## Dokumentation

**Der Gruppe 1**

**Gruppenmitglieder:**

Levin Vollnhals (Gruppenleiter)

Mathias Wandrei

Marcel Reichelt

## Inhaltsverzeichnis

1. **Die Projektaufgabe**
   1. **Allgemein: Die 3-Schichtenarchitektur**
   2. **Allgemein: Unser Design-Pattern**
   3. **Unser Anwendungsproblem**
2. **Das Produkt**
   1. **Das Klassendiagramm**
   2. **Die Benutzungsoberflächen**
   3. **XML-Schema der dateibasierten Datenbank**
   4. **ER-Modell der SQL-Datenbank**
   5. **Realisierung des Design-Patterns**
   6. **Sequenzdiagramm der Kommunikation**
3. **Quellen**
4. **Die Projektaufgabe**
   1. **Allgemein: Die 3-Schichtenarchitektur**

Die 3-Schichtenarchitektur ist ein Modell, welches aus 3 Schichten besteht. Hierbei werden die Funktionen eines Programms in 3 Teile Aufgeteilt.

GUI-Schicht

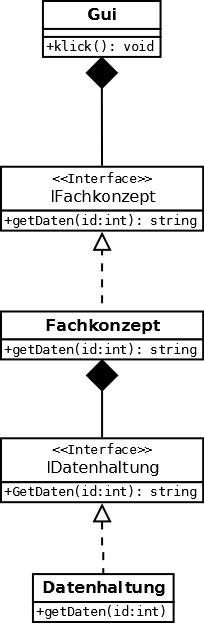
Fachkonzeptschicht

Datenhaltungsschicht

In der GUI-Schicht befindet sich alles was mit der Darstellung zu tun hat. Sowie die Dialogführung durch das Programm.

In der Fachkonzeptschicht befinden sich die verarbeitende Logik des Programms, sowie die Verbindung zur Datenhaltungsschicht.

Die Datenhaltungsschicht kümmert sich um das Abspeichern und Laden von Daten, z.B. in eine Datenbank.



* 1. **Allgemein: Unser Design-Pattern**

Wir haben uns für das Strategy-Pattern entschieden.

Hierbei handelt es sich um ein Verhaltensmuster, welche eine Familie von Algorithmen definiert

Dabei werden Klassen mit unterschiedlichem Verhalten, durch eine Schnittstelle angesprochen.

