|  |
| --- |
| **DataAcquisition** |
|  |
|  |
|  |
|  |
| **F. Weber** |
| **14.06.11** |
|  |

Inhalt

[Softwarebeschrieb 3](#_Toc295979082)

[UseCase 3](#_Toc295979083)

[Klassendiagramm 4](#_Toc295979084)

[Übersicht 4](#_Toc295979085)

[Package domain 5](#_Toc295979086)

[DataManager 5](#_Toc295979087)

[Data 5](#_Toc295979088)

[Generator 5](#_Toc295979089)

[Package gui 6](#_Toc295979090)

[DataGui 7](#_Toc295979091)

[DatabaseSettingsGui 7](#_Toc295979092)

[DataManagerGui 7](#_Toc295979093)

[DataPlotGui 7](#_Toc295979094)

[DataPlot 7](#_Toc295979095)

[DataPlotPopupMenu 7](#_Toc295979096)

[Package persistence.db 8](#_Toc295979097)

[Database 8](#_Toc295979098)

[Dummy 8](#_Toc295979099)

[CSV 8](#_Toc295979100)

[MysqlConnector 8](#_Toc295979101)

[Package tester 9](#_Toc295979102)

[Test 9](#_Toc295979103)

[Sequenzen 10](#_Toc295979104)

[Startsequenz 10](#_Toc295979105)

[Hinzufügen Sequenz 11](#_Toc295979106)

[JUnit Tests 11](#_Toc295979107)

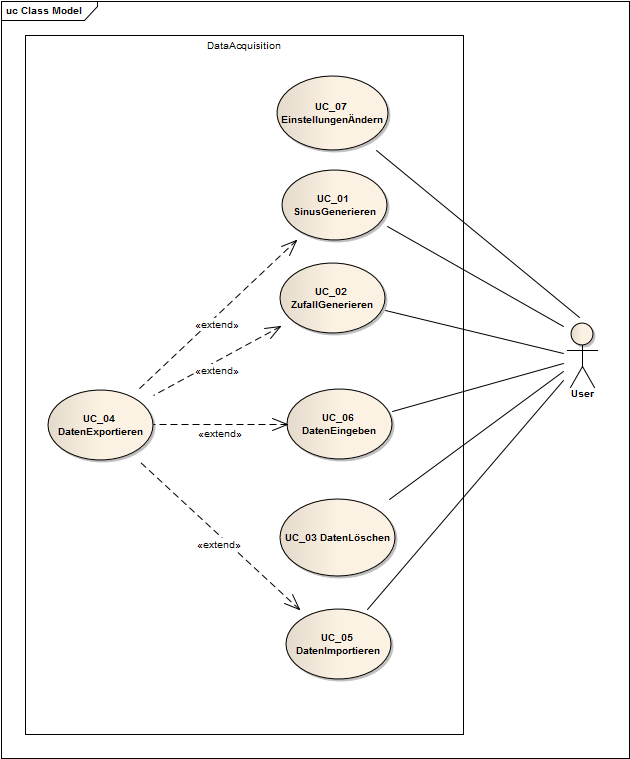
[Logger 12](#_Toc295979108)

[Versionskontrolle 12](#_Toc295979109)

# Softwarebeschrieb

Mit der Software können Koordinaten eingegeben oder erzeugt werden die danach in einem Koordinatensystem angezeigt werden. Die Daten können als Random oder als Sinus erzeugt werden. Danach können die Daten wieder gelöscht werden. Die Daten selber werden (falls vorhanden) in einer MySQL Datenbank persistiert. Ansonsten werden sie nur während der Laufzeit des Programmes gespeichert. Die Daten können als CSV exportiert und importiert werden.

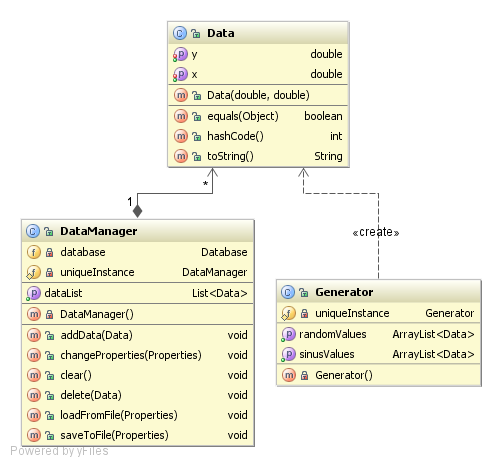
# UseCase



# Klassendiagramm

## C:\Users\tzdwef\Desktop\Neuer Ordner\DataAcquisition\doc\Gesamtuebersicht.pngÜbersicht

## Package domain



### DataManager

Der DataManager enthält eine Liste mit allen Daten und eine Klasse welche Database implementiert. Die Klasse enthält Funktionen um Daten hinzuzufügen, zu löschen, von einer Datei zu importieren und um zu exportieren. Der DataManager dient so als Schnittstelle zwischen der jeweiligen Datenbank und den Datenlieferanten und Datenbezüger. Der DataManager ist als Singleton implementiert.

