Software-Entwicklung 2

Sommersemester 2019 – Walter Kriha

BATTLESHIPS

MI7-2

gitlab.mi.hdm-stuttgart.de/lh108/battleshipproject

Leon Heinrich, lh108
Son Hai Vu, sv048
Sebastian Dubiel, sd114

1. Kurzbeschreibung:

Unser Projekt handelt von einer digitalen Version des Klassikers „Schiffe versenken“ oder auch „Battleships“.

Zuerst muss der Spieler seine Schiffe auf ein vorgegebenes Spielfeld setzen und kann sie hierbei vertikal oder auch horizontal platzieren. Dabei ist zu beachten, dass sich keine Schiffe direkt berühren dürfen.
Nachdem der Spieler seine Schiffe gesetzt hat, setzt der Computer-Spieler seine Schiffe automatisch nach dem Zufallsprinzip und die Angriffsphase kann beginnen.
In der Angriffsphase werden nun beide Felder angezeigt. Das Feld des Spielers auf der rechten Seite (mit den gesetzten Schiffen) und auf der linken Seite das Feld des Computers, welches dem Spieler leer angezeigt wird (verborgene Schiffe).
Der Spieler beginnt eine Stelle des Feldes (des Gegners) auszuwählen. Je nachdem ob er ein Schiff getroffen hat oder nicht, darf der Spieler noch einmal schießen. Falls er verfehlt haben sollte, ist der Computer an der Reihe und schießt so oft, bis er anstatt eines Schiffes, ein Wasser-Feld trifft.
Derjenige der zuerst alle Schiffe des Gegners zerstört hat, gewinnt das Spiel.

2. Startklasse

Main-Methode befindet sich in der Klasse ***GuiDriver*** im Package **Gui**

3. Besonderheiten

Der Computer-Spieler besitzt keine Intelligenz, sondern agiert rein zufällig

Wir nutzen JDK 11

Kennzeichnungs-Hinweise:

