# Einleitung

Dieses Dokument ist dazu da, einen schnelleren Start in das Projekt *Statistische Geheimhaltung* zu ermöglichen. Dazu werden hier wichtige Eigenschaften und Dateien der Software aufgeführt. Es handelt sich jedoch nicht um eine Detaildokumentation, die alle Klassen vollständig beschreibt.

Da ich nicht selbst an der Software programmiert habe (außer Testfälle) und daher die Programmarchitektur auch nicht mit entwickelt habe, sind hier nur Ansatzpunkte erwähnt, die mit Sicherheit nicht alle Probleme behandeln. Ob es sich also anbietet auf einem Ansatzpunkt aufzubauen oder eine komplett eigene Lösung ohne die Verwendung von bereits vorhandenen Einlesemethoden etc. zu implementieren, muss je nach konkretem Fall entschieden werden.

# Projektstruktur

## Quelltextverwaltung

Zur Quelltextverwaltung wird *git* genutzt. Eingecheckt werden prinzipiell alle Dateien, die notwendig sind, um das Programm zu bauen. Da keine Online Programmierbibliotheksverwaltung wie *maven* oder ähnliches genutzt wird, müssen deshalb auch neuhinzugefügte Programmierbibliotheken eingecheckt werden. Dateien, die explizit nicht genutzt werden, werden wie bei *git* üblich in die *.gitgnore* Datei eingetragen. Dort finden sich im Moment folgende Einträge:

|  |  |
| --- | --- |
| Eintrag | Beschreibung |
| \*.class | Alle Binärdateien nach dem Übersetzen des Programmes |
| Build | Der Ordner, der die Binärdateien enthält |
| Classes | Falls ein Entwickler das Projekt doch unter marven Verwaltung hat, liegen hier die Binärdateien |
| *.settings/org.eclipse.php.core.prefs* | Die lokalen Einstellungen des Entwicklers an seiner Entwicklungsumgebung |
| *src/test/java/resources/mdav\_output.csv* | Eine Testdatei, die bei jedem Durchlauf neu erzeugt wird |

## Quelldateien

Der aktuelle Projektstand ist aus zwei Programmen von verschiedenen, vorangegangenen Projektgruppen zusammengesetzt worden. Daher gibt es auf oberster Ebene drei Pakete:

|  |  |  |
| --- | --- | --- |
| Paket | Projektgruppe | Inhalt |
| de.htw.sg | 2012 | * Microdatengeheimhaltung * Oberfläche des Tabs „Microdaten“ |
| matrixManipulation | 2013 | * Matrixmanipulation * Oberfläche des Tabs Matrixmanipulation |
| Startup | 2013 | * Hauptfenster der Oberfläche * Initialisierung der einzelnen Panels |

**Tipp:**

Obwohl die Komponenten nicht voneinander abhängen, ist es wahrscheinlich bei vielen Oberflächenfunktionen mit relativ wenig Aufwand möglich, eine der beiden Implementierungen in abgewandelter Form zu nutzen (beachten Sie jedoch Bekannte Fehler).

Historisch bedingt hängen die Komponenten auf den einzelnen Tabs deshalb nicht voneinander ab. Öffnen- und Speicherndialoge sowie Ausgabeformate sind unterschiedlich und werden im Programmcode von andern Klassen und Methoden bedient.

## Bibliotheken

Außerdem wurden verschiedene Java Bibliotheken genutzt, die im Verzeichnis *lib* zu finden sind. Diese Bibliotheken finden z.B. beim Lesen der CSV Dateien, beim Oberflächendesign und beim automatischen Testen der Anwendung Verwendung.

|  |  |  |
| --- | --- | --- |
| Bibliothek | Verwendung | URL |
| Designgridlayout-1.9 | Gruppe 2012: Als Layoutmanager für ihren Tab | https://designgridlayout.java.net/ |
| Forms-1.2.1 | Gruppe 2012: JGoodie Erweiterung um Formulare darzustellen | <http://grepcode.com/snapshot/repo1.maven.org/maven2/com.jgoodies/forms/1.2.1> |
| Guava-10.0.1 | Gruppe 2012: Bibliothek für interne Datenstrukturen | https://code.google.com/p/guava-libraries/ |
| Jsr305-1.3.9 | Gruppe 2012: Findbugs um Bugs durch statische Codeanalyse zu finden | <http://mvnrepository.com/artifact/com.google.code.findbugs/jsr305/1.3.9> |
| Junit-4.8.2 | Alle: Automatische Komponententests | http://junit.org/ |
| Looks-2.0.1 | Gruppe 2012: Interne Abhängigkeit von jGoodies | http://mvnrepository.com/artifact/com.jgoodies/looks/2.0.1 |
| Miglayout-core-4.2 | Gruppe 2012: weitere Layoutmanager für das Tab | <http://mvnrepository.com/artifact/com.miglayout/miglayout-core/4.2> |
| Miglayout-swing-4.2 | Gruppe 2012: siehe oben | <http://mvnrepository.com/artifact/com.miglayout/miglayout-core/4.2> |
| Opencsv-2.0 | Gruppe 2012: Lesen der CSV Dateien | http://opencsv.sourceforge.net/ |
| Slf4j-api-1.6.4 | Gruppe 2012: Logging | http://www.slf4j.org/ |
| Slf4j-jdk14-1.6.4 | Gruppe 2012: Logging | http://www.slf4j.org/ |

**Tipp:**

Da bereits eine Vielzahl an Bibliotheken zur Verfügung steht sollten Sie jedoch vor dem Hinzufügen von weiteren prüfen, ob sich die benötigte Funktion nicht bereits in einer vorhandenen befindet um die Programmgröße und Abhängigkeiten zu verringern.

