests der Applikation können über Maven mit Maven test, package und install ausgeführt werden. Die Applikation selbst lässt sich in der Kommandozeile bedienen.

2 Clean Architecture

2.1 Was ist Clean Architecture?

[allgemeine Beschreibung der Clean Architecture in eigenen Worten]

2.2 Analyse der Dependency Rule

[(1 Klasse, die die Dependency Rule einhält und eine Klasse, die die Dependency Rule verletzt); jeweils UML der Klasse und Analyse der Abhängigkeiten in beide Richtungen (d.h., von wem hängt die Klasse ab und wer hängt von der Klasse ab) in Bezug auf die Dependency Rule]

2.2.1 Positiv-Beispiel: Dependency Rule

2.2.2 Negativ-Beispiel: Dependency Rule

2.3 Analyse der Schichten

[jeweils 1 Klasse zu 2 unterschiedlichen Schichten der Clean-Architecture: jeweils UML der Klasse (ggf. auch zusammenspielenden Klassen), Beschreibung der Aufgabe, Einordnung mit Begründung in die Clean-Architecture]

2.3.1 Schicht: [Name]

2.3.2 Schicht: [Name]

3 SOLID

3.1 Analyse Single-Responsibility-Principle (SRP)

[jeweils eine Klasse als positives und negatives Beispiel für SRP; jeweils UML der Klasse und Beschreibung der Aufgabe bzw. der Aufgaben und möglicher Lösungsweg des Negativ-Beispiels (inkl. UML)]

3.1.1 Positiv-Beispiel

3.1.2 Negativ-Beispiel

3.2 Analyse Open-Closed-Principle (OCP)

[jeweils eine Klasse als positives und negatives Beispiel für OCP; jeweils UML der Klasse und Analyse mit Begründung, warum das OCP erfüllt/nicht erfüllt wurde – falls erfüllt: warum hier sinnvoll/welches Problem gab es? Falls nicht erfüllt: wie könnte man es lösen (inkl. UML)?]

3.2.1 Positiv-Beispiel

3.2.2 Negativ-Beispiel

3.3 Analyse Liskov-Substitution- (LSP), Interface-Segreggation- (ISP), Dependency-Inversion-Principle (DIP)

$jeweilseineKlassealspositivesundnegativesBeispielfürentwederLSPoderISPoderDIP); jeweilsU$

$Anm.: es darf nur ein Prinzip ausgewählt werden; es darf NICHT z.B. ein positives Beispiel für LSP un$

3.3.1 Positiv-Beispiel

3.3.2 Negativ-Beispiel

4 Weitere Prinzipien

4.1 Analyse GRASP: Geringe Kopplung

[jeweils eine bis jetzt noch nicht behandelte Klasse als positives und negatives Beispiel geringer Kopplung; jeweils UML Diagramm mit zusammenspielenden Klassen, Aufgabenbeschreibung und Begründung für die Umsetzung der geringen Kopplung bzw. Beschreibung, wie die Kopplung aufgelöst werden kann]

4.1.1 Positiv-Beispiel

4.1.2 Negativ-Beispiel

4.2 Analyse GRASP: Hohe Kohäsion

[eine Klasse als positives Beispiel hoher Kohäsion; UML Diagramm und Begründung, warum die Kohäsion hoch ist]

4.3 Don't Repeat Yourself (DRY)

[ein Commit angeben, bei dem duplizierter Code/duplizierte Logik aufgelöst wurde; Code-Beispiele (vorher/nachher); begründen und Auswirkung beschreiben]

5 Unit Tests

5.1 10 Unit Tests

[ein Commit angeben, bei dem duplizierter Code/duplizierte Logik aufgelöst wurde; Code-Beispiele (vorher/nachher); begründen und Auswirkung beschreiben] <Todo Tabelle hier>

5.2 ATRIP: Automatic

[Begründung/Erläuterung, wie 'Automatic' realisiert wurde]

5.3 ATRIP: Thorough

[jeweils 1 positives und negatives Beispiel zu 'Thorough'; jeweils Code-Beispiel, Analyse und Begründung, was professionell/nicht professionell ist]

5.4 ATRIP: Professional

[jeweils 1 positives und negatives Beispiel zu 'Professional'; jeweils Code-Beispiel, Analyse und Begründung, was professionell/nicht professionell ist]

5.5 Code Coverage

[Code Coverage im Projekt analysieren und begründen]

5.6 Fakes und Mocks

[Analyse und Begründung des Einsatzes von 2 Fake/Mock-Objekten; zusätzlich jeweils UML Diagramm der Klasse]

6 Domain Driven Design

6.1 Ubiquitous Language

[4 Beispiele für die Ubiquitous Language; jeweils Bezeichung, Bedeutung und kurze Begründung, warum es zur Ubiquitous Language gehört]

Bezeichnung	**Bedeutung**	**Begründung**
Klasse	tbw	tbw
Rasse	tbw	tbw
Equipment	tbw	tbw
Spell	tbw	tbw

6.2 Entities

[UML, Beschreibung und Begründung des Einsatzes einer Entity; falls keine Entity vorhanden: ausführliche Begründung, warum es keines geben kann/hier nicht sinnvoll ist]

6.3 Value Objects

[UML, Beschreibung und Begründung des Einsatzes eines Value Objects; falls kein Value Object vorhanden: ausführliche Begründung, warum es keines geben kann/hier nicht sinnvoll ist]

6.4 Repositories

[UML, Beschreibung und Begründung des Einsatzes eines Repositories; falls kein Repository vorhanden: ausführliche Begründung, warum es keines geben kann/hier nicht sinnvoll ist]

6.5 Aggregates

[UML, Beschreibung und Begründung des Einsatzes eines Aggregates; falls kein Aggregate vorhanden: ausführliche Begründung, warum es keines geben kann/hier nicht sinnvoll ist]

7 Refactoring

7.1 Code Smells

[jeweils 1 Code-Beispiel zu 2 Code Smells aus der Vorlesung; jeweils Code-Beispiel und einen möglichen Lösungsweg bzw. den genommen Lösungsweg beschreiben (inkl. (Pseudo-)Code)]

7.2 2 Refactorings

[2 unterschiedliche Refactorings aus der Vorlesung anwenden, begründen, sowie UML vorher/nachher liefern; jeweils auf die Commits verweisen]

8 Entwurfsmuster

[2 unterschiedliche Entwurfsmuster aus der Vorlesung (oder nach Absprache auch andere) jeweils sinnvoll einsetzen, begründen und UML-Diagramm]

8.1 Entwurfsmuster: [Name]

8.2 Entwurfsmuster: [Name]