Dokumentation Notenverwaltung

Technischer Prototyp

Autor: Karl Klotz

Datum: 2. Juni 2023

# Aufgabenstellung

Erstellung eines Objektrelationalen Mappers (ORM) in C#, welcher genutzt werden kann, um eine Datenbank für eine Notenverwaltung zu verwalten.

Der ORM soll die Möglichkeit bieten die zuerst anzubindende SQLite-Datenbank durch eine andere Datenbank auszutauschen, zum Beispiel MySQL.

Die Notenverwaltung soll es Lehrern und Schülern ermöglichen Noteneinzusehen, Noten zu vergeben, Schülern Klassen, Bildungsgängen und Kursen zuzuordnen.

Optional ist ein Frontend, welches die Notenverwaltung steuern kann. Eine rudimentäre Konsolenausgabe, welche die Funktionalität des ORM demonstrieren, ist ausreichend.

Abgesehen von Standard-Bibliotheken dürfen keine weiteren Bibliotheken oder Frameworks eingesetzt werden.

# Prototyp

Der technische Prototyp befindet sich im Ordner „Release“, die Datei „Notenverwaltung.exe“ muss ausgeführt werden.

Es werden nacheinander verschiedene Objekte von Personen, Schülern, Lehrern, Bildungsgängen, Kursen und Noten erstellt, und deren Inhalte werden direkt auf der Konsole ausgegeben.

Es wurde auf jegliche Menüführung oder die Möglichkeit individuelle Aktionen auszuführen verzichtet. (*Hinweis: Dies geschah in Absprache mit dem projektgebenden Lehrer.)*

# Umsetzung

## Dependency Injection

Der Entwickler entschied sich, die Datenbank-Klasse „DatabaseStorage“ und „DatabaseBuilder“ als Interfaces zu implementieren.

Mithilfe der Technik der Dependency Injection, wird ein einziges Objekt aus einer Klasse erstellt, welche das Interface DatabaseStorage realisiert. Dieses einzelne Objekt wird mit der vom Entwickler implementierten Klasse „GlobalObjects“ für alle anderen Klassen im Projekt zugänglich gemacht.

Um sicherzustellen, dass jedes globale zu injizierende Objekt einmalig vorkommt, hat der Entwickler die Klasse „InterfaceListing“ genutzt, in welcher globale Objekte vermerkt werden. Es ist somit nicht mehr möglich mehr als ein Objekt eines bestimmten Interfaces im Programm zu instanziieren.

(*Hinweis: Die Verwendung von Dependency Injection geschah in Absprache mit dem projektgebenden Lehrer.)*

### Vorteile von Dependency Injection

Austauschbarkeit der Datenbank-Anbindung

Der Entwickler kann beim Instanziieren Klasse, welche „DatabaseStorage“-Interface realisiert, entscheiden, welche konkrete Realisierung genutzt wird. Hier ist es nun leicht möglich, zwischen den Klasse „SQLiteDatabaseStorage“ und anderen Realisierungen des Interface zu wechseln.

Vereinheitlichung des Zugriffs und Kapselung

Das Verwenden des „DatabaseStorage“ Interface schafft eine klare Schnittstelle, über welche die restlichen Programmteile mit der Datenbank kommunizieren können.

Verminderung komplizierter Objektübergaben zwischen Klassen

Die Möglichkeit die „DatabaseStorage“-Instanz in jeder beliebigen Klasse zu injizieren, ermöglicht es komplizierte Objektübergaben zu vermeiden. Es ist global verfügbar.

Gute Möglichkeiten modular und parallel mit mehreren Entwicklern zu arbeiten

Dependency Injection ermöglicht vor allem bei simultaner Arbeit innerhalb eines Projekts gute Bedingungen: Die Verwendung von Interfaces ermöglicht es komplette Klassen erneut zu implementieren, und bis diese fertig sind, weiterhin die alten zu verwenden. Bei Fertigstellung kann die alte Klasse einfach mit der neuen Klasse ausgetauscht werden, ohne das andere Entwickler dies bemerken.

### Der Entwickler wollte unbedingt lernen die Technik der Dependency Injection selbst, manuell, und ohne Frameworks zu verwirklichen, um sein Wissen zu erweitern. Der Entwickler beabsichtigte mit der Verwendung seine Fähigkeiten im Umgang mit Dependency Injection weiter zu verbessern.

