Software Entwicklung 2

Wintersemester 19/20

Anton Richter, ar140@hdm-stuttgart.de
Kai Schwabe, ks204@hdm-stuttgart.de
Filippo Matraxia, fm080@hdm-stuttgart.de

https://gitlab.mi.hdm-stuttgart.de/swifters/inquiz.git

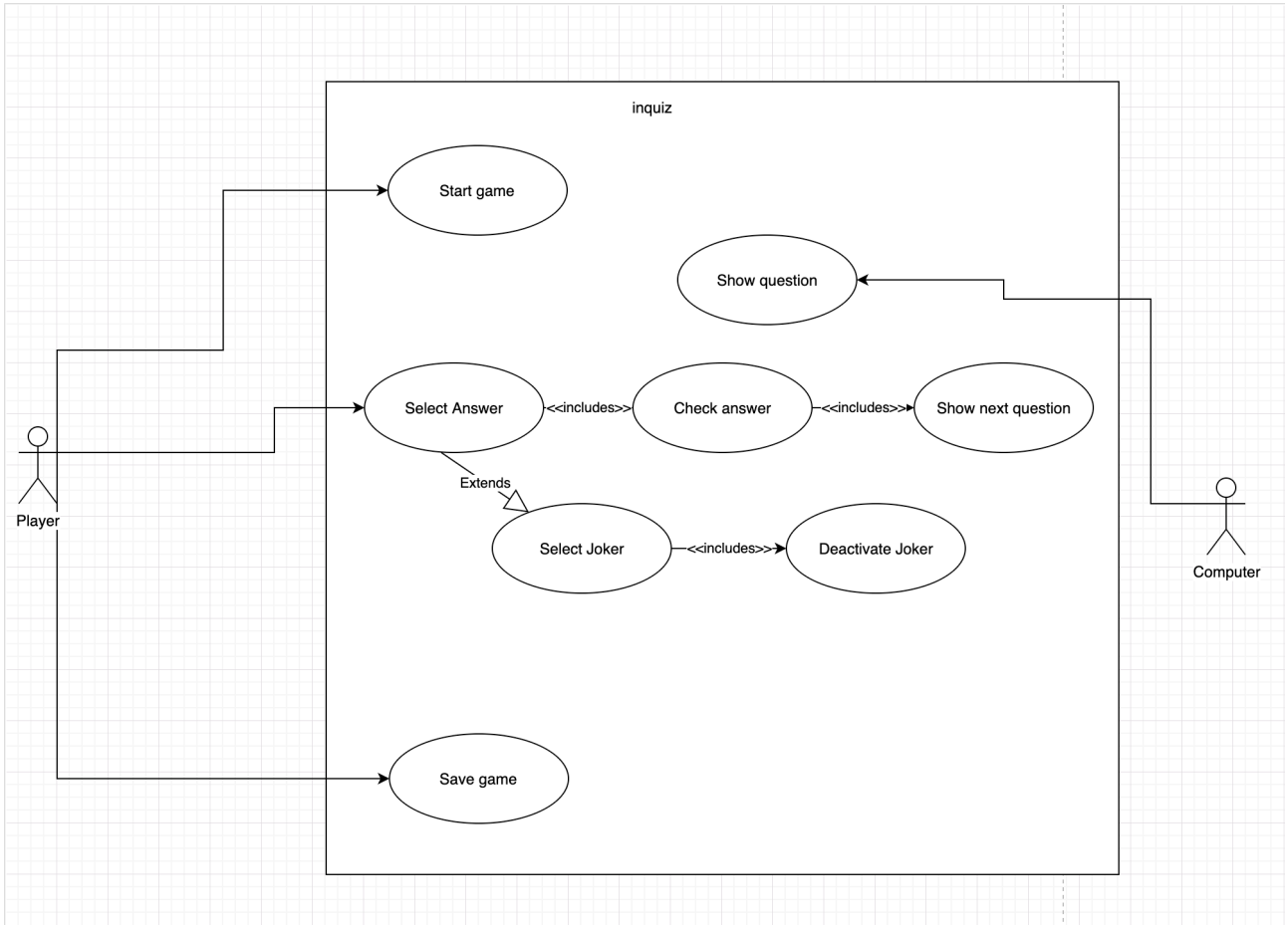
1. Kurzbeschreibung:

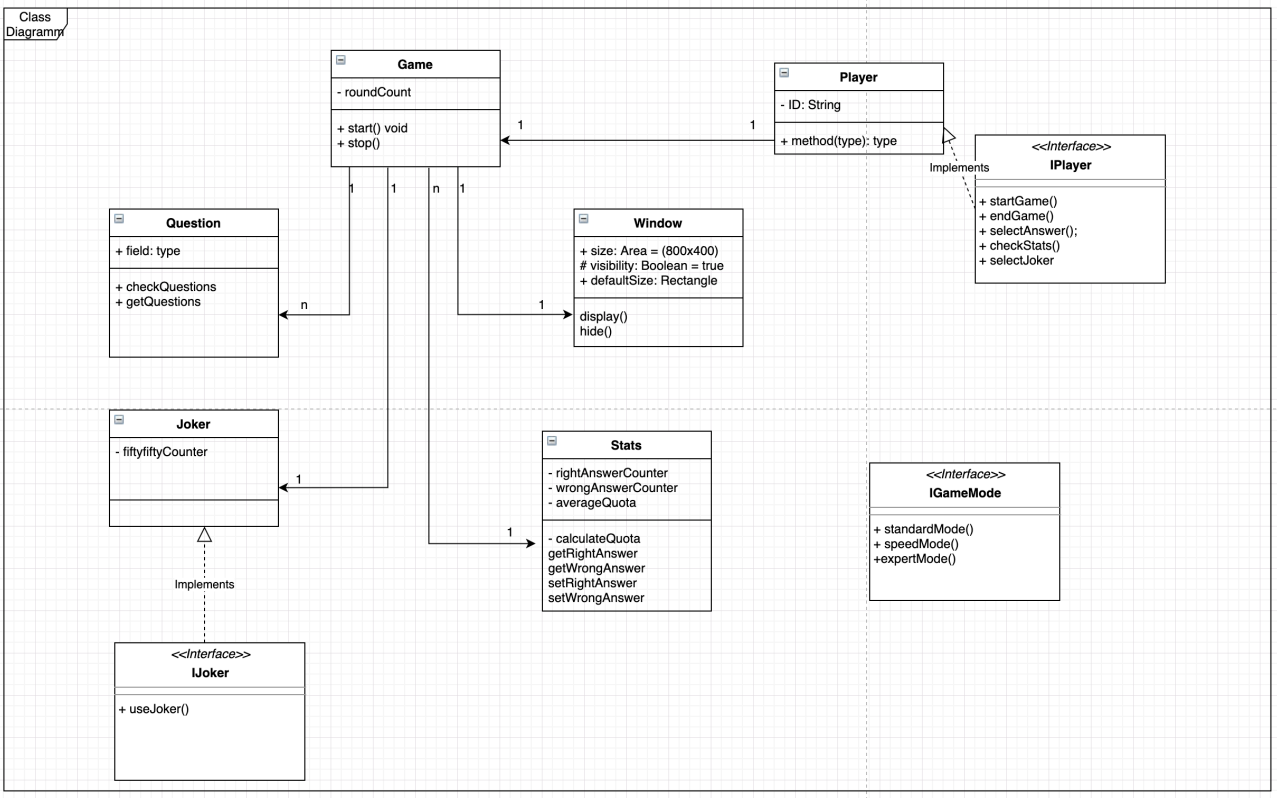
Bei unserem Projekt handelt es sich um ein Quiz, welches für die Vorlesung Software-Entwicklung 2 (EDV-Nr.: 113215) im zweiten Semester an der Hochschule der Medien im Bachelor-Studiengang Medieninformatik/Mobile Medien erstellt wurde.
Das Quiz ist standardmäßig aufgebaut, wie man es zum Beispiel von “Wer wird Millionär” kennt, man erhält eine Frage mit vier Antwortmöglichkeiten. Weiterhin hat man die Möglichkeit sich zwischen drei verschiedenen Schwierigkeitsgraden zu entscheiden (Standard, Speed, Expert). Mit
ansteigender Schwierigkeit wird sowohl die Antwortzeit als auch die Anzahl der Joker, die einem zur Verfügung stehen verringert. Jeder Joker kann nur einmal verwendet werden. Am Ende erhält man eine Übersicht, wie viele Fragen man richtig und falsch beantwortet hat.

