**Verfahren und Werkzeuge moderner Softwareentwicklung Wintersemester 2020/2021  
Aufgabe 4: Clean Code Development**

Mareike Röncke (Matrikelnr. 516497)

# Spiel: „Battleships“

Das Konsolenspiel „Battleships“ ist an das bekannte „Schiffe versenken“ angelehnt. Ich habe es in Java programmiert. Der Spieler spielt gegen den Computer, der zufällig „Schiffe“ in Form von Schrift-Symbolen auf einem variablen Spielfeld verteilt. Der Spieler gibt Positionen ein und erhält ein „Leben“, wenn ein Schiff getroffen wurde. Er verliert, wenn alle Leben aufgebraucht sind und er nicht alle Schiffe gefunden hat.

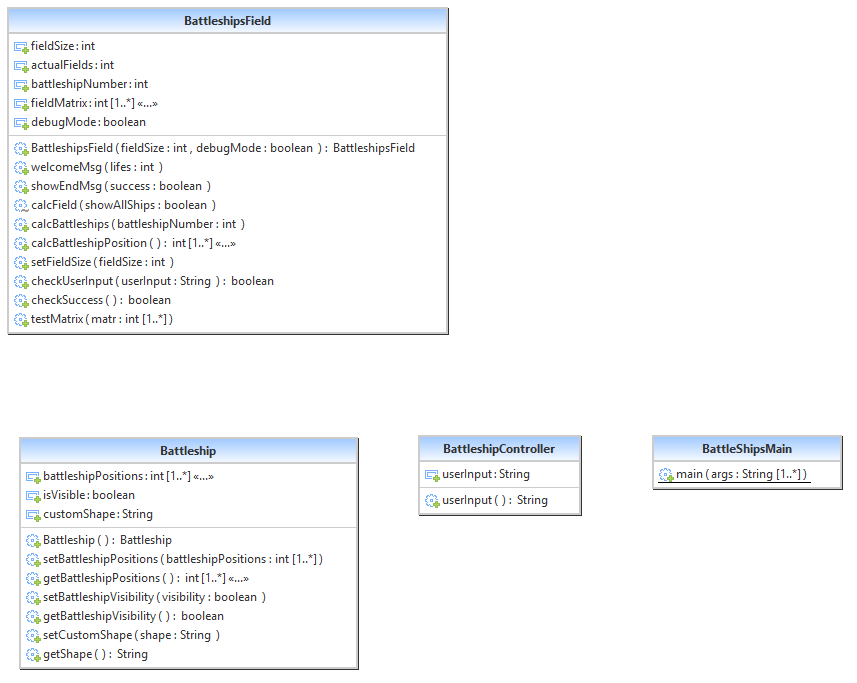
|  |  |
| --- | --- |
|  |  |
|  |  |

# Refactoring

## Pakete und Klassen vor dem Refactoring

|  |  |
| --- | --- |
| Der ursprüngliche Code enthält keine Pakete, alle Klassen befinden sich im automatischen „default package“.  Es ist unklar, was die einzelnen Klassen tun bzw. es gibt keine klare Aufgabenteilung. |  |