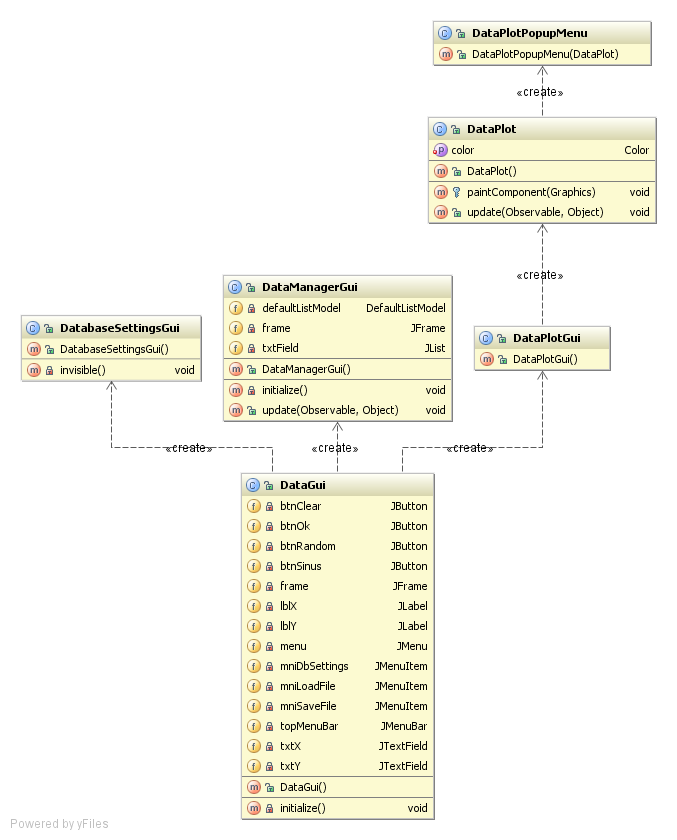
### Data

Ein Data Objekt repräsentiert ein Datensatz welcher ein X und einen Y Wert enthält. Die Klasse bietet Methoden um die Werte zu holen und zu vergleichen. Die Werte werden beim erzeugen des Objektes im Konstruktor definiert und können später nicht mehr geändert werden.

### Generator

Der Generator generiert Zufallswerte und Sinuswerte. Die entsprechenden Methoden geben eine Liste mit den Werten zurück. Der Generator ist als Singleton implementiert.

## Package gui



### DataGui

Das DataGui ist das Hauptfenster des Programms. Es bietet dem User die meisten Kontrollfunktionen wie z.B. Daten eingeben oder generieren. Das DataGui erzeugt die anderen Fenster.

### DatabaseSettingsGui

Das DatabaseSettingsGui wird geöffnet sobald der User die Datenbankeinstellungen anpassen will. In diesem Fenster kann er alle Datenbankeinstellungen vornehmen. Wurden die Einstellungen geändert verschwindet das Fenster.

### DataManagerGui

Das DataManagerGui zeigt dem User alle bereits vorhandenen Daten in einer Liste an. Falls die Liste länger ist als die Höhe des Fensters erscheint ein Scrollbalken. Das Objekt trägt sich beim DataManager als Observer ein. Sobald die Update Methode aufgerufen wird, werden die Daten in der Liste aktualisiert.

### DataPlotGui

Das DataPlotGui bietet einen Container für ein DataPlot Objekt welches vom DataPlotGui erzeugt wird.

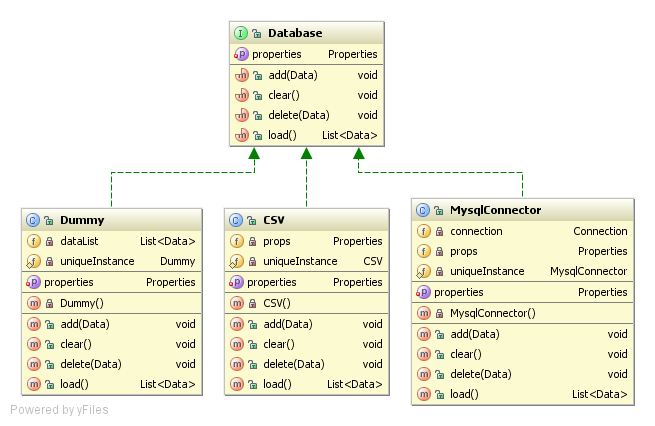
### DataPlot

Der DataPlot zeigt ein Koordinatensystem mit den vorhandenen Daten in einem Fenster an. Er trägt sich beim DataManager als Observer ein und zeichnet die Daten bei Veränderungen neu. Auch bei Grössenänderungen des Fensters werden die Daten abhängig von der Grösse des Fensters neu gezeichnet. Der DataPlot erzeugt das DataPlotPopupMenu.

### DataPlotPopupMenu

Das DataPlotPopupMenu erbt vom JPopupMenu. Im Konstruktor muss das DataPlot Objekt übergeben werden. Es zeigt bei einem Rechtsklick auf den DataPlot ein Kontextmenu mit einer Farbauswahl. Der User kann dann auswählen welche Farbe die Koordinaten haben sollen.

