# DOKUMENTATION PROGRAMMENTWURF

Advanced Software-Engineering

Nico Rahm

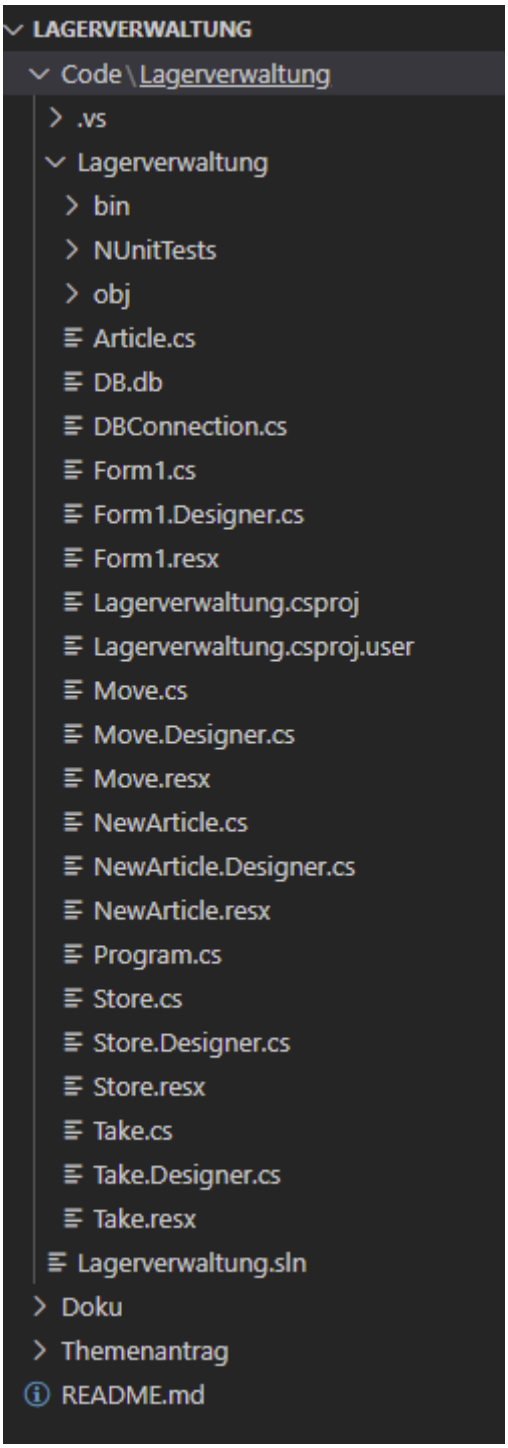
# Inhalt



Link zum Repository: https://github.com/Nico-Rahm/Lagerverwaltung

# Allgemeines

## Ordnerstruktur




Dieses Dokument ist unter „Doku“ zu finden.

Im Ordner „Code“ ist sämtlicher Programmcode untergebracht. Hier befinden sich die Visual-Studio-Solution und ein weiterer Ordner „Lagerverwaltung“, der die Programmdateien enthält. Unter „bin“ sind die Builds zu finden. Unter „NUnitTests“ sind die Unit Test gespeichert.

## Installierte NuGet-Pakete

Die Folgenden NuGet sind installiert:

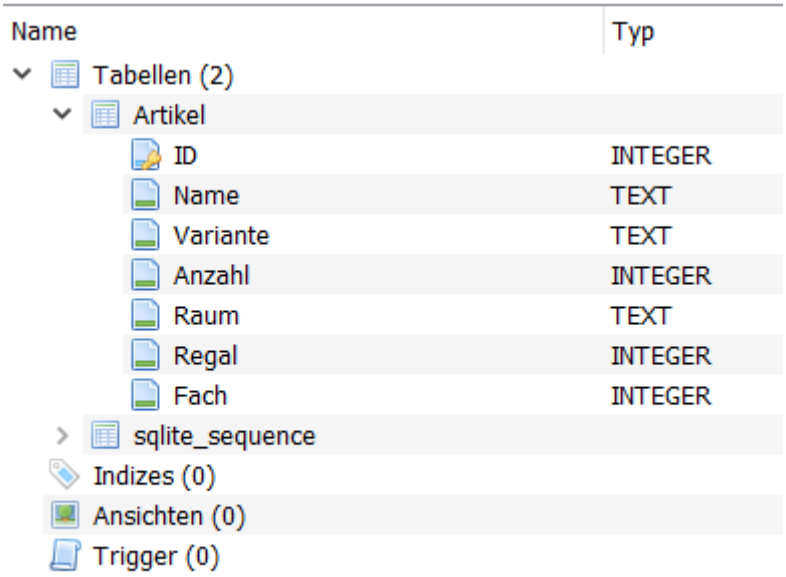
- System.Data.SQLite.Core
  Zum Ansprechen der SQLite-Datenbank
- Dapper
  zum einfachen Verbindungsaufbau und ausführen der SQL-Queries
- NUnit
  Das Framework für die Unit Tests

## Framework-Abhängigkeiten

Die Folgenden Frameworks wurden verwendet:

- Microsoft.NETCore.App
- Microsoft.WindowsDesktop.App.WindowsForms

## Datenbank Struktur




Legende für die Tabellen:

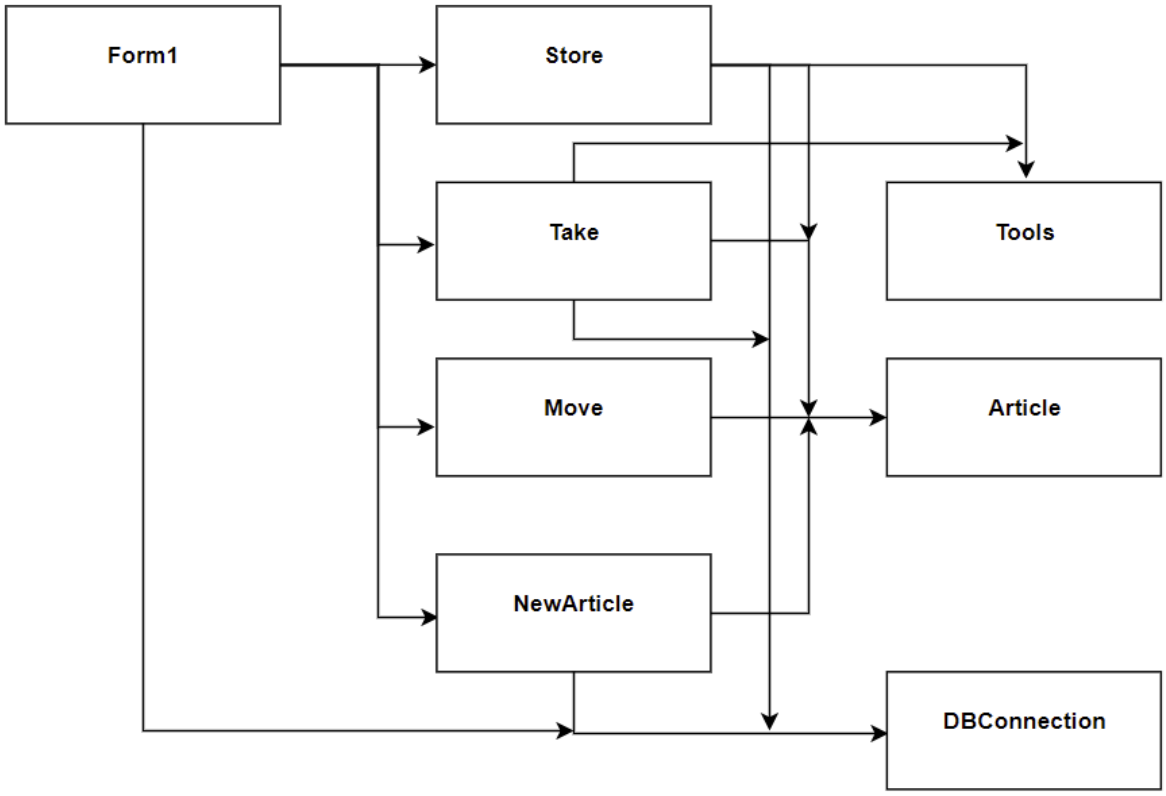
| | | |
|---|---|---|
| NN | Not Null | Das Feld darf nicht leer bleiben, beim Erstellen eines Datensatzes muss ein Wert angegeben werden. |
| PK | Primary Key | Primärschlüssel, identifiziert einen Datensatz eindeutig. Ist dieses Attribut gewählt, sind die Attribute NN und U automatisch mit ausgewählt. |
| AI | Auto inkrement | Der Wert dieses Feldes wird mit jedem neuen Datensatz automatisch inkrementiert. Beim Erstellen eines Datensatzes wird für dieses Feld kein Wert erwartet. |
| U | Unique | Der Inhalt dieses Feldes muss eindeutig sein. |