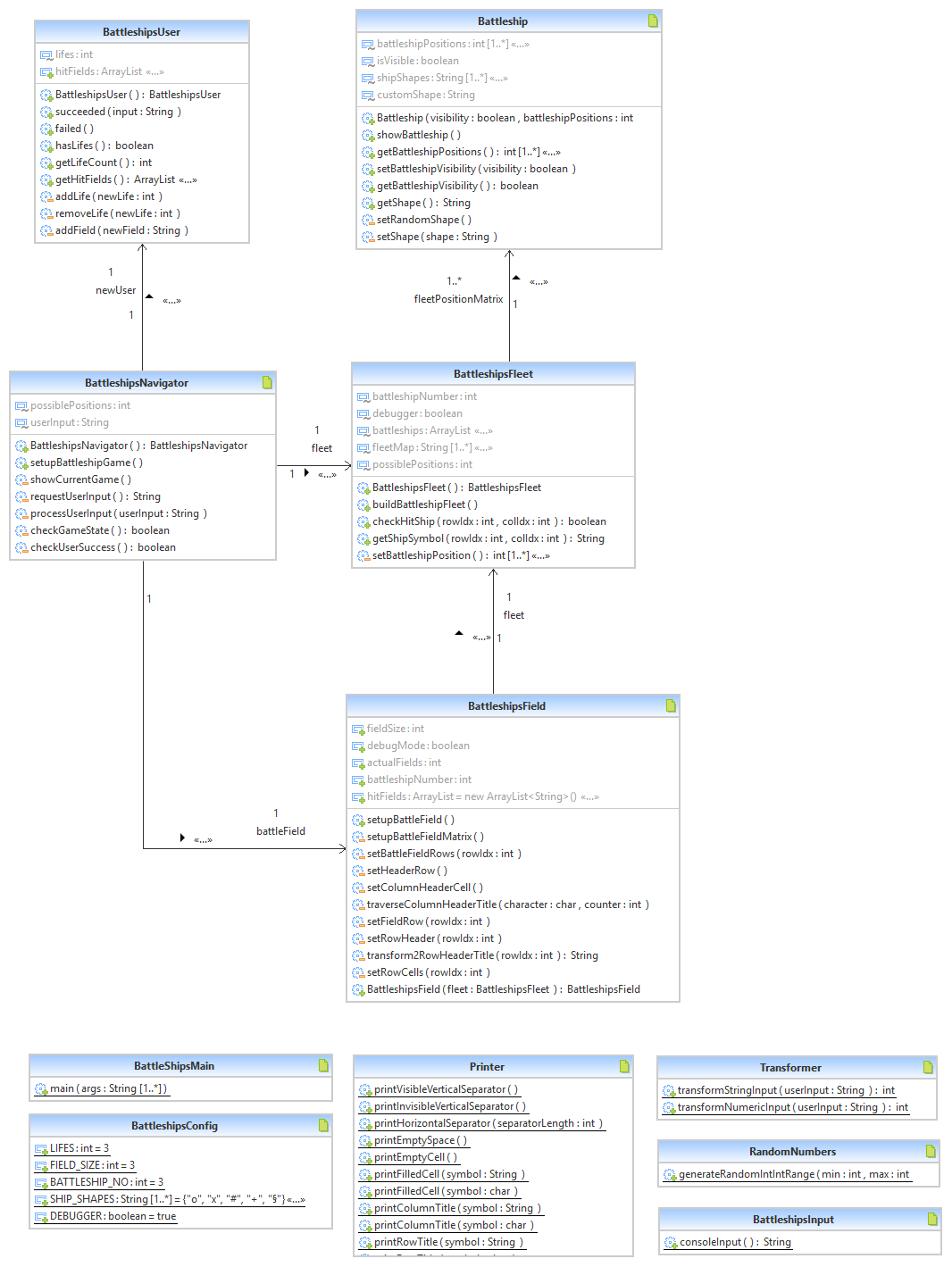
Im Klassendiagramm ist zu erkennen, dass sich ein Großteil der Logik des Spiels in der Klasse „BattleshipsField“ befindet. Es ist nicht zu erkennen, wie die Klassen einander aufrufen.



## Pakete und Klassen nach dem Refactoring

|  |  |
| --- | --- |
|  | Durch das Refactoring ist eine aufgeräumte MVC-Architektur entstanden.  Diese ist erkennbar durch die Pakete „battleship.controller“, „battleship.model“ und „battleship.view“. |
|  | Das Paket „battleship“ enthält die Main-Methode und eine Config-Datei. |
|  | Durch die Verwendung einer Config-Datei wird zum einen die Verwendung von „Magic Numbers“ im Code vermieden, und zum anderen macht es den Code erweiterbar (z.B. könnten die Config-Parameter über eine XML-Dateo eingelesen werden). |

Das UML-Diagramm macht deutlich, dass nun die Verantwortlichkeiten klar getrennt sind und der Kontrollfluss deutlich erkennbar ist. Auch ohne den konkreten Code zu sehen, lässt sich aus den Benennungen von Klassen und Methoden lesen, wie das Spiel programmiert ist.



## Methoden vor dem Refactoring