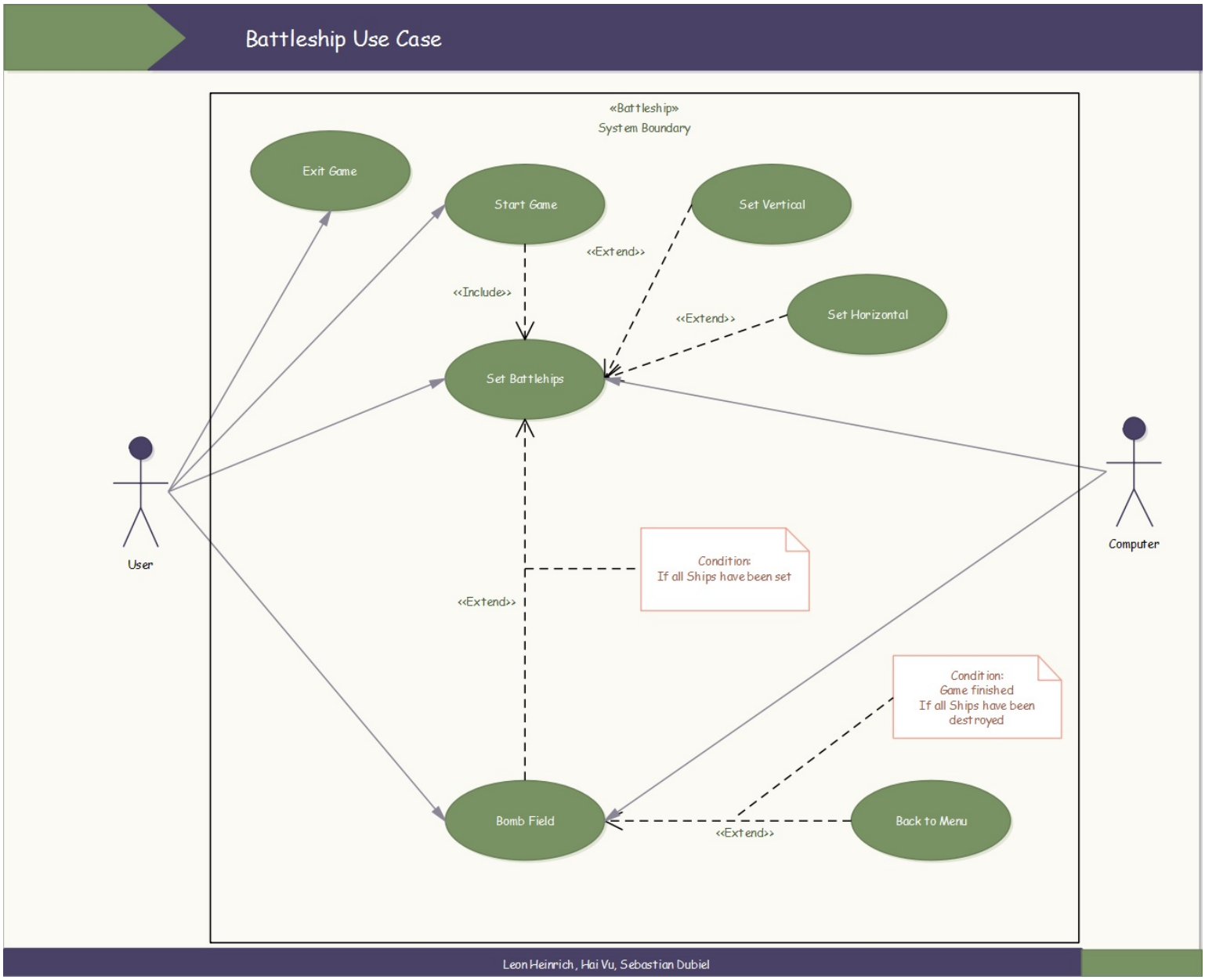
Packages

Klassen

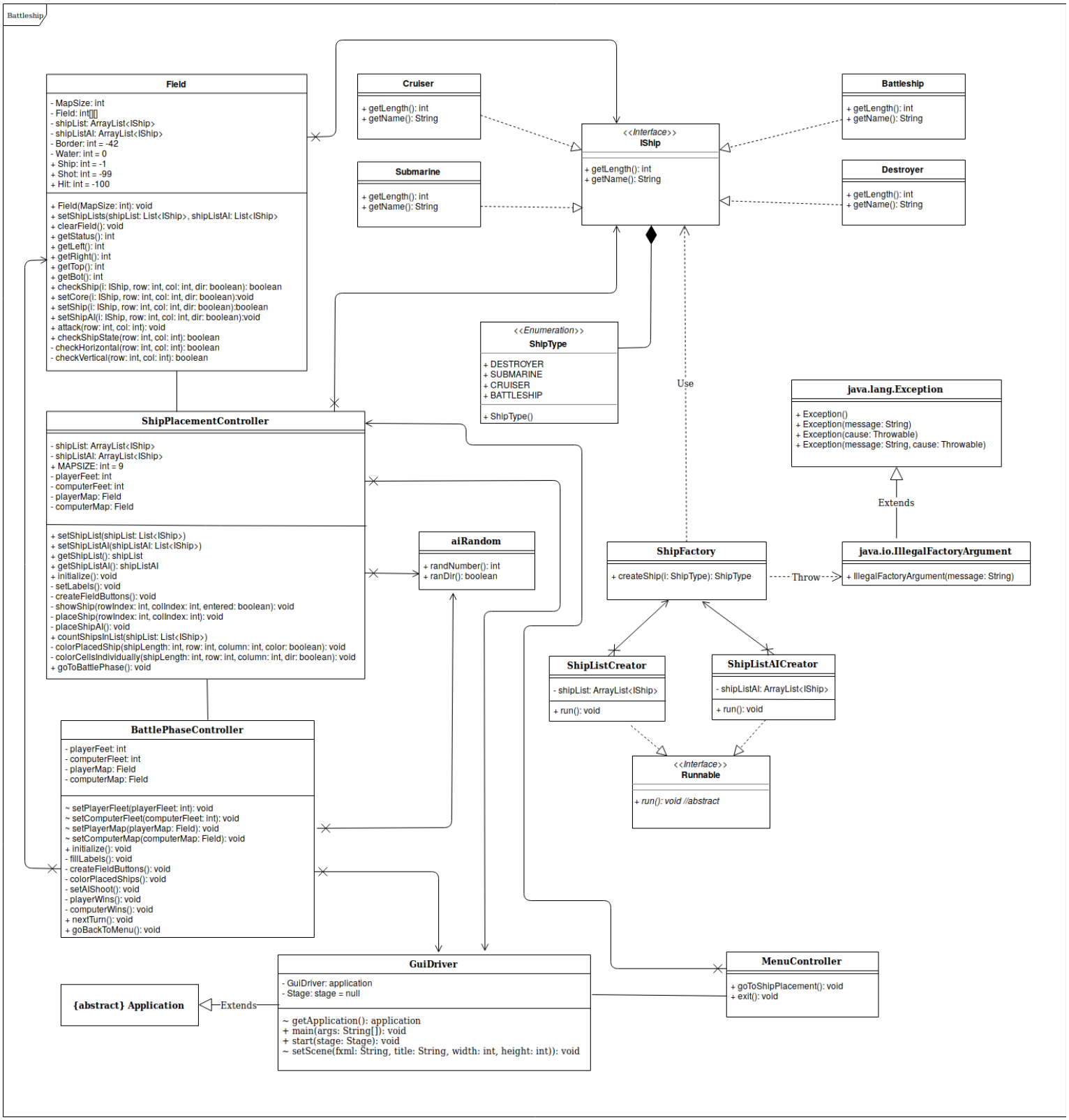
Methoden/Variablen

4. UML

4.1 Use-Case Diagramm



4.2 Klassendiagramm



5. Stellungnahmen

Architektur

- Interfaces: Eigenes Interface IShip (**Interfaces → *IShip***) mit erweiterbarer Architektur (weitere Schiffstypen implementierbar) + Factory
  → „ShipFactory“ (**Ships → *ShipFactory***),
  sowie Implementierung von Java-eigenen Interfaces:

  (1) „List<E>“ (**Gui → *ShipPlacementController***
  →*shipList*, *shipListAI*): beinhalten von der ***ShipFactory*** erstellte Schiffe
  (2) „Runnable“ (**Threads → *shipListCreator***, ***shipListAICreator***), zur Erstellung von Threads

- Vererbung: Unsere eigene Exception muss von Exception erben:
  Exceptions → *IllegalFactoryArgument*

- Struktur: Im Ordner **sd2** befinden sich 5 Packages:

  (1) Exceptions: eigene Exceptions
  (2) Gui: Controller und Gamelogic der Gui (FXML)
  (3) Interfaces: eigenes Interface ***IShip***
  (4) Ships: Klassen der Schiffstypen + ***ShipFactory***
  (5) Threads: eigene Threads enthalten

  Sowie zwei weitere einzelne Klassen.

- FXML: Die FXML-Dateien für die Gui befinden sich im Ordner:
  resources → fxml
  Die zugehörige CSS-Datei befindet sich im gleichen Package

Clean Code

Es werden keine public-Member verwendet, abgesehen von final-Variablen, die nicht schreibbar sind. Wir haben zudem einige statische Methoden genutzt:

(1) Gui → *BattlePhaseController*
(2) Gui → *ShipPlacementController*
→ Jeweils getter/setter-Methoden von statischen Variablen

