**Javaprojekt-Dokumentation „TicTacToe“**



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

Inhaltsverzeichnis

[1 Summary (Kurzfassung) 3](#_Toc514320819)

[2 Einleitung 4](#_Toc514320820)

[3 Aufgabenbeschrieb 4](#_Toc514320821)

[3.1 Spielregeln 4](#_Toc514320822)

[3.2 Pflichtenheft 4](#_Toc514320823)

[4 Analyse (WAS) 5](#_Toc514320824)

[5 Design (WIE) 5](#_Toc514320825)

[5.1 UML (Unified Modeling Language) 7](#_Toc514320826)

[5.1.1 Use-Case Diagram 7](#_Toc514320827)

[5.1.2 Klassendiagramm 7](#_Toc514320828)

[6 Implementation 8](#_Toc514320829)

[7 Testszenario 8](#_Toc514320830)

[8 Zusammenfassung 9](#_Toc514320831)

[9 Reflexion 9](#_Toc514320832)

[9.1 Referenzverzeichnis 10](#_Toc514320833)

# 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
    - Fehlerbehandlung
    - Strukturierter Code
    - Kapselungen
    - Konsolenausgabe

Vorgehensweise/Lösungsfindung:

* Teilschritte:
  + Recherche: Welche Möglichkeiten sind vorhanden 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 «korrigieren». (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. Auch bringen sie dem Code Sicherheit und Robustheit.
    - Bei losen Koppelungen lassen sich Änderungen einfacher durchführen.

Aussicht:

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

# Einleitung

Die vorliegende Dokumentation beschreibt den Aufbau und den Ablauf der Applikation.

# Aufgabenbeschrieb

Die Aufgabe ist es in Java ein Programm inklusiv Dokumentation zu schreiben.

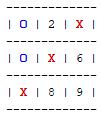
Ich werde das bekannte Spiel Tic Tac Toe für diese Aufgabe verwenden.

## 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 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 |
| P-1 | Spielfläche | 9 Felder in einem Quadrat dargestellt | muss |  |
| P-2 | Zeichen | Die Zeichen X und O müssen abwechselnd erscheinen | muss |  |
| P-3 | Felder | Jedes Feld kann nur einmal gewählt werden | muss |  |
| P-4 | Sieg/Unentschieden | Bei drei gleichen Zeichen Diagonal, Waagrecht oder Senkrecht ist es ein Sieg. | muss |  |
| P-5 | Gegenspieler | Es spielen zwei Spieler gegeneinander | muss |  |
| P-6 | Auswahl | Am Anfang auswählen ob man Einzelspieler oder Mehrspieler will | kann |  |
| P-7 | Spieler | Einzelspieler, den Computer als Gegner | kann |  |
| P-8 | Fehlerhandlung | Kontrolliert alle Eingaben und informiert wen sie nicht zugelassen sind | muss |  |
| P-9 | Spielabbruch | Das Spiel soll jeder Zeit beenden werden können. | muss |  |
| P-10 | Wiederholen | Am Ende des Spiels soll gefragt werden, ob man noch eine Runde möchte oder das Spiel beenden will. | kann |  |
| P-11 | Einleitung | Spieleinleitung mit Regeln | kann |  |
| P-12 | Farbe | Zur besseren Übersicht | kann |  |

# Analyse (WAS)

P-1. Spielfläche: Besteht aus neun Feldern.

P-2. Zeichen: Es werden die Zeichen «X» und «O» gesetzt.

P-3. Felder: Jedes Feld nur einmal wählbar.

P-4. Sieg/Unentschieden: Der Gewinner wird ausgewertet.

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

P-8. Fehlerhandlung: Ungültige Eingaben führen nicht zum Spielabbruch.

P-9. Spielabbruch: Das Spiel kann jederzeit beendet werden.

P-10. Einleitung: Kleine Einleitung zum Spiel.

P-11. Farbe: Hebt Einzelheiten farblich hervor.

# Design (WIE)

Referenziert auf das Pflichtenheft.

