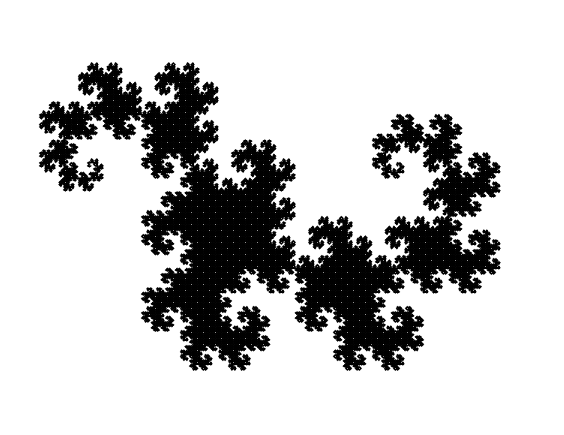
Grafische Darstellung von Fraktalen

Portfolio Arbeit

Von Jonah Sebright

Klasse 6A



EF-Lehrer: PD Dr. Victor Yakhontov

Gymnasium Kirschgarten Basel 2021

Inhaltsverzeichnis

[Inhaltsverzeichnis 1](#_Toc71807408)

[1. Einleitung 3](#_Toc71807409)

[1.1 Was sind Fraktale? 3](#_Toc71807410)

[1.2 Ziele des Projekts 3](#_Toc71807411)

[1.3 Vorschau Resultate 3](#_Toc71807412)

[2. Hauptteil (Material und Methoden) 4](#_Toc71807413)

[2.1 Programmiersprache 4](#_Toc71807414)

[2.2 Bibliothek StdDraw 4](#_Toc71807415)

[2.3 Reduktion des Problems 5](#_Toc71807416)

[2.4 Entwicklung der Module 5](#_Toc71807417)

[2.5 Entwicklung der Algorithmen 5](#_Toc71807418)

[2.5.1 Schneeflocke 5](#_Toc71807419)

[2.5.2 Pfeilspitze 5](#_Toc71807420)

[2.5.3 Drachenkurve 5](#_Toc71807421)

[2.6 Testing 5](#_Toc71807422)

[3. Resultate 5](#_Toc71807423)

[3.1 Bilder 5](#_Toc71807424)

[3.2 Effizienz 5](#_Toc71807425)

[3.3 Features des Main-Programms? Oder zu einleitung?? 5](#_Toc71807426)

[4. Diskussion 5](#_Toc71807427)

[Flächedrachenkurve 5](#_Toc71807428)

[4.1 Optimierungsmöglichkeiten 5](#_Toc71807429)

[4.2 Rückblick auf das Problem 5](#_Toc71807430)

[5. Nachwort? 6](#_Toc71807431)

[6 Quellenverzeichnis 6](#_Toc71807432)

[7. Abbildungsverzeichnis 6](#_Toc71807433)

[8. Tabellenverzeichnis 6](#_Toc71807434)

[9. Quellcodeverzeichnis 6](#_Toc71807435)

[10. Anhang 6](#_Toc71807436)

[6.3 Glossar? 6](#_Toc71807437)

[6.6 Weitere Diagramme? 6](#_Toc71807438)

[6.7 Gesamter Quellcode 6](#_Toc71807439)

[3.4.2 Dynamic app 6](#_Toc71807440)

[3.9 Testing 7](#_Toc71807441)

[4. Results 7](#_Toc71807442)

[4.4 Source code 7](#_Toc71807443)

[4.4 Comparison to products on the market 10](#_Toc71807444)

[5. Discussion 13](#_Toc71807445)

[5.1 Further development 13](#_Toc71807446)

[5.2 What could I have done differently? 13](#_Toc71807447)

[6. Afterword and acknowledgements 13](#_Toc71807448)

[7. References 14](#_Toc71807449)

[8. Figure index 15](#_Toc71807450)

[9. Listing index 16](#_Toc71807451)

[10. Table index 16](#_Toc71807452)

[11. Appendix 17](#_Toc71807453)

[11.1 Personas 17](#_Toc71807454)

[11.2 Questionnaire about GMAs 17](#_Toc71807455)

[11.3 Questionnaire about my app 17](#_Toc71807456)

[11.4 Libraries 17](#_Toc71807457)

[12.5 Complete data structure 17](#_Toc71807458)

[11.6 Further information on the accompanying CD 18](#_Toc71807459)

1. Einleitung

1.1 Was sind Fraktale?

Fraktale sind…

1.2 Ziele des Projekts

Das Hauptziel dieses Projekts ist, mit den gelernten Fähigkeiten und dem erworbenen Wissen ein konkretes Informatik-Problem zu lösen. Das Problem, das in diesem Projekt untersucht wurde, soll bestimmte Fraktale, wie die „Schneeflocke“, die „Pfeilspitze“ und die „Drachenkurve“ darstellen, bzw. zeichnen.

Zu diesem Problem soll ein Programm erstellt werden, das dem Benutzer eine Annäherung des ausgewählten Fraktals bis zu einer gewünschten Stufe grafisch wiedergibt. Zur Zeichnung der Kurve soll die Zeichnungs-Bibliothek „StdDraw“ verwendet werden. Sobald die Darstellung vollendet ist, wird der Benutzer gefragt, ob er die Zeichnung als pdf-Datei speichern möchte. Dabei kann der Benutzer den Dateinamen, selbst eingeben.

Für die grafische Darstellung der Fraktale müssen die Algorithmen der der einzelnen Fraktale gschrieben werden, die alle das Basisprinzip der Rekursion implementieren. Zudem verwenden alle Algorithmen bestimmte Hilfs-Klassen, wie z.B. die *Schildkroete*.

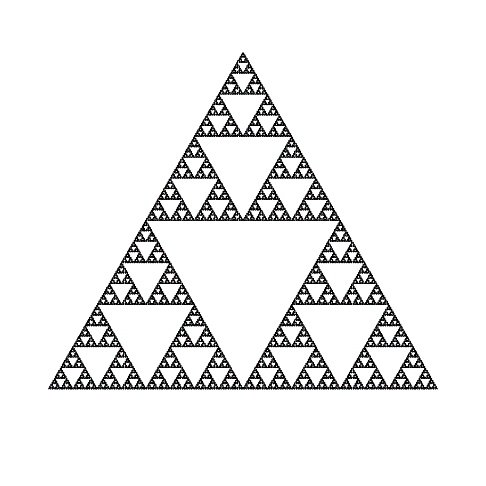