(3) Ships → *ShipFactory* → *createShip*:
zur Instanziierung der Schiffstypen

Bei getter-Methoden haben wir stets darauf geachtet, nur final-Werte (z.B. ***Field*** → *getStatus*) oder Kopien von schreibbaren Membern (**Gui** → ***ShipPlacementController*** → *getShipList* bzw. *getShipListAI*) zurückzugeben.

Bezüglich des Punktes „loose coupling“ haben wir stets Interfaces auf der linken Seite des „=“-Zeichen verwendet um die Austauschbarkeit zu gewährleisten und Abhängigkeiten zwischen Klassen zu reduzieren:

(1) Gui → ***ShipPlacementController*** → *shipList bzw. shipListAI*
(2) Threads → ***ShipListCreator*** *bzw.* ***shipListAICreator***

Tests

Wichtige Klassen/Methoden wurden mit JUnit-Tests im Ordner **test** erstellt:

(1) ***TestField:*** wichtigste Methoden von ***Field*** getestet und kommentiert
Negativtests: in jeder Methode wurden ein oder mehrere Negativtests mit „assertFalse“ und „assertNotEquals“ getestet, zusätzlich dazu prüft die Methode *testArrayIndexOutOfBoundsException**,*** ob eine ArrayIndexOutOfBoundsException geworfen wird
(2) ***TestsShipListCreator*****:** Testen, ob die Threads zur Erstellung der *shipLists* funktionieren (**Threads** → ***ShipListCreator***, ***shipListAICreator***)
Negativtests: „assertNotEquals“-Tests, dass die Liste nach dem Befüllen nicht leer ist
(3) ***TestShipPlacement*****:** Testbare Methoden getestet
Negativtests: „assertNotEquals“ zum Testen, ob richtig gezählt wurde

GUI (JavaFX)

Zahlreiche Events verwendet, darunter ActionEvent und MouseEvents:

(1) Gui → *ShipPlacementController* → *createFieldButtons*
(2) Gui → *BattlePhaseController* → *createFieldButtons*

Im Ordner **Gui** befinden sich die Controller für die verschiedenen Szenen.
Der Übergang ist folgendermaßen aufgebaut:

(1) Hauptmenü: Spiel starten, Spiel beenden.
(resources → fxml → M*enu.fxml)*
(2) Vorbereitungsphase: Setzen der Schiffe
(resources → fxml → *ShipPlacement.fxml)*
(3) Angriffsphase: Angreifen des gegnerischen Feldes
(resources → fxml → *BattlePhase.fxml)*
=> Nach dem Ende des Spiels kann entschieden werden in das Menü zurückzukehren oder die Applikation zu schließen.

Logging/Exceptions

In allen Klassen, wo Logging sinnvoll ist, wurde das Logging-Framework importiert.

Allgemeines LogLevel: *TRACE*
(in **resources → *log4j2.xml*** festgelegt)

Nutzung von Log-Leveln:

- TRACE: Detailliertes Logging der Erstellung einzelner Labels/Buttons und Einfügen in playerGrid bzw. enemyGrid (**Gui → *ShipPlacementController*/ *BattlePhaseController*** → *fillLabels, createFieldButtons*)

- DEBUG: Allgemeines Logging von Variablen, Methoden und Threads

- INFO: Logging des Programmablaufes, u.a. Start und Beenden der Applikation, Szenenwechsel etc. (**Gui** → ***GuiDriver /*** Controller-Klassen)

- ERROR: Exceptions, die abgefangen werden, jedoch nicht zum Programmabbruch führen; bei uns vor allem ArrayIndexOutOfBoundsException (**Gui** → ***ShipPlacementController*** → *showShip*, *placeShip*, *placeShipAI*)

- FATAL: Unsere eigene Exception *IllegalFactoryArgument (**Exceptions*** → *IllegalFactoryArgument)* wird in den aufrufenden Threads zur Erstellung der shipLists geloggt:
Threads → ***ShipListCreator*** bzw. ***ShipListCreatorAI***
Daraufhin wird das Programm abgebrochen.

Die Logs werden auf der Konsole ausgegeben und zusätzlich im File **A1.log** gespeichert.

UML

(1) Use-Case: Grundlegende Nutzer-Aktionen (Spieler, Computer) in der Anwendung werden im Use-Case-Diagramm dargestellt und in Beziehung gesetzt
(2) Klassendiagramm: Im Use-Case-Diaramm dargestellte Nutzer-Aktionen auf Klassen, Interfaces und Exceptions abgebildet, die durch Variablen und Methoden spezifiziert wurden.

Threads

Zum Erstellen der Schiffstypen und dem Befüllen von *ShipList* und *ShipListAI*, haben wir jeweils ein Runnable-Objekt erstellt:

Threads → ***ShipListCreator***, ***ShipListAICreator***

Diese übergeben wir anschließend zwei Threads:

Gui → ***ShipPlacementController*** → *initialize*

die wir anschließend starten mit „ .start() “

Die Threads greifen dabei gleichzeitig auf die Methode *createShip* (**Ships → *ShipFactory***) zu, welche wir daher mit synchronized geschützt haben.

Streams und Lambda-Funktionen

- Lambda-Funktionen:
Eventhandling für Buttons (ActionEvents/ MouseEvents):
Gui → *(ShipPlacementController / BattlePhaseController)* → *createFieldButtons*

- Streams und Lambda-Funktionen:

(1) Filtern von Buttons aus allen Nodes und Styleanpassungen, je nach Nutzer-Aktionen:
Gui → *BattlePhaseController* → *colorAIShots*
Gui → *ShipPlacementController* → *colorPlacedShips bzw. colorCellsIndividually*

(2) Durchsuchen und zählen von redundanten Schiffstypen in *shipList*:
Gui → *ShipPlacementController* → *countShipsInList*

(3) ParallelStream zum parallelisierten und damit performanteren Durchsuchen aller Buttons von *playerGrid* und Stylen je nachdem wo in der vorangegangenen Szene die Schiffe platziert wurden:
Gui → *BattlePhaseController* → *colorPlacedShips*

Factories

Ships → *ShipFactory*
Erstellung von neuen Objekten der jeweiligen Schiffstypen, die das Interface IShip implementieren:
Ships →
Battleship, Cruiser, Destroyer, Submarine