**SE2-Projekt: Dokumentation**

**Autoren**

Jolanda Jerg (jj033)

Rajan Serafi (rs113)

Freddy Newton Akdogan (fa019)

Georg Habermann (gh014)

**Projektname**

Raumschiffe versenken (Raumschiffe\_versenken)

**Pfad zum Repository**

https://gitlab.mi.hdm-stuttgart.de/gh014/Raumschiffe\_versenken.git

**Abstract**

Unser Projekt nennt sich Raumschiffe versenken und ist im Prinzip gleich aufgebaut wie das bekannte Spiel Schiffe versenken. Es handelt sich um ein Spiel, in welchem zwei Spieler teilnehmen können. Beim Start werden die Raumschiffe automatisch vom Computer gesetzt. Diese werden für jeden Spieler unten links im Feld angezeigt, sodass die eigenen Raumschiffe sichtbar sind. Unser Feld hat eine Größe von 10 mal 10 Felder, also insgesamt 100 Felder.

Die Player spielen nacheinander, das heißt immer nur eine Person ist an der Reihe und sieht den Bildschirm, der andere muss warten bis er am Zug ist.

Spieler Eins fängt an mit schiessen und wählt ein Feld aus, um ein mögliches Raumschiff zu versenken. Es gibt vier verschiedene Raumschiffarten: fünf Jäger der Länge eins, vier Bomber der Länge zwei, zwei Fregatten der Länge vier und einen Zerstörer der Länge fünf.

Ist ein Schiff versenkt, so verfärbt sich das Feld rot. Trifft der Spieler daneben ins Wasser, so erscheint ein Kreuz auf dem Feld. Bei jedem Treffer darf nochmal geschossen werden, bis kein Raumschiff mehr versenkt wird.

Das Spiel ist beendet, sobald einer der beiden Player alle Raumschiffe versenkt hat (26 Treffer erzielt hat).

**Startklasse**

SpielSystemMainFX

**Besonderheiten**

was funktioniert nicht?

**UML-Klassendiagramm**

**Stellungnahmen**

**Interfaces/Vererbung**

In unserem Projekt befindet sich ein Interface in dem Package Interface\_Factory und die Interface-Klasse heißt RaumSchiff. Wir verwenden die Funktionalität des Interfaces, um die verschiedenen Raumschiffsarten zu erstellen. Alle Schiffe haben gleiche Eigenschaften (gleiche Methoden), nur ihre Implementierung ist unterschiedlich. Deswegen besitzt das Interface leere Methoden, welche in den einzelnen Schiffsklassen angepasst werden.

Zum Beispiel hat jedes Raumschiff eine eigene Länge, die mit der Methode getLänge( ) und setLänge( ) ausgegeben und geändert werden kann. Alle Klassen, die das Interface RaumSchiff implementieren, müssen alle Methoden übernehmen und implementieren.

In unserem Projekt ist das Interface sinnvoll, weil wir damit nicht gleiche Methoden bei jedem Raumschiff einzeln erstellen müssen und all unsere Schiffe einige Gemeinsamkeiten haben.

**Package-Struktur**

All unsere Packages befinden sich in dem Hauptpackage RaumSchiffeVersenken.

Aufgeteilt sind diese in Core, Exception, GUISteuerung, Interface\_Factory und SchiffsArten.

Diese Unterteilung bezieht sich auf die verschiedenen Inhalte des Projekts.

Im Package Core, also im „Kern“, befinden sich die wichtigsten Klassen, unter anderem die Main-Klasse, die Ablauf- und Funktionsklasse für das Feld. Ohne dem Core-Package ist unser Projekt nicht ausführbar.

Unsere eigene Exception ist in dem Package Exception angelegt.

Die verschiedenen GUI-Fenster sind in dem Package GUISteuerung enthalten. Hier befinden sich die einzelnen Klassen, welche die FXML-Dateien mit dem Programmcode verbinden.

Wie der Name Interface\_Factory bereits verrät, befinden sich in diesem Paket unser Interface und die Factory für die Erstellung der einzelnen Raumschiffsarten.