## Konfiguration

Es existiert eine Konfigurationsdatei im Hauptverzeichnis der Anwendung die von der Projektgruppe 2012 erstellt wurde. Diese Datei hat nur Einfluss auf den *Microdaten* Tab und ermöglicht es dort, die Anzahl der Nachkommastellen und das Zahlenformat einzustellen.

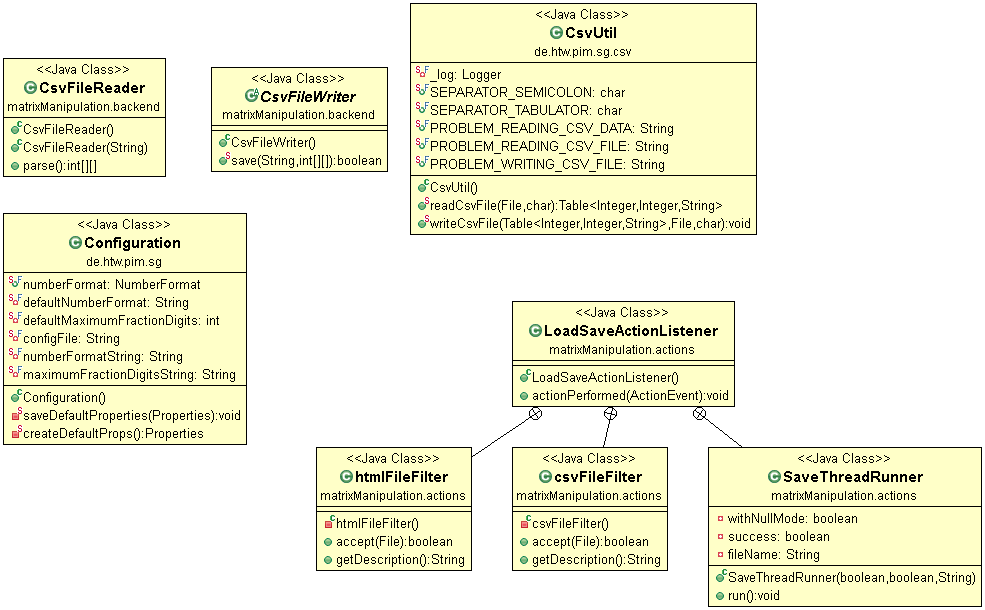
**Tipp:**

Benötigen Sie ebenfalls eine Konfigurationsdatei, so ist es wahrscheinlich mit einfachen Mitteln möglich, diese Datei zu erweitern. Um die Einträge der verschiedenen Verfahren unterscheiden zu können, sollte im Programmcode jedoch ein neues Schlüsselschema (z.B. *microaggretion\_numberFormat* anstelle von *numberformat*) eingeführt werden. Schnittstellentechnisch gibt es eine Klasse de.htw.pim.sg.*Configuration.java* die, erweitert werden kann.

# Klassenübersicht

Die obige Klassenübersicht beschreibt die wichtigsten bereits vorhandenen Klassen und deren Abhängigkeiten untereinander. Dabei werden sowohl die Oberflächenklassen als auch die Logikklassen dargestellt. Die Klasse *StatistischeGeheimhaltung* hat dabei die besondere Rolle, die zwei verschiedenen Ansätze der beiden Projektgruppen zusammenzufügen. Die Klassen *View* und *MainLauncher* sind die Haupteinstiegspunkte der verschiedenen Projektgruppen und zeichnen außerdem den zugehörigen Teil der Oberfläche. Sie bauen auf verschiedenen Organisationsstrukturen und Helferklassen auf. Alle weiteren Äste dieser Klassen beschäftigen sich mit den konkreten Verfahren. Dabei zeichnen sie entweder die Eingabekomponenten für ein Verfahren oder führen ein Verfahren aus. Aus Übersichtsgründen wurden die *Hilfsklassen,* die die Kopplung zwischen Verfahren und Oberfläche darstellen, weggelassen. Da die *Hilfsklassen* trotzdem nützlich sind, werden sie nachfolgend beschrieben.

## Nützliche Helferklassen



**CsvFileReader:** Diese Klasse liest eine Matrix aus einer CSV Datei.

**CsvFileWriter:** Diese Klasse schreibt eine Matrix in eine CSV Datei.

**CsvUtil:** Diese Klasse schreibt und liest ebenfalls CSV Dateien. Jedoch wird hier ein google Datenformat genutzt, um die Matrix im Speicher darzustellen. Dabei werden die Zellenelemente nicht als Zahlen sondern als Zeichenkette interpretiert.

**Configuration:** Hier wird die oben beschriebene Konfigurationsdatei geladen und die dort enthaltenen Einstellungen verwaltet.

**LoadSaveActionListener:** Mit dieser Klassenstruktur wird das Laden und Speichern von CSV Dateien aus der Oberfläche heraus organisiert. Dabei wird mithilfe eines zweiten Thread gespeichert, um die Oberfläche nicht einfrieren zu lassen.

# Bekannte Fehler

* Wird ein Öffnendialog im ersten Tab ohne die Auswahl einer Datei geschlossen, wird eine NullPointerException ausgelöst.
* In der Klasse CsvFileReader wird beim Lesen der Datei eventuell eine NullPointerException ausgelöst sofern keine Datei ausgewählt wurde (Methode checkIfFileExists).
* Oftmals gibt es in den Testfällen nicht mehr benötigte Variablen, die entfernt werden sollten.
* Die Klasse *DistanceElement* definiert die *compareTo* Methode jedoch nicht die *equals* Methode.
* Es gibt eine leere Klasse *MatrixManipulation*  im Paket *matrixManipulation.core*