**Javaprojekt-Dokumentation „TicTacToe“**



| **Institution** | Bénédict |
| --- | --- |
| **Schülerin** | Fabienne Kägi |
| **Abgabedatum** | Ende Mai |

Inhaltsverzeichnis

[1 Summary (Kurzfassung) 3](#_Toc514222943)

[2 Einleitung 4](#_Toc514222944)

[3 Aufgabenbeschrieb 4](#_Toc514222945)

[3.1 Spielregeln 4](#_Toc514222946)

[3.2 Pflichtenheft 4](#_Toc514222947)

[3.3 UML (Unified Modeling Language) 5](#_Toc514222948)

[3.3.1 Use-Case Diagramm 5](#_Toc514222949)

[4 Analyse (WAS) 6](#_Toc514222950)

[5 Design (WIE) 6](#_Toc514222951)

[6 Implementation 7](#_Toc514222952)

[7 Testszenario 7](#_Toc514222953)

[8 Zusammenfassung 8](#_Toc514222954)

[9 Reflexion 9](#_Toc514222955)

[9.1 Referenzverzeichnis 9](#_Toc514222956)

# Summary (Kurzfassung)

Ausgangslage:

* Für das Modul 226A Klassenbasiert (ohne Vererbung) implementieren, schreibe ich ein Programm in Java inklusive dieser dazugehörigen Dokumentation.
* Gegeben war:
  + Aufbau der Dokumentation.
    - Pflichtenheft
    - Analyse
    - Zusammenfassung
    - Erkenntnisse
    - Etc.
  + Anforderungen an den Code
    - Fehlerhandlung
    - Strukturierter Code
    - Kapselungen
    - Konsolenausgabe

Vorgehensweise/Lösungsfindung:

* Teilschritte:
  + Recherche welche Möglichkeiten vorhanden sind für die Umsetzung der Anforderungen
  + Grobkonzept: Konsolenausgabe des Spieles Tic Tac Toe.
    - Zwei Spieler spielen gegeneinander.
    - Sie sind immer abwechselnd am Zug
    - Auswertung des Siegers.

Erkenntnisse:

* In der Theorie erscheint vieles einfacher als in der Praxis.
* Theorie-Gap
  + Fehlerbehandlung
    - Verschiedene Möglichkeiten falsche Eingabe zu «korriegieren». (enum/try-catch)
  + Sichtbarkeit/Kapselung/Methodenstruktur/Lose Kopplung
    - Nicht in jeder Methode soll jeder Wert verändern werden können.
    - Kapselungen ersparen Schreibarbeit und machen den Code übersichtlicher.
    - Bei losen Koppelungen lassen sich Änderungen oft einfacher durchführen.

Aussicht:

* Erlerntes werde ich 1:1 im Geschäftsumfeld einsetzen können.

# Einleitung

Die vorliegende Dokumentation beschreibt den Aufbau und den Ablauf meiner Aufgabe ein kleines Programm selber zu schreiben.

# Aufgabenbeschrieb

Die Aufgabe ist es in Java ein kleines Script und eine Dokumentation zu schreiben.

Ich werde das bekannte Spiel Tic Tac Toe für diese Aufgabe verwenden.

## Spielregeln

Das Spielfeld besitzt neun Felder. Auf diesen Feldern setzt jeder Spieler abwechselnd sein Zeichen (Spieler A ein X und Spieler B ein O).

Gewonnen hat der, dessen Zeichen dreimal neben einander in einer Linie vorkommen, ob jetzt Waagrecht, Senkrecht oder Diagonal.

Erreicht keiner diese Linie ist es ein Unentschieden.