Im letzten Paket ist eine Klasse für jede Schiffsart (Bomber, Fregatte, Jaeger, Zertsoerer).

Für uns erscheint diese Struktur als geordnet und logisch, da jedes Package einem bestimmten Thema zugeordnet ist.

In dem Package resources befinden sich alle Dateien, die für unsere GUI notwendig sind. Darunter befinden sich Bilder, FXML-Dateien, Geräusche und Musik.

Alle Bilder sind als PNG-Dateien in dem Ordner bilder gespeichert.

Die einzelnen grafischen Fenster der GUI wurden als FXML-Dateien erstellt und sind in dem Package fxml gesammelt.

Außerdem sind Geräusche in unser Projekt implementiert, welche sich als wav-Dateien, in dem Paket geräusche befinden.

Während dem Spiel läuft eine Hintergrundmusik, die im Paket musik als mp3 hinterlegt ist.

Unsere Testklasse (FeldTest) befindet sich in einer extra Klasse (test) und in dem Package RaumSchiffeVersenken.Core.

**Exceptions:**

In dem Package Exception haben wir unsere eigene Exception erstellt. Diese erbt von java.lang.Exception und wird in der Methode schifffSetzenAutomatisch in der Klasse FeldFX geworfen und angewendet, wenn ein Schiff nicht richtig gesetzt wird.

Ansonsten verwenden wir die IOException (Input-Output-Exception) bei Methoden, wo Ein- oder Ausgaben zu Fehlern führen können.

ausführlicher??

**Grafische Oberfläche (JavaFX)**

**Logging**

Am Beginn einer Klasse wird der Logger erstellt, der mit der log4j2.xml verknüpft ist. Diese xml-Datei befindet sich im Package resources unter dem Namen log4j2.xml. In dieser Datei ist der Filename A1.log festgelegt, welcher für die Ausprintung der Loggings verantwortlich ist. Außerdem legt die xml fest, ab welchem Level geloggt wird in der AppenderRef und ab welchem Level die Loggings auf die Konsole geschrieben werden. Bei uns im Projekt darf ab dem Level „debug“ geloggt werden und auf der Konsole erscheinen alle Loggings ab „info“.

Im Code verwenden wir den Logger überall dort, wo stattdessen „System.out.println“ stehen würde. Er dient dafür Infos auf der Konsole auszugeben, wenn etwas im Code passiert oder wenn der User eine Aktion auslöst. Bei jedem Logger geben übergeben wir eine Nachricht, dass wir wissen was zu diesem Zeitpunkt im Code passiert.

Bei jeder Exception verwenden wir das Level „error“ für den Logger, da beim Auslösen einer Exception ein Fehler auftritt und dieser behoben werden muss. Bei allgemeinen Aktionen, die der User oder der Computer auslösen verwenden wir meist das Level „info“, dass wir wissen was passiert ist.

Ist das richtig!?

**UML**

**Threads**

**Streams und Lambda-Funktionen**

**Dokumentation und Test-Fälle**

**Factories**

Unsere Factory ist in dem Package Interface\_Factory in der Klasse SchiffFactory zu finden. Die Factory ist ein Hilfsmittel zum Erzeugen von Objekten der vier verschiedenen Schiffsarten.

Anhand eines Kriteriums (Integer-Werten) wird aus einem Switch-Statement ausgewählt, welches Raumschiff gebildet werden soll. Dadurch ist es möglich, Objekte auf eine einfache und schnelle Art und Weise für die Weiterverwendung zu erzeugen.

Die Methode nennt sich getRaumschiff und gibt ein Objekt eines Schiffes zurück. Durch den Methodenaufruf wird nach gewünschter Schiffsart eine Zahl als Argument übergeben: bei einer eins wird ein Jäger erstellt, bei einer zwei ein Bomber, bei einer vier eine Fregatte und bei einer fünf ein Zerstörer. Das Kriterium ist das Entscheidungsmerkmal für jedes Schiff.