SE 2-Projekt

Projektname: Turnierverwaltung

GIT: <https://version.mi.hdm-stuttgart.de/git/SE2Turnierverwaltung>

Autoren:

|  |  |  |
| --- | --- | --- |
| Name | Kürzel | Matrikel-Nr. |
| Axel Forstenhäusler | af073 | 31105 |
| Daniel Elsenhans | de024 | 31098 |
| Samuel Hack | sh245 | 31783 |
| Vincenzo Piccolo | vp019 | 31109 |

## Abstract

Unser Projekt umfasst eine Turnierverwaltung mit bis zu 32 Mannschaften. Diese können in zwei Verschiedenen Turnier-Modi gegeneinander antreten. Der eine Modus ist ein sogenanntes KO-Turnier, bei dem der Sieger in die nächste Runde gelangt. Der andere Modus ist ein Gruppenturnier, bei dem die Mannschaften für die Vorrunde in Gruppen eingeteilt werden und die Spiele gegeneinander austragen können. Anschließend treten die Gruppenbesten in einem KO-Modus gegeneinander an.

## Startklasse

Die Startklasse befindet sich in *main/Main.java*

## Besonderheiten

Läuft.

* Turnierbaum
* Ergebnisse simulieren
* Rangliste

UML

## Erweitere Funktionalitäten

### Package-Struktur

Alle Klassen, die im Package *GUI.\** liegen, sind für die grafische Darstellung der GUI zuständig. Alle anderen Funktionalitäten sind von der GUI abgekapselt. Die Turnier-Klassen befinden sich in *backend.\*.*

### Interfaces

Für die Matches (einzelne Spiele, die ausgetragen werden) haben wir ein Interface namens IMatch (*backend.interfaces/IMatch.java*) erstellt, das den Zugriff auf diese gewährleistet. Alle Unterklassen von Match greifen auf das Interface zu.

### Factory

Zusätzlich haben wir eine Factory-Design-Pattern implementiert (*backend.turnier/MatchFactory.java*). Diese gibt ein Objekt zurück, welches das Interface IMatch implementiert hat. Die build-Methode ist überladen, so dass die Factory durch verschiedene Parameter entscheidet, welchen Match-Typ sie erstellt (Match, FolgeMatch oder FinalMatch).

## Vererbung

Für die Vererbung haben wir eine Klasse Match (im Package backend.turnier), von der die Klassen FolgeMatch und FinalMatch abgeleitet sind. FolgeMatches haben jeweils zwei Turniere, die vorausgehen. Das FinalMatch stellt das letzte Match des Turnieres dar.

## Exceptions

Wir haben zwei eigene Exception-Klassen erstellt (im Package *backend.exception*). Die *GameNotFinishedException* wird aufgerufen, wenn ein Ergebnis des Matches abgefragt wird, aber das Turnier noch nicht beendet wurde. Die *GameUntentschiedenException* wird aufgerufen, wenn bei einer KO-Runde kein eindeutiger Sieger feststeht. Die Exceptions sollten nicht auftreten, allerdings kann es bei Tests passieren, wenn manuell Daten erstellt werden und müssen daher behandelt werden.

## Tests

Die Testklasse befindet sich unter *src/test/java/AppTest.java*. und beinhaltet folgende Tests:

* incrementTore: Mannschaften und Match wird manuell erstellt. Tore werden gesetzt, erwartete Toranzahl abgefragt und verglichen
* testeGruppenSieger: Mannschaften manuell gesetzt, Matches manuell erstellt und Tore manuell über Methoden hinzufügen. Danach Gruppensieger mit erwartetem Gruppensieger vergleichen
* testeGruppenArrayList: Wie testeGruppenSieger, nur wird nach den beendeten Spielen die erwartete Rangfolge im Array mitverglichen
* testeMatchSiegerM1: Match wird manuell erstellt. Danach werden Tore hinzugefügt, der Sieger festgesetzt und kontrolliert ob die richtige Mannschaft als Sieger/Verlierer gesetzt wurde
* testeMatchSiegerM2: Wie testeMatchSiegerM1, nur umgekehrt
* testeMatchUnentschieden: Wie testeMatchSiegerM1, nur wird ist beim Versuch bei gleicher Toranzahl das Spiel zu beenden. Es wird abgefragt ob die GameUnentschiedenException korrekt geworfen wird
* testeMatchUnentschiedenKO: Wie testeMatchUnentschieden, nur mit anderer Turnierart. Unendschieden ist erlaubt, erwartete Boolean Flag über getUnentschieden() verglichen
* gesamteKlasseMatch: Mischung aus aller oberen Tests

Logging

* Streams
* Threads
* Lambda-Notation
* GUI