## Pflichtenheft

Im Pflichtenheft stehen die gewünschten Anforderungen an das Programm. Sie sind eingeteilt in «muss» und «kann», was für die Priorität der einzelnen Anforderungen spricht.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID-NR | Name | Beschreibung | Muss/kann | Erfüllt |
| 1 | Spieler | Einzelspieler, den Computer als Gegner | kann |  |
| 2 | Spielfläche | 9 Felder in einem Quadrat dargestellt | muss |  |
| 3 | Zeichen | Die Zeichen X und O müssen abwechselnd erscheinen | muss |  |
| 4 | Sieg/Unentschieden | Bei drei gleichen Zeichen Diagonal, Waagrecht oder Senkrecht ist es ein Sieg. | muss |  |
| 5 | Gegenspieler | Es spielen zwei Spieler gegeneinander | muss |  |
| 6 | Auswahl | Am Anfang auswählen ob man Einzelspieler oder Mehrspieler will | kann |  |
| 7 | Felder | Jedes Feld kann nur einmal gewählt werden | muss |  |
| 8 | Fehlerhandlung | Kontrolliert alle Eingaben und informiert wen sie nicht zugelassen sind | muss |  |
| 9 | Wiederholen | Am Ende des Spiels soll gefragt werden, ob man noch eine Runde möchte oder das Spiel beenden will. | kann |  |
| 10 | Einleitung | Spieleinleitung mit Regeln | kann |  |

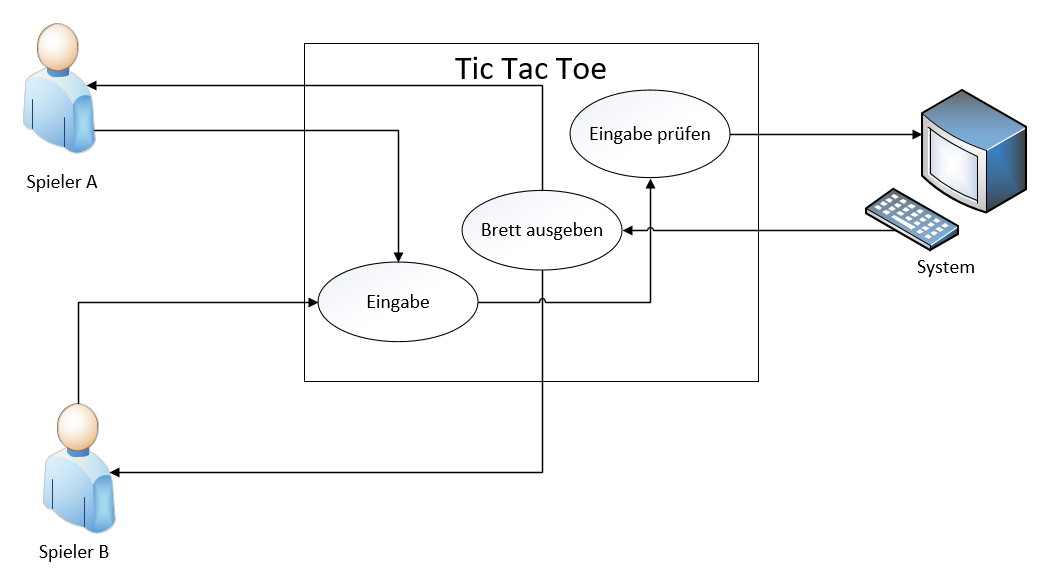
## UML (Unified Modeling Language)

Im UML ist in der ersten Spalte der Name, in der Zweiten folgen die Variablen/Attribute und in der letzten Spalte die Methoden.

|  |
| --- |
| Runner |
| board: String  UserInput: String  value: String  checkWinner: boolean  numValue: int  turn: int |
| + main (String [])  - isExitOfTheGabme (int)  - initBoard ()  - printBoard (String [])  - readUserInput ()  - checkWinner (String [], Player) |

### Use-Case Diagramm

Dieses Use-Case Diagramm stellt den Spielablauf dar. Nach einander setzten die Spieler abwechselnd ihre Zeichen auf ein freies Feld:



# Analyse (WAS)

2. Spielfläche: Für jedes Feld soll eine Zahl als Lokation dienen.

3. Zeichen: Um erkennen zu können welches Feld welcher Spieler gewählt hat, müssen abwechselnd die Zeichen X und O erscheinen.

4. Sieg/Unentschieden: Nach jeder Runde muss überprüft werden, ob ein Spieler schon gewonnen hat. Erreicht keiner den Sieg soll das Spiel automatisch enden, wenn kein Feld mehr frei ist und ein Unentschieden ausgeben.