Die Datenbank besteht aus einer Tabelle, die die Artikel sowie ihren Standort enthält.

| Name | Typ | NN | PK | AI | U |
|---|---|---|---|---|---|
| ID | INTEGER | ☑ | ☑ | ☑ | ☑ |
| Name | TEXT | ☑ | ☐ | ☐ | ☐ |
| Variante | TEXT | ☐ | ☐ | ☐ | ☐ |
| Anzahl | INTEGER | ☑ | ☐ | ☐ | ☐ |
| Raum | TEXT | ☑ | ☐ | ☐ | ☐ |
| Regal | INTEGER | ☑ | ☐ | ☐ | ☐ |
| Fach | INTEGER | ☐ | ☐ | ☐ | ☐ |

Die Felder Variante und Fach sind nicht zwingend notwendig, weshalb sie den Wert „NULL“ annehmen dürfen. Ein Artikel könnte in lediglich einer Variante existieren, was das Feld nicht notwendig macht. Sollte ein Regal in einem Lager sehr klein sein oder dessen Funktion von einem Schrank oder einer Schublade übernommen werden, kann die Fach-Nummer ebenfalls weg gelassen werden.

## Funktion




## Klassenbeschreibung:

### *Form1:*

Die Haupt-Klasse hinter dem Haupt-Fenster. Hier werden die gesuchten und gefunden Datensätze dargestellt. Diese können ausgewählt werden, um sie zu modifizieren oder ein neuer Datensatz kann angelegt werden.

*Store:*
Wird aufgerufen, wenn ein Artikel ausgewählt ist und die Schaltfläche „Einlagern“ betätigt wird. Es wird die ID des ausgewählten Artikels übergeben und eine Anzahl abgefragt, die eingelagert werden soll.

*Take:*
Das Gagenstück zu „Store“. Anstatt die Anzahl dem Lagerbestand hinzuzufügen, wird sie subtrahiert.

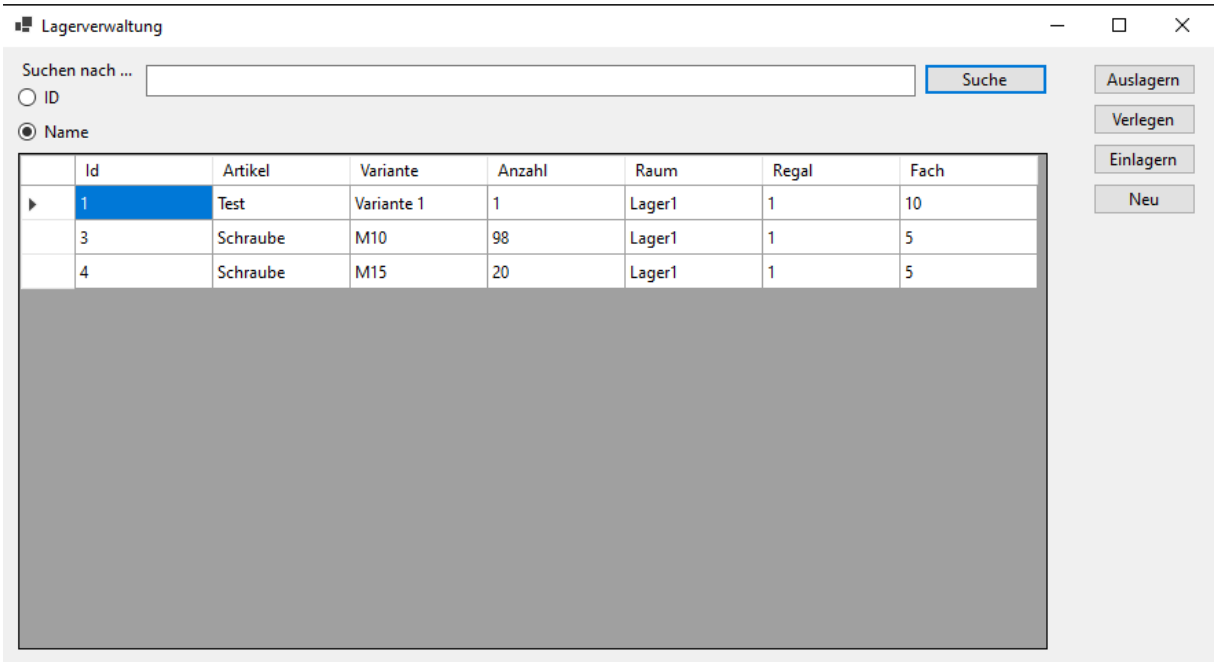
*Move:*
Wird ein Artikel ausgewählt und die Schaltfläche „Verlegen“ betätigt, zeigt diese Methode ein Dialogfenster an, das den neuen Raum, das neue Regal und das neue Fach abfragt und den Datensatz den User-Angaben entsprechend ändert.

*NewArticle:*
Öffnet ein Dialogfenster, wenn der User auf „Neu“ klickt. Nach dem Abfragen der Artikeldaten Name, Variante, Anzahl, Raum, Regal und Fach wird ein neuer Datensatz angelegt.

*Article:*
Eine Klasse, die das Instanziieren von Artikel-Objekten ermöglicht. Mit Hilfe dieser Objekte werden Artikelinformationen übermittelt und Artikel in Listen zusammengefasst.

*DBConnection:*
Über diese Klasse werden alle Datenbankzugriffe abgewickelt. Sie enthält Methoden für das Suchen, Einlagern, Auslagern und Verlegen von Artikeln.

*Tools:*
Eine Sammlung von Methoden, die von mehreren Methoden verwendet werden und durch das Methode-Extraction-Refactoring ausgelagert wurden.




## Suchen:
Durch einen Klick auf diese Schaltfläche werden die Einträge der Datenbank abgerufen, die zu den angegeben Such-String passen. Wird kein Such-String angegeben, werden alle Einträge angezeigt. Nach dem Anwählen eines Antrags können weitere Funktionen durchgeführt werden.

### Auslagern:
Nach der Angabe einer Anzahl wird die Entsprechende Anzahl eines Artikels ausgelagert.

### Einlagern:
Nach der Angabe einer Anzahl wird die Entsprechende Anzahl eines Artikels dem Bestand hinzugefügt.

### Neu:
Legt einen neuen Datensatz an, der in die Datenbank geschrieben wird.

# Clean Architecture

## Technologien
Es wurden die folgenden Technologien verwendet:

### C#
Als Programmiersprache wurde C# ausgewählt, weil mir diese vertraut ist. Außerdem ist es mit Hilfe des .Net-Frameworks und Windows Forms einfach, Desktopanwendungen und deren GUI zu gestalten.

### .Net Core 3.1.0
Das .Net-Framework wurde verwendet, da es das einfache Erstellen von GUI-Anwendungen für Windows-Maschinen ermöglicht. Frameworks besitzen den Nachteil, dass diese die Anwendung an sich binden. Es wird daher nahezu unmöglich, die Anwendung auf ein anderes Framework oder Libraries umzuziehen, ohne sie komplett neu zu schreiben.

### SQLite
SQLite bietet eine sehr kompakte, dennoch vollständige Datenbank. Der große Vorteil ist, dass kein gesonderter Datenbank-Server installiert und konfiguriert werden muss. Eine SQLite-Datenbank ist in einer einzigen Datei untergebracht, die auf derselben Maschine gespeichert ist, die auch den Programmcode ausführt. Das macht sie Zwar nicht zentral administrierbar, aber jede Installation der Anwendung bringt seine eigene Datenbank mit sich und kann somit nur die eigenen Daten lesen und schreiben.

### NUnit
Als Test-Framework wird NUnit eingesetzt. Es ermöglicht einfaches Testen von C# Code.

## Schichtenarchitektur
Die Anwendung wurde in zwei Schichten aufgebaut: der Application-Code und Plugins. Der Application Code umfasst die Klasse „Article“. Diese Klasse kann einzeln kompiliert werden und ist weder vom .Net-Framework noch von anderen Klassen Abhängig.

Die GUI-Komponenten sind alle vom .Net-Framework abhängig und somit nicht teil des langlebigen Codes. Sie sind vom Lebenszyklus und der Verfügbarkeit des .Net-Frameworks abhängig.

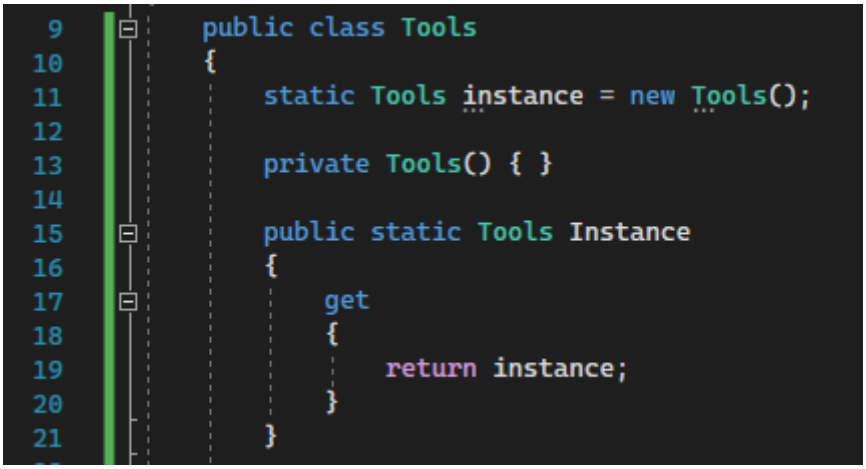
Ebenso verhält sich die Klasse „DBconnector“. Sie ist die Schnittstelle zur SQLite Datenbannk. Anders als beim .Net-Framework lässt sich die Datenbank einfacher wechseln. Es muss lediglich die DBconnector-Klasse verändert werden, um eine andere Datenbanktechnologie verwenden zu können.

# Entwurfsmuster

## Implementiertes Entwurfsmuster: Singelton

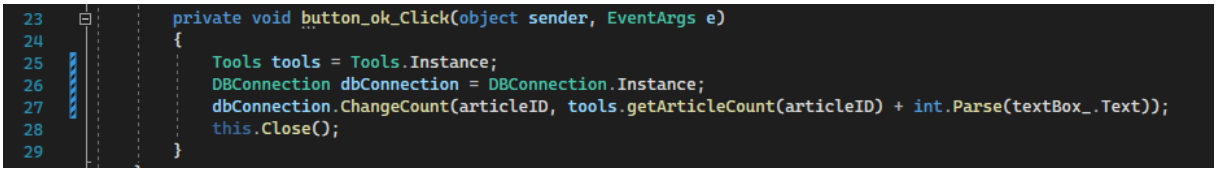
Die Klassen „Tools“ und „DBConnection“ sind Methoden-Sammlungen, die global verfügbar sein und nur einmal instanzierbar sein sollen. Daher wurde das Singelton-Entwurfsmuster implementiert, da dieses sich genau für diese Fälle eignet.

In folgender Abbildung ist der Code zu sehen, der das Singelton-Entwurfsmuster implementiert:



```
 9    public class Tools
10    {
11        static Tools instance = new Tools();
12
13        private Tools() { }
14
15        public static Tools Instance
16        {
17            get
18            {
19                return instance;
20            }
21        }
```

Der Konstruktor der Klasse wird auf private gesetzt, damit diese nicht instanziiert werden kann. Es wird genau eine Instanz angelegt, die bei Bedarf referenziert wird. Um auf die Instanz zugreifen zu können, wird in der Klasse „Store folgender Code verwendet:



```
23    private void button_ok_Click(object sender, EventArgs e)
24    {
25        Tools tools = Tools.Instance;
26        DBConnection dbConnection = DBConnection.Instance;
27        dbConnection.ChangeCount(articleID, tools.getArticleCount(articleID) + int.Parse(textBox_.Text));
28        this.Close();
29    }
```

Die Instanzen von „Tools“ und „DBConnection“ werden abgefragt und im Anschluss können ihre Methoden wie bei einem gewöhnlichen Objekt verwendet werden.

## Unit Tests

Für die Unit Tests wird das Framework „NUnit“ verwendet.

### ATRIP

Bewertung der geschrieben Tests anhand der ATRIP-Regeln:

#### Automatic

Die Tests Laufen automatisch durch, kein Eingreifen erforderlich.

#### Thorough

Die Tests sind für dieses Projekt zu wenig und decken nur einen kleinen Teil des Codes ab.

#### Repeatable

Manche Tests sind nur wiederholbar, wenn sich der Zustand der Datenbank nicht ändert.

#### Independent

Die Tests sind voneinander unabhängig, sie laufen in egal welcher Reihenfolge.

#### Professional

Der Testcode ist leicht verständlich, die Tests sind kurz und lesbar.

# Refactoring

## Code Smells

### Doppelter Code

Doppelter Code kann in den Klassen „Take“ und „Store“ vor. Beide Klassen müssen zunächst die aktuelle Anzahl der eingelagerten Artikel ermitteln, bevor sie die neue Anzahl berechnen können.

„Button_ok_Click“-Methode der Klasse „Store“:

```
23      private void button_ok_Click(object sender, EventArgs e)
24      {
25          List<Article> articles = new List<Article>();
26          articles = DBConnection.LoadArticlesId(articleID.ToString());
27          int oldCount = articles[0].Anzahl;
28          int newCount = oldCount + int.Parse(textBox_.Text);
29
30          DBConnection.ChangeCount(articleID, newCount);
31          this.Close();
32      }
```

„Button_ok_Click“-Methode der Klasse „Take“:

```
24
25      private void button_ok_Click(object sender, EventArgs e)
26      {
27
28          List<Article> articles = new List<Article>();
29          articles = DBConnection.LoadArticlesId(articleID.ToString());
30          int oldCount = articles[0].Anzahl;
31          int newCount = oldCount - int.Parse(textBox1.Text);
32
33          DBConnection.ChangeCount(articleID, newCount);
34          this.Close();
35
36      }
```

Der einzige unterschied beider Klassen ist die Berechnung der neuen Anzahl.

*Refactoring:*

Die Refactoring-Methode, die diesen Code-Smell entfernt, ist „Extract Method“. Es wurde eine Klasse angelegt, die diese extrahierten Methoden beinhaltet. Weil sie wie eine Werkzeugkiste funktioniert, nennt sich die Klasse „Tools“. Zum Vergleich sind hier die Methoden nach dem Refactoring abgebildet:

Aus Klasse „Store“:

```
23      private void button_ok_Click(object sender, EventArgs e)
24      {
25
26          int oldCount = Tools.getArticleCount(articleID);
27          int newCount = oldCount + int.Parse(textBox_.Text);
28
29          DBConnection.ChangeCount(articleID, newCount);
30          this.Close();
```

Aus Klasse „Take“:

```
25          private void button_ok_Click(object sender, EventArgs e)
26          {
27              int oldCount = Tools.getArticleCount(articleID);
28              int newCount = oldCount - int.Parse(textBox1.Text);
29
30              DBConnection.ChangeCount(articleID, newCount);
31              this.Close();
```

Der Extrahierte Code ist in der Methode „getArticleCount“ der Klasse „Tools“ untergebracht. Die Methode holt die aktuelle Artikel-Anzahl aus der Datenbank und gibt diese als Rückgabewert zurück. Als Parameter wird ihr die Article-ID übergeben.

Extrahierte Methode „getArticleCount“:

```
12          public static int getArticleCount(int id)
13          {
14              List<Article> articles = new List<Article>();
15              articles = DBConnection.LoadArticlesId(id.ToString());
16              int oldCount = articles[0].Anzahl;
17              return oldCount;
18          }
```

## Auskommentierter Code

Es beim Durchsuchen des Quellcodes wurde Auskommentierter Code gefunden. Dieser wird nicht verwendet und erfüllt auch keinen anderen Zweck. Der Code wurde gelöscht.

Auskommentierter, alter Code der Methode „button_ok_Click“ der Klasse „Store“. Dieser Code wurde durch neuen ersetzt und erfüllt keine Funktion. Ähnlicher Code befand sich in der Klasse „Take“:

```
33
34              /*
35              List<Article> articles = new List<Article>();
36              articles = DBConnection.LoadArticlesId(articleID.ToString());
37              int oldCount = articles[0].Anzahl;
38              int newCount = oldCount + int.Parse(textBox_.Text);
39
40              DBConnection.ChangeCount(articleID, newCount);
41              this.Close();
42              */
43
```

Connection String in der Klasse „DBConnection“. Dieser Kommentar wurde als Notiz erstellt und erfüllt keinen Zweck für die laufende Applikation:

```
11      {
12          public class DBConnection
13          {
14              // String Connectionstring = "Data Source=.\DB.db;Version=2;";providerName="System.Data.SqlClient
15
```

## Replace Temp with Query

Einige temporäre Variablen konnten eingespart werden, indem die Berechnung direkt in der Ausgabe bzw. im return-Statement durchgeführt werden. Ein Beispiel hierfür findet sich in der Methode „button_ok_Click“ der Klasse „Store“. Hier ist der Code vor dem Refactoring zu sehen:

```
23      private void button_ok_Click(object sender, EventArgs e)
24      {
25
26          int oldCount = Tools.getArticleCount(articleID);
27          int newCount = oldCount + int.Parse(textBox_.Text);
28
29          DBConnection.ChangeCount(articleID, newCount);
30          this.Close();
31      }
```

Hier ist der Code nach dem Refactoring abgebildet:

```
23      private void button_ok_Click(object sender, EventArgs e)
24      {
25
26          DBConnection.ChangeCount(articleID, Tools.getArticleCount(articleID) + int.Parse(textBox_.Text));
27          this.Close();
28      }
29  }
```

Sowohl der Aufruf der Tools.getArticleCount als auch die Addition der beiden Werte findet nun direkt innerlb der Parameterübergabe an die Methode DBConnection.ChangeCount. Die Beiden temporären Variablen „oldCount“ und „newCount“ sind nicht mehr notwendig. Ein ähnlicher Fall lag in der Methode „button_ok_Click“ der Klasse „Take“ vor.