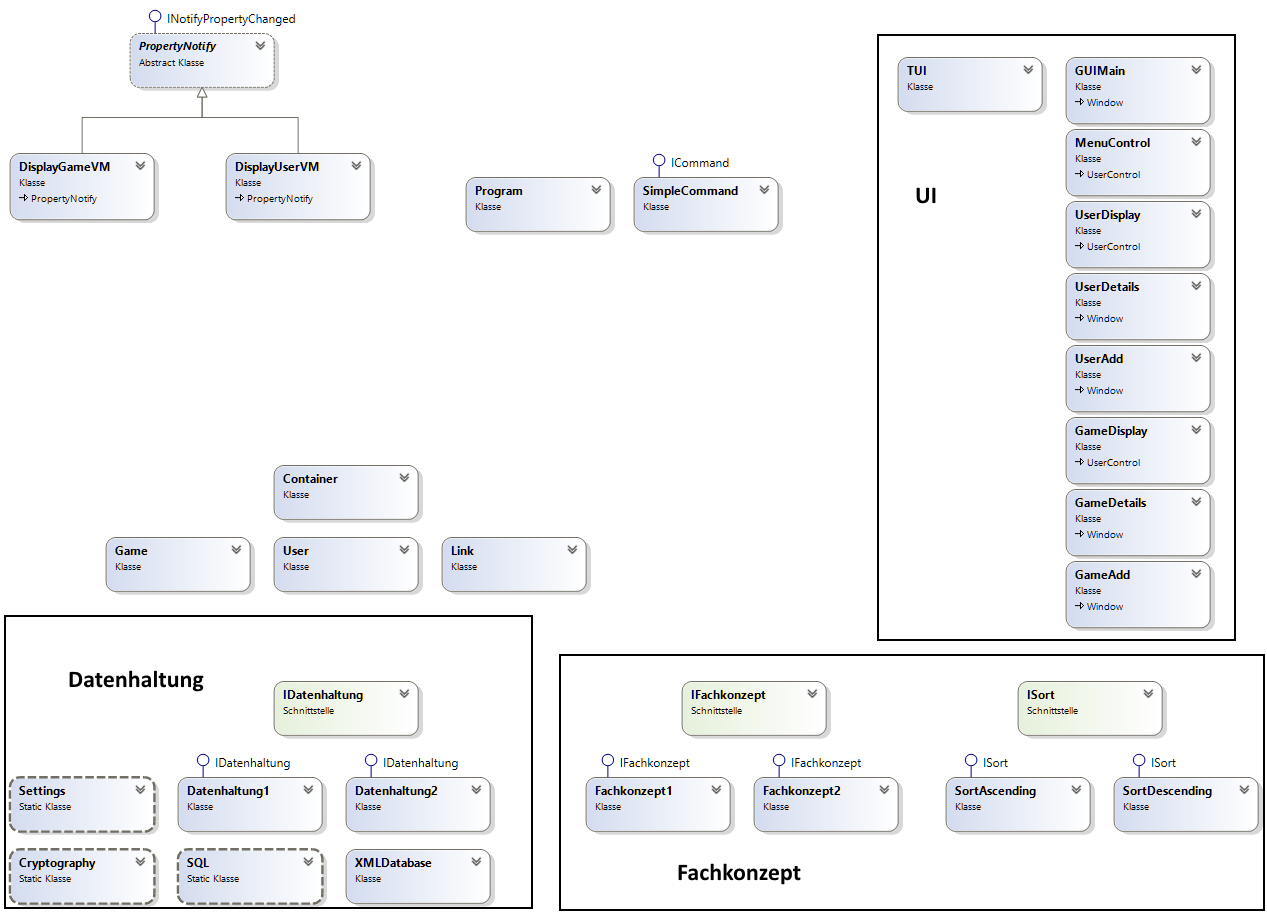
Dies erlaubt uns das verhalten zur Laufzeit zu verändern.

****

* 1. **Unser Anwendungsproblem**

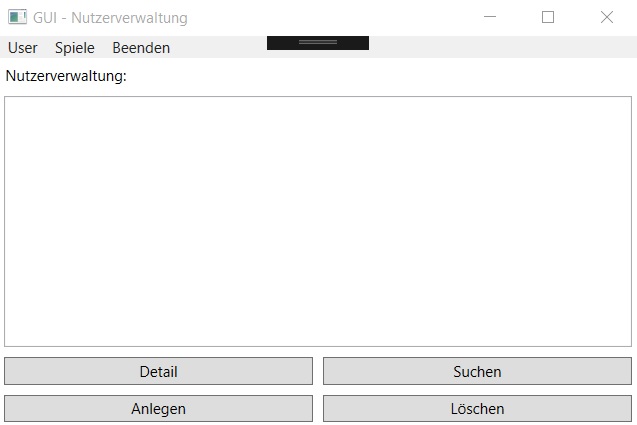
Die Firma Pustekuchen möchte eine Verwaltungssoftware Für ihre verkauften Spiele haben. Das Programm soll eine Verknüpfung zwischen Nutzern und Spielen herstellen.

1. **Das Produkt**
   1. **Das Klassendiagramm**

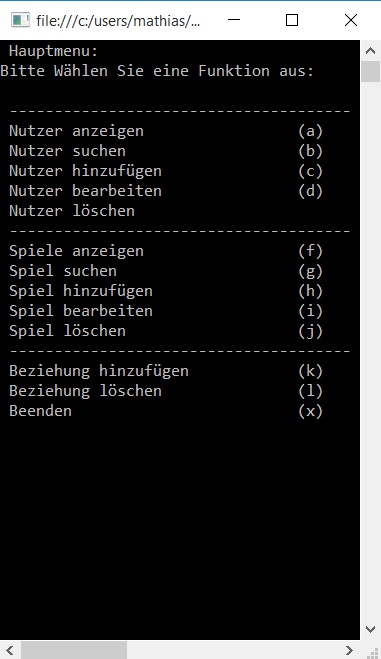
****

* 1. **Die Benutzungsoberflächen**

**GUI:**



**TUI:**



* 1. **XML-Schema der dateibasierten Datenbank**

**<?xml version="1.0" encoding="Windows-1252"?>**

**<xs:schema xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xs="http://www.w3.org/2001/XMLSchema" attributeFormDefault="unqualified" elementFormDefault="qualified">**