5. Gegenspieler: Ist eine zweite Person, welche abwechselnd mit dem ersten Spieler seine Zeichen setzt.

7. Felder: Ein belegtes Feld kann nicht nochmal ausgewählt werden, da dies die Regeln nicht erlaubt.

8. Fehlerhandlung: Das ungültige Eingaben nicht einfach zu einem Abbruch führen, soll mit Hilfe einer Fehlerhandlung der Spieler kontrolliert und informiert werden.

10. Einleitung: Am Anfang sollen die Regeln des Spieles und das Beenden deklariert werden.

# Design (WIE)

2. Spielfläche: Mit Hilfe von Zahlen welche der Reihe nach in einem Array gespeichert sind, wird jedes «Feld» mit einer Lokation von 1-9 deklariert, welche nach Auswahl des Feldes durch das jeweilige Zeichen ersetzt wird. Damit das Spielfeld etwas übersichtlicher ist, wird pro Linie drei Positionen vom Array ausgegeben. Mit einem Strich als Trennung zum nächsten Array-Wert werden die einzelnen Felder noch etwas übersichtlicher.

Spielfeld:

-------------

| 1 | 2 | 3 |

-------------

| 4 | 5 | 6 |

-------------

| 7 | 8 | 9 |

-------------

3. Zeichen: Die Zeichen werden mit einer If-Anweisung und dem Rundenzähler in der Schlaufenbestimmung abwechselnd ausgegeben.

-------------

| **X** | 2 | 3 |

-------------

| 4 | **O** | 6 |

-------------

| 7 | 8 | 9 |

-------------

Spieler 1 hat auf 1 gesetzt.

Spieler 2 hat auf 5 gesetzt.

4. Sieg/Unentschieden: In der Kontrollstruktur werden alle möglichen Reihen und Diagonalen überprüft, welche für einen Sieg in Frage kommen. Stehen drei Zeichen in diesen Reihen/Diagonalen wird der Spieler der gewonnen hat ausgegeben. Erreicht dies keiner der Spieler wird durch den Rundenzähler nach neun Runden das Spiel als Unentschieden beendet.

5. Gegenspieler: Dadurch das die Zeichen abwechselnd erscheinen sind es schon «zwei» Spieler. In der gleichen Schleife wie den Zeichen-Wechsel, habe ich programmiert das gezeigt wird welcher Spieler gerade den Zug ausgeführt hat.

7. Felder: Da ein Feld nur einmal besetzt werden darf, wird mit einer Kontrollstruktur das jeweilige gewählt Feld kontrolliert und nur ausgegeben wen es noch frei ist.

Ist es jedoch schon belegt, wird eine Information ausgegeben und der gleiche Spieler kann sich ein anderes Feld aussuchen.

8. Fehlerhandlung: Bei Fehlerhandlung wird überprüft ob die Eingabe erlaubt ist oder nicht. Es ist nur möglich eine Zahl von 1-9 einzugeben, bei jeder weiteren Zahl und auch bei Buchstaben oder Wörtern soll eine Warnmeldung erscheinen und der Spieler kann erneut eine Eingabe tätigen. Diese Fehlerhandlungen sind zum einen in «enum» deklariert, oder wie bei den Buchstaben hilft die Exception-Funktion «try/catch».

10. Einleitung: Mit einem einfachen «System.*out*.println() » wird eine Begrüssung, die Regeln und der Beendungs-Befehl (99) aufgeführt.

# Implementation

Zu Beginn habe ich die Programmstruktur aufgebaut. Schon beim Einteilen welche Strukturen in die While-Schleife kommen und welche nicht, kam es ein wenig ins Stocken. Nach diesem Arbeitsschritt wurden die einzelnen Punkte wie Spielbrett, Eingabe, Ausgaben und weiteres geschrieben. Schwierigkeiten gab es da bei der Fehlerhandhabung, wegen den erlaubten und unerlaubten Eingaben. Der Aufbau der Kontrollstruktur des Sieges war relativ simpel, einfach etwas mehr Schreibarbeit.

Die verschiedenen Aufgaben wurden in einzelne Methoden unterteilt.