## DatabaseWrapper, Model und Entity

Die abstrakte Klasse „Entity“ wird im DatabaseWrapper-Package implementiert. Objekte die diese Klasse realisieren, können von der Datenbank als Relationen gespeichert, abgerufen, aktualisiert und gelöscht werden.

Die Entity-realisierenden Objekte sind im Package „Model“ im Projekt zu finden. Hier finden sich also die Schüler-, Lehrer-, Noten-, und Bildungsgänge-Klassen (uvm.), mit welchen man im restlichen Programm arbeiten kann.

## Probleme während der Implementierung

### Zirkuläre Referenzierung und Endlosschleife

Beispiel: Ein Schüler besucht viele Kurse, und jeder Kurs hat viele Schüler. Bei jeder Erstellung und Aktualisierung von Schülern oder Kursen, möchte der Entwickler für beide Objekte Listen der besuchten Kurse und der eingetragenen Schüler aktualisieren.

Die Aktualisierung eines Schülers soll also alle dessen besuchten Kurse in der Datenbank abfragen, und diese Liste als Attribut in diesen Schüler verfügbar machen. Nun werden aber bei einer Abfrage der Kurse diese auch aktualisiert, es wird abgefragt welche Schüler in einem Kurs sind, und die eingetragenen Schüler in der Liste eines Kurses aktualisiert. Das Erstellen des Schüler-Objekts in einem Kurs benötigt aber erneut die Abfrage aller seiner Kurse.

Lösung: Es wurde die Variable „recursionLevel“ genutzt, welche bei internen Aktualisierungsabfragen zwischen Entitäts-Objekten und der DatabaseStorage erhöht wird. Falls die Rekursionstiefe größergleich 3 ist, wird von weiteren Hinzufügen von Kurs-Objekten die in Beziehung zum Schüler-Objekt sein sollen abgesehen.

### Nicht gelöschtes Reader-Objekt von Sqlite verhindert das die Datenbankverbindung geschlossen wird

Innerhalb der Klasse SqliteDatabaseStorage wird die Sqlite Datenbank mithilfe eines Objekts der Klasse SqliteConnection angesprochen. Dieses SqliteConnection Objekt nutzt ein Objekt der Klasse SqliteDataReader um die Ergebnisse einer Datenbankabfrage bereit zu stellen. Falls es während der Verarbeitung der Ergebnisse zu einer Exception kommt, kann es aber in manchen Fällen passieren, dass das SqliteDataReader-Objekt nicht gelöscht wird. Die weiterhin bestehende Referenz auf den Reader verhindert, dass die SqliteConnection geschlossen wird. Nach nur wenigen Abfragen blockiert die Sqlite-Datenbank, und gab die Fehlermeldung zurück, dass es zu viele simultane Abfragen gäbe.

Die Lösung hierfür war, dass für jedes der genannten Objekte ein neuer using-Block deklariert wurde, welcher sicher stellt, dass auch bei Fehlern diese Objekte dann zuverlässig wieder gelöscht werden.

## AttributToValueDescription beschreibt auch Relationen

Die Klasse wird von Entity-realisierenden Klassen genutzt, um eine Beschreibung des Objekts zu erstellen, wie es als Relation in der Datenbank existiert. Desweiteren speichert die Klasse die Attribute, und transferiert sie zwischen den DatabaseWrapper und Model-Bereich des Programms.

Auch Relationen werden in diesen Objekt beschrieben.

Attribute sind mithilfe von der Klasse KeyValue beschrieben.

Ein-zu-Eins- und Eins-zu-Viele Beziehungen werden mithilfe von Objekten der Klasse OneToXRelationKeyValue beschrieben.

Viele-Zu-Viele-Beziehungen nutzen hierfür die Klasse ManyToManyKeyValue.

## Verzicht auf die Umsetzung von einigen Relationen

Der Entwickler entschied, dass für den technischen Prototyp die folgenden im ERD beschriebenen Relationen nicht im Projekt verwirklicht werden:

* BildungsgangHatFach
* NotenTyp

Der Hauptgrund war, dass diese Relationen die geringste Priorität haben, um zu demonstrieren, dass der ORM mit der Datenbank erfolgreich arbeitet.