**<xsd:element name="XMLDatabase">**

**<xsd:complexType>**

**<xsd:sequence>**

**<xsd:element name="UserData">**

**<xsd:complexType>**

**<xsd:sequence>**

**<xsd:element maxOccurs="unbounded" name="User">**

**<xsd:complexType>**

**<xsd:sequence>**

**<xsd:element name="Firstname" type="xsd:string" />**

**<xsd:element name="Lastname" type="xsd:string" />**

**<xsd:element name="User\_id" type="xsd:unsignedByte" />**

**</xsd:sequence>**

**</xsd:complexType>**

**</xsd:element>**

**</xsd:sequence>**

**</xsd:complexType>**

**</xsd:element>**

**<xsd:element name="GameData">**

**<xsd:complexType>**

**<xsd:sequence>**

**<xsd:element name="Game">**

**<xsd:complexType>**

**<xsd:sequence>**

**<xsd:element name="Name" type="xsd:string" />**

**<xsd:element name="Game\_id" type="xsd:unsignedByte" />**

**</xsd:sequence>**

**</xsd:complexType>**

**</xsd:element>**

**</xsd:sequence>**

**</xsd:complexType>**

**</xsd:element>**

**<xsd:element name="Links">**

**<xsd:complexType>**

**<xsd:sequence>**

**<xsd:element name="Link">**

**<xsd:complexType>**

**<xsd:sequence>**

**<xsd:element name="User\_id" type="xsd:unsignedByte" />**

**<xsd:element name="Game\_id" type="xsd:unsignedByte" />**

**</xsd:sequence>**

**</xsd:complexType>**

**</xsd:element>**

**</xsd:sequence>**

**</xsd:complexType>**

**</xsd:element>**

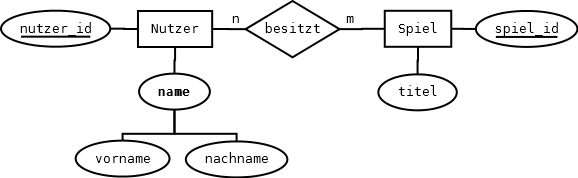
**</xsd:sequence>**

**</xsd:complexType>**

**</xsd:element>**

**</xs:schema>**

* 1. **ER-Modell der SQL-Datenbank**



* 1. **Realisierung des Design-Patterns**

Das Strategy-Pattern wurde benutzt um die Sortierreihenfolge zu ändern.

Wir haben einen Controller der Steuert welcher Algorithmus benutzt wird, sowie 2 verschiedenen Algorithmen

Sorter SortHandler = new Sorter();

new SortAscending()

new SortDescending()

die Auf- und Absteigend sortieren.

Durch den Aufruf der Funktion:

public Container sort(Container unsorted)

{

return strategy.sort(unsorted);

}

Wird der übergebene Container sortiert, in Abhängigkeit davon welche Strategie der Controller im Moment verfolgt.

Das Verhalten kann durch den Aufruf einer einzelnen Funktion geändert werden.

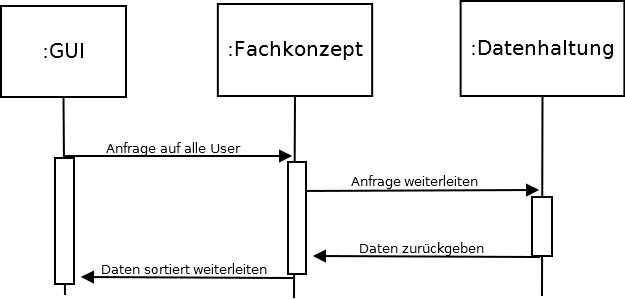
public void changeSortOrder()

{

sortDescending = !sortDescending;

}

* 1. **Sequenzdiagramm der Kommunikation**



1. **Quellen**

<http://www.oop-uml.de/drei-schichten-architektur.php>

<https://en.wikipedia.org/wiki/Strategy_pattern>