2.Startklasse

Die Main-Methode befindet sich in der Klasse **App** im Package **de.hdm_stuttgart.mi.classes**.

3. UML-Klassendiagramm





Das Diagramm wurde am Ende des Projektes erstellt

4. Stellungnahme

Architektur:

Für das Quiz wurden zwei Interfaces implementiert **IGamemode** und **IJoker**. Für beide Interfaces wurde eine Factory angelegt.

- **IGamemode** wird von **GameMode** implementiert
- **IJoker** wird jeweils von **FiftyFiftyJoker**, **SkipQuestionJoker** und **TimeJoker** implementiert

Weiterhin haben wir mehrere Packages in unserem Projekt angelegt:

- **classes:** Enthält alle Hauptklassen, die nicht zur GUI gehören
- **exceptions:** Enthält unsere eigene Exception
- **interfaces:** Enthalten die Interfaces
- **gameModeFactory:** Enthält die Factory und die Klasse die das Interface IGamemode implementieren
- **JokerFactory:** Enthält die Factory und die Klasse die das Interface IJoker implementieren
- **guiHandler:** Enthalten alle Klassen die für die GUI relevant sind

Clean Code:

Das Projekt enthält einige wenige Variablen, die aus Gründen der Performance static sind. Denkbare wäre gewesen, diese Variablen jedesmal über einen Getter anzufordern, da man hierfür aber wieder zwingend ein Objekt der entsprechenden Klasse erstellt werden müsste. Aufgrund des Hauptgedanken der Erweiterbarkeit wurde hierauf verzichtet, man mit immer mehr Klassen immer mehr Objekte erstellen müsste.
Bei verwendeten Getter Methoden wurde darauf geachtet, dass keine Referenz übergeben wird.

Exception:

Eine eigene Exception wurde für das Quiz entwickelt **IllegalFactoryArgument** diese wird geworfen, sobald man versucht einen nicht existenten Gamemode zu erstellen.

GUI:

Die GUI besteht aus mehreren Stages, die jeweils über eine eigen FXML-Datei erstellt wurde. Diese Dateien befinden sich im Verzeichnis **ressources > fxml.**

.Logging:

- Die Exception wurde mit fatal geloggt
- Allgemeines Debugging (z.B in classes > Game)
- Der Thread wurde ebenso geloggt (classes > Music)
- Die Konfiguration kann unter **ressources > log4j2.xml** eingesehen werden

Thread:

Der Thread Music dient dazu während des Spiels, wie der Name suggeriert, Musik im Hintergrund abzuspielen. Es besteht jedoch die Möglichkeit die Musik stummzuschalten. Der Thread wird in der Klasse **APP** gestartet.

Tests:

Alle wichtigen Methoden wurden mit Hilfe von Unit Tests getestet. Die Tests der einzelnen Klassen befinden sich im Package **src > test**. Negativ Tests wurden ebenfalls implementiert, zum Beispiel für die selbstgebaute Exception **(IllegalFactoryArgumentTest)**. Selbstverständlich wurde auch der Thread getestet.