P-1. Spielfläche: Mit Hilfe von Zahlen, welche in einem String Array gespeichert sind, wird jedes «Feld» mit einer Lokation von 1-9 deklariert, welche nach Auswahl des Feldes durch das jeweilige Zeichen ersetzt wird. (String [] = [1][2][3][4][5][6][7][8][9])

Beachtet werden muss das ein Arrey immer von 0 beginnt, das heisst um den ersten Array mit dem Wert [1] zu bekommen, ist es notwendig den Array-Wert plus eins zu rechnen. Dies gilt auch bei der Feldeingabe, bei der eingegebenen Zahl wird programmiert das eins abgezogen wird um den richtigen Array-Wert zu ändern.

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 |

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

P-2. 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.

P-3. Felder: Da ein Feld nur einmal besetzt werden darf, wird mit einer Kontrollstruktur das jeweilige gewählt Feld kontrolliert und nur ausgegeben, wenn es noch frei ist.

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

P-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.

P-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.

P-8. Fehlerbehandlung: Bei Fehlerbehandlung 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».

P-9. Spielabbruch: Die while-Schleife ist mit «true» initialisiert. Tätigt ein Spieler die Eingabe 99 wird ein «false» zurückgegeben, was dazu führt, dass das Spiel beendet wird.

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

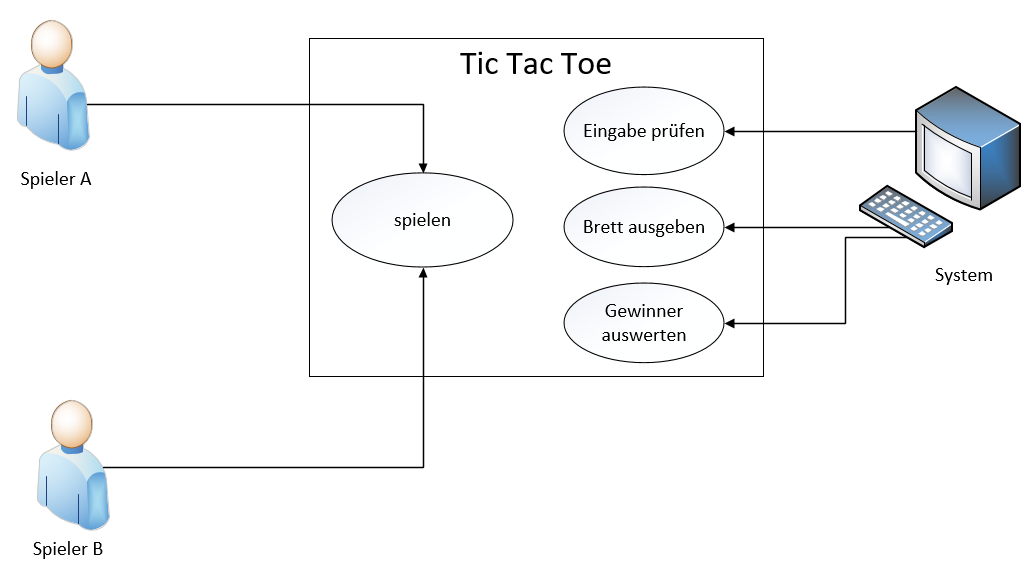
11. Farbe: Die Farben sind direkt unter der Klasse in einem String initialisiert. Vor jedem Zeichen/Wort, das Farbig sein soll, wird der gegebene Name für die gwünschte Farbe geschrieben (z.B. COLOR\_RED für Rot). Um die Farbgebung wieder aufzülösen ist das ende einzel initialisiert unter «COLOR\_CLEAN», nach diesem «Befehl» erfolgen die Zeichen/Wörter wieder wie Vorgegeben.

## UML (Unified Modeling Language)

Die UML ist eine grafische Modellierungssprache für die Konstruktion und Dokumentation von Applikationen und Softwarentwicklungen:

### Use-Case Diagram

Dieses Use-Case Diagramm stellt den Spielablauf dar. Nacheinander setzen die Spieler abwechselnd ihr Zeichen auf ein freies Feld:



### Klassendiagramm

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

# 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 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 ersten 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.

Mit der Eingabe «99» kann das Spiel jederzeit beendet werden.

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 |
| Wurden die Anforderungen im Pflichtenheft erreicht? | Alle Anforderungen, welche mit «muss» deklakiert sind, wurden erfüllt. Auch zwei der «kann» Anforderungen sind eingebaut worden. |
| 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 Programm 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