

Studiengang: Wirtschaftsinformatik Bachelor



**Westfälische
Hochschule**

Sommersemester 2017

4. Fachsemester

Architektur betrieblicher Informationssysteme

Semesteraufgabe 8 - Entwurfsmuster

Abgabedatum:

12. Juli 2017

Aktualisierungsdatum:

18. Juli 2017

Autoren:

Mehmet Tüfekci (Matr. 201521617) – Mehmet.Tuefekci@studmail.w-hs.de

Mario Kellner (Matr. 201520916) – Mario.Kellner@studmail.w-hs.de

Julian Kranen (Matr. 201223532) – Julian.Kranen@studmail.w-hs.de

Aufgabe 1 (Konzept und Aufbau des MVC-Musters)

Teilaufgabe A)

Aus welchen Komponenten besteht das MVC-Muster und welche Aufgabe hat die jeweilige Komponente?

Model

Die erste Ebene dieses Musters ist die Modelkomponente. Sie repräsentiert Daten in der Anwendung und implementiert das Muster „Observer“ um Änderungen der View mit zu teilen.

Die Aufgabe des Models ist es Daten unabhängig von dem Controller und der View zu halten. Durch diese Eigenschaft kann es zu einem Modell mehrere Controller und mehrere Views geben.

View

Die nächste Komponente ist die View. Diese repräsentiert die Interaktionsschicht, in den meisten Fällen eine GUI, für den Benutzer.

Sie hält jeweils Referenzen zu den Models und dem Controller.

Somit hat die View die Aufgabe Daten aus dem Model anzuzeigen. Dies kann auf unterschiedliche Art, z. B. als Tabelle, geschehen.

Controller

Die Logikschicht bildet der Controller. In dieser Komponente wird auf Benutzereingaben reagiert, wertet diese aus und agiert entsprechend.

Teilaufgabe B)

Welche Muster finden sich im zusammengesetzten MVC-Muster wieder? Nennen und beschreiben Sie (kurz!) drei enthaltene Muster.

Strategie

Wird verwendet in: **View & Controller**

Eine Strategie beschreibt, was ein Objekt zutun hat ohne es aber direkt zu implementieren.

In unserem Konzept bedeutet das, dass die View weiß, dass es ein Controller gibt und was dieser kann, aber nicht weiß, was dieser genau macht.

Composite

Wird verwendet in: **View**

Mittels dem Composite ist es Ganz-teil Hierarchien zu implementieren.

Im Kontext des Musters kann eine View aus mehreren Komponenten bestehen.

Observer

Wird verwendet in: **Modell**

Das Muster Observer informiert registrierte Klassen über Veränderungen an den eigenen Eigenschaften.

Im Kontext des MVC Konzept informiert das Modell so die View, dass sich Daten geändert haben.

Teilaufgabe C)

In den Java Swing-Klassen wird das MVC-Muster nicht 1:1 implementiert. Welche Veränderung wird dort vorgenommen und warum geschieht dies?

Die Komponenten in Swing werden in einer Delegation zusammengefasst. Dieses Design wurde aus Performance-Gründen so gewählt, da beim MVC-Muster erhöhte Aufrufbeziehungen zwischen View und Controller bestehen und so versucht wird verschachtelte Aufrufe zu minimieren.

Aufgabe 2 (Implementierung eines Beispiels in Java)

Implementieren Sie eine sehr einfache Mini-Tabellenkalkulation: Erstellen Sie eine Tabelle mit drei Spalten und fünf Zeilen. In der ersten Spalte sollen zu Beginn die Zahlen 1 bis 5 ausgegeben werden. Der Benutzer kann hier aber auch eigene Werte eingeben. In der zweiten Spalte sollen automatisch die Quadratwurzeln der Zahlen aus der ersten Spalte stehen (am Anfang also 1, 1,414, 1,732, ...). In der dritten Spalte schließlich sollen die 3. Wurzeln (am Anfang also 1, 1,26..., 1,442..., ...) ausgegeben werden.

Erläutern Sie im Quellcode kurz, welche Methoden Sie wie und warum implementieren.