## Package persistence.db



### Database

Das Interface Database gibt die benötigten Methoden vor, welche eine Datenbank die vom DataManager gebraucht wird haben muss. Alle Database Implementierungen sind als Singleton ausgeführt.

### Dummy

Die Dummy Database wird gebraucht falls keine MySQL Datenbank vorhanden ist. Sie hält alle Daten in einer Liste. Bei der Dummy Database können keine Eigenschaften verändert werden.

### CSV

Die CSV Database wird gebraucht sobald Daten als CSV Datei importiert oder exportiert werden müssen. In der Eigenschaft „file.name“ wird der Dateinamen angegeben von der das Objekt liest und auf welches es schreibt. Falls die Datei nicht gefunden werden kann wird eine Exception geworfen.

### MysqlConnector

Diese Database verwendet den JDBC Treiber für MySQL um mit einer MySQL Datenbank zu kommunizieren. Falls keine Datenbank erreichbar ist oder ein anderer Fehler auftritt wird eine Exception geworfen.

## Package tester

### Test

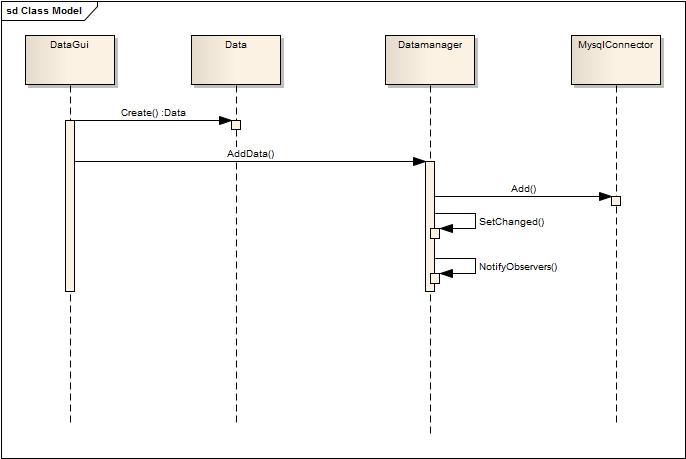
Die Klasse Test enthält die main Methode welche den Logger konfiguriert und das DataGui erzeugen.

# Sequenzen

## C:\Users\tzdwef\Desktop\Neuer Ordner\DataAcquisition\doc\SequenzStart.pngStartsequenz

Zuerst wird in der Test Klasse die main Methode aufgerufen. Diese Methode erzeugt das DataGui welche zuerst das DataPlotGui und danach das DataManagerGui erzeugt. Das DataManagerGui erzeugt den DataManager welcher standardmässig den MysqlConnector erzeugt.

## Hinzufügen Sequenz



Wenn der User einen Datensatz manuell hinzufügt erzeugt das DataGui zuerst ein Data Objekt. Danach übergibt er dieses Objekt dem DataManager welcher es im MysqlConnector oder in der entsprechenden Database speichert und danach sich selber als geändert markiert und die Observer benachrichtig.

# JUnit Tests

Die JUnit Tests sind alle im Ordner test vorhanden. Es werden alle relevanten und testbaren Methoden mittels JUnit getestet. Dazu gehören insbesondere auch die Klassen welche Database implementieren. Bei den Singleton wird zusätzlich getestet ob die Singleton Implementierung korrekt arbeitet und ein Objekt zurückgegeben wird.

# Logger

Als Logger wird in dieser Applikation Log4j von verwendet. Der Logger wird mittels einer XML Datei konfiguriert. In dieser Applikation heisst die XML Datei log.xml. Eine Beispielkonfiguration könnte etwa so aussehen:

*<?xml version="1.0" encoding="UTF-8" ?>  
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">  
<log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/">  
 <appender name="console" class="org.apache.log4j.ConsoleAppender">  
 <param name="Target" value="System.out"/>  
 <layout class="org.apache.log4j.PatternLayout">  
 <param name="ConversionPattern" value="%d - %p %c - %m%n"/>  
 </layout>  
 </appender>  
 <root>  
 <priority value="trace"/>  
 <appender-ref ref="console"/>  
 </root>  
</log4j:configuration>*

Die Konfiguration des Loggers kann durch anpassen der XML Datei auch während der Laufzeit der Applikation geändert werden und die Einstellungen werden übernommen.

Der Logger hat verschiedene Loglevels wie z.B. Error oder Info. Im Programm selber sind die Loglevels der Nachrichten fest eingestellt. Welche Lognachrichten angezeigt werden hängt dann von der Konfiguration ab. Ein Logaufruf könnte etwa so aussehen:

*Logger.getLogger(DataManager.class).trace(„Neues DataManager Objekt wurde erzeugt“);*

Man kann dem Logger auch zusätzlich ein Throwable mitgeben welches dann geloggt wird.

# Versionskontrolle

Die Versionskontrolle wird in diesem Projekt mit Git gemacht. Das Projekt inklusive der verschiedenen Versionen kann man auf <https://github.com/frederikweber/DataAcquisition> finden.