Streams:

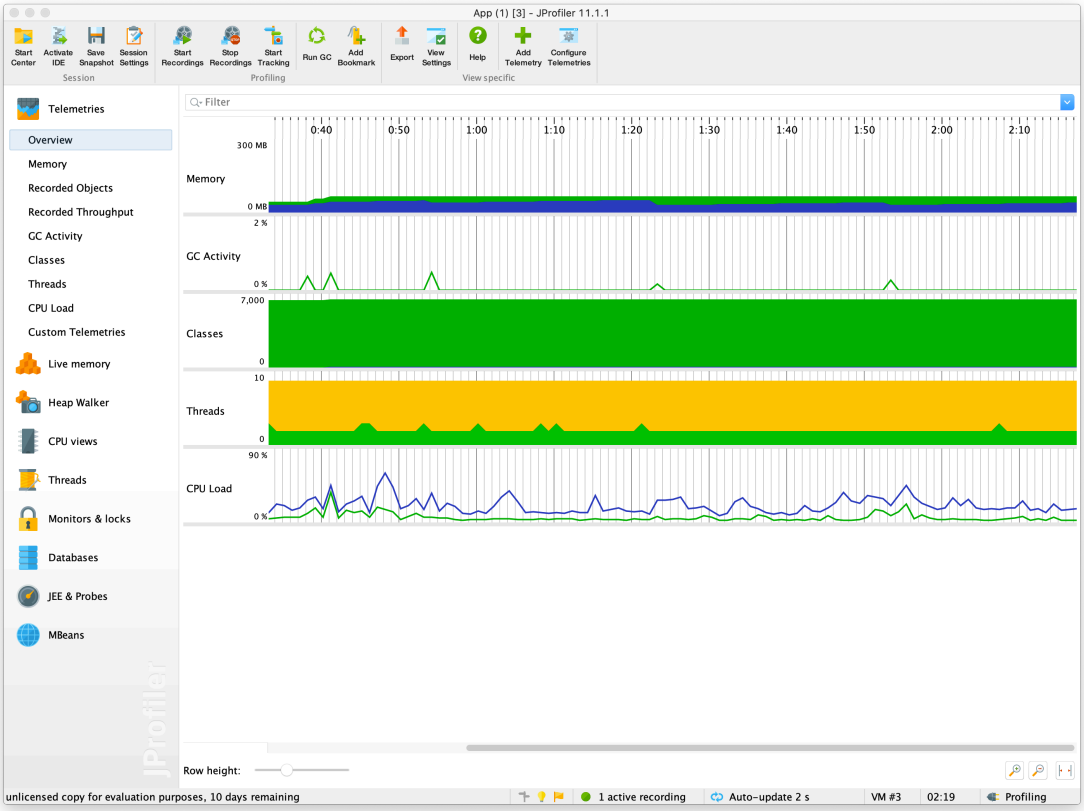
Stream und Lambda Funktionen wurden ebenfalls implementiert.

- Streams sind unter anderem in den Folgender Klassen zu finden:
  - **de.hdm_stuttgart.mi > classes > Quiz:** Hier wird ein Array mit 10 zufälligen Indizes für die Fragen gefüllt.

- Lamda Funktionen sind unter anderem in Folgender Klasse zu finden:
  - **de.hdm_stuttgart.mi > guiHandler >Gamecontroller:** Diese werden zum Beispiel für Animationen verwendet.

5. Profiling

Das Profiling wurde mit dem Programm „JProfiler“ erstellt. Das Ergebnis ist in nachfolgender Abbildung zu sehen.



Die große Anzahl der Klassen ist vermutlich auf JavaFX zurückzuführen. Es ist unklar welche Klassen beim Start einer JavaFX im Hintergrund aufgerufen werden. Dasselbe gilt für die wartenden Threads.

6. Bewertungsbogen

Der Bewertungsbogen ist ebenfalls im Repository zu finden.

Vornamae	Nachname	Kürzel	Matrikelnummer	Projekt	Arc.	Clean Code	Doku	Tests	GUI	Logging/Excep.	UML	Threads	Streams	Profiling	Summe - Projekt	Kommentar	Projekt-Note
Anton	Richter	ar140	38630	inquiz	3	1	3	3	3	3	3	2	2	3	26,00		1,70
Kai	Schwabe	ks204	38598	inquiz	3	1	3	3	3	3	3	2	2	3	26,00		1,70
Filippo	Matraxia	fm080	38498	inquiz	3	1	3	3	3	3	3	2	2	3	26,00		1,70
															0,00		5,00