# Testszenario

|  |  |  |
| --- | --- | --- |
| Testfall 1 |  | Doppelte Feldauswahl: Wen ein Feld nochmals von einem Spieler ausgewählt wird, erfolgt eine Meldung das dieses Feld bereits besetzt ist.  Im Beispiel ist das Feld 5 schon vom Spieler A belegt. |
| Testfall 2 |  | Ungültige Eingabe: Auch wenn eine Eingabe ausserhalb der Zahlen 1 bis 9 oder ein Buchstabe/ ein Wort eingegeben wird, erfolgt eine Warnmeldung das die Eingabe ungültig ist. |
| Testfall 3 |  | Siegerausgabe: Wen ein Spieler seine Zeichen in einer Reihe erreicht hat, wird dieser als Gewinner Ausgegeben.  Im Beispiel hat Spieler A seine drei Zeichen in der Diagonalen erreicht |
| Testfall 4 |  | Spiel beenden: Mit der Eingabe der Zahl 99 wird das Spiel jederzeit beendet. |

# Zusammenfassung

Ich habe das Spiel Tic Tac Toe programmiert.

Als erster Schritt habe ich das Spielfeld aufgebaut. Daraufhin wurden Schritt für Schritt die Eingaben und Ausgaben eingebaut.

Das komplette Programm läuft dank einer While-Schleife, solange bis ein Gewinner feststeht, oder alle Felder belegt sind.

Um das Spiel jeder Zeit beenden zu könne ist die While-Schleife mit «true» deklariert und bei der Eingabe des EXIT\_CODE «99» wird das Spiel abgebrochen, da sich die Bedingung in der Schleife ins Negative ändert (false).

Alle Gewinnmöglichkeiten werden mit Hilfe einer If-Anweisung überprüft und der jeweilige Sieger ausgegeben.

Ebenfalls mit einer If-Anweisung wird auch berechnet welcher Spieler an der Reihe ist.

Die erlaubten Eingaben sind in einem «enum» hinterlegt und werden in einer If-Anweisung geprüft. Um ein übersichtlichen Code zu erhalten, wurden die einzelnen Anwendungen in Methoden deklariert.

# Reflexion

|  |  |
| --- | --- |
| Frage | Antwort |
| Wurde das Projektziel erreicht? | Teils. Es ist ein lauffähiger Code, jedoch nicht mit allen Details die ich mir erwünscht habe. |
| Was trug zum Erreichen des Zieles bei? | Gute Unterstützung von Fachleuten, welche mir immer wieder neue Anregungen anboten.  Auch Recherchen im Internet bezüglich Schlaufen zum Ergebnis beigetragen. |
| Welche Schwierigkeiten gab es? | Schwierigkeiten gab es unter anderem bei den Einteilungen und der Reihen folge, wie auch bei den einbauen teils If-Schlaufen.  Die Reihenfolge der Anforderungen, welche ich schreiben wollte, in sinnvoller Folge abzuarbeiten. |
| Wie könnte das Projekt weitergeführt werden? | Es könnt zusätzlich einprogrammiert werden das man auch dem Computer als Gegner wählen kann. Und das man nach einem fertigen Spiel die Wahl hat noch eine Runde zu spielen oder zu Beenden. |
| Was habe ich gelernt? | Ich habe gelernt wie ich am besten beim Aufbau eines neuen Codes vorgehe. Auch habe ich viel über die Java-Kommandos und die Aufbauart gelernt. Und wie man einzelne Aufgaben zu einem komplettem Script zusammen fügen. |
| Was könnte ich beim nächsten Mal besser machen? | Das Einteilen der Reihenfolge vom Codieren der Anforderungen sollte am Anfang besser aufgelistet sein, um Punkt für Punkt abzuarbeiten. |

## Referenzverzeichnis

Instructables, <http://www.instructables.com/id/How-to-Write-a-Tic-Tac-Toe-Program-in-Java/>

Stackoverflow, <https://stackoverflow.com/questions/18038533/using-while-loop-in-java>

<https://stackoverflow.com/questions/3978654/best-way-to-create-enum-of-strings>

GitHub, <https://gist.github.com/xaviablaza-zz/3844825>

Java ist auch eine Insel 5. Auflage, while-Schleife, Seite 133 & 134, Christian Ullenboom

Java ist auch eine Insel 5. Auflage, Exceptions, Seite 423 & 424, Christian Ullenboom