Die Basis des TableModels bildet die Rumfimplementierung, die uns in Moodle zur Verfügung gestellt worden ist. Statt das Interface „TableModel“ zu implementieren, besteht in Swing die Möglichkeit von der abstrakten Klasse „AbstractTableModel“ zu erben. Der Vorteil besteht darin, dass weniger Methoden implementieren werden müssen, als wenn wir direkt das Interface implementieren.

Folgende Methoden müssen implementiert werden, wenn von „AbstractTableModel“ abgeleitet worden ist:

getRowCount Gibt die Anzahl der Zeilen aus

getColumnCount Gibt die Anzahl der Spalten aus

columnName Gibt den Namen der Spalte mittels eines Index aus

getValueAt Gibt an, welchen Wert eine Zelle besitzt.

Unser implementiertes Model sieht folgendermaßen aus:

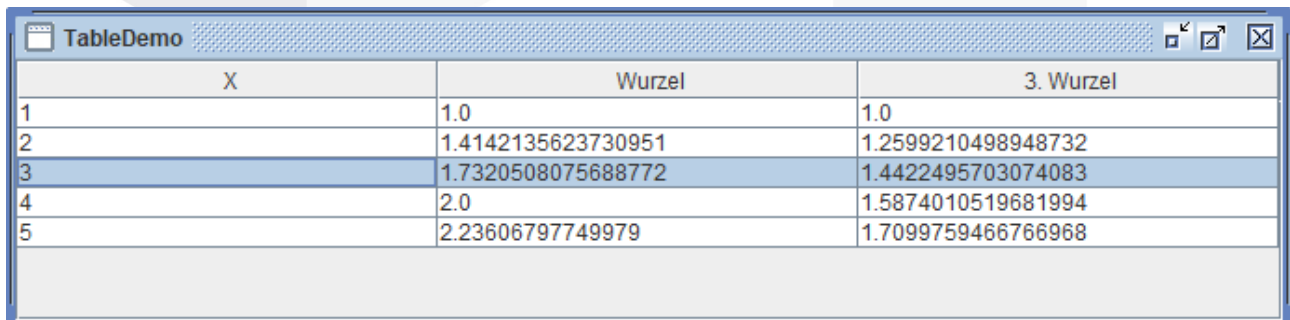
```
1 package tableModel;
2
3 import javax.swing.table.*;
4
5 class WurzelTableModel extends AbstractTableModel {
6     final String[] columnNames = {"X", "Wurzel", "3. Wurzel"};
7     int[] ersteSpalte = {1,2,3,4,5};
8     private static final long serialVersionUID = 930096348695238927L;
9
10    /**
11     * Da wir nur 3 Spalten haben, ist hier hardcoded eine 3
12     */
13    public int getColumnCount() {
14        return 3;
15    }
16    /**
17     * Zurückgegeben wird hier die Länge des Arrays ersteSpalte.
18     */
19    public int getRowCount() {
20        return ersteSpalte.length;
```

Aufgabe 2 (Implementierung eines Beispiels in Java)

```
21  }
22
23
24  /**
25   * Mit dieser Funktion wird der Name der Spalte zurückgegeben.
26   * Zugriffen wird auf das Array Columnname
27   */
28  public String getColumnName(int columnIndex) {
29      return columnNames[columnIndex];
30  }
31  /**
32   * Diese Funktion errechnet unsere Werte und gibt die für jede einzelne
33   * Position entsprechend aus
34   */
35  public Object getValueAt(int rowIndex, int columnIndex) {
36      switch (columnIndex) {
37          case 2:
38              return Math.pow(ersteSpalte[rowIndex], (1.0/3));
39          case 1:
40              return Math.sqrt(ersteSpalte[rowIndex]);
41          case 0:
42              default:
43                  return ersteSpalte[rowIndex];
44      }
45  }
46 }
```

Listing 1: WurzelTableModel.java

Unser Ausgabe sieht dann folgendermaßen aus:



X	Wurzel	3. Wurzel
1	1.0	1.0
2	1.4142135623730951	1.2599210498948732
3	1.7320508075688772	1.4422495703074083
4	2.0	1.5874010519681994
5	2.23606797749979	1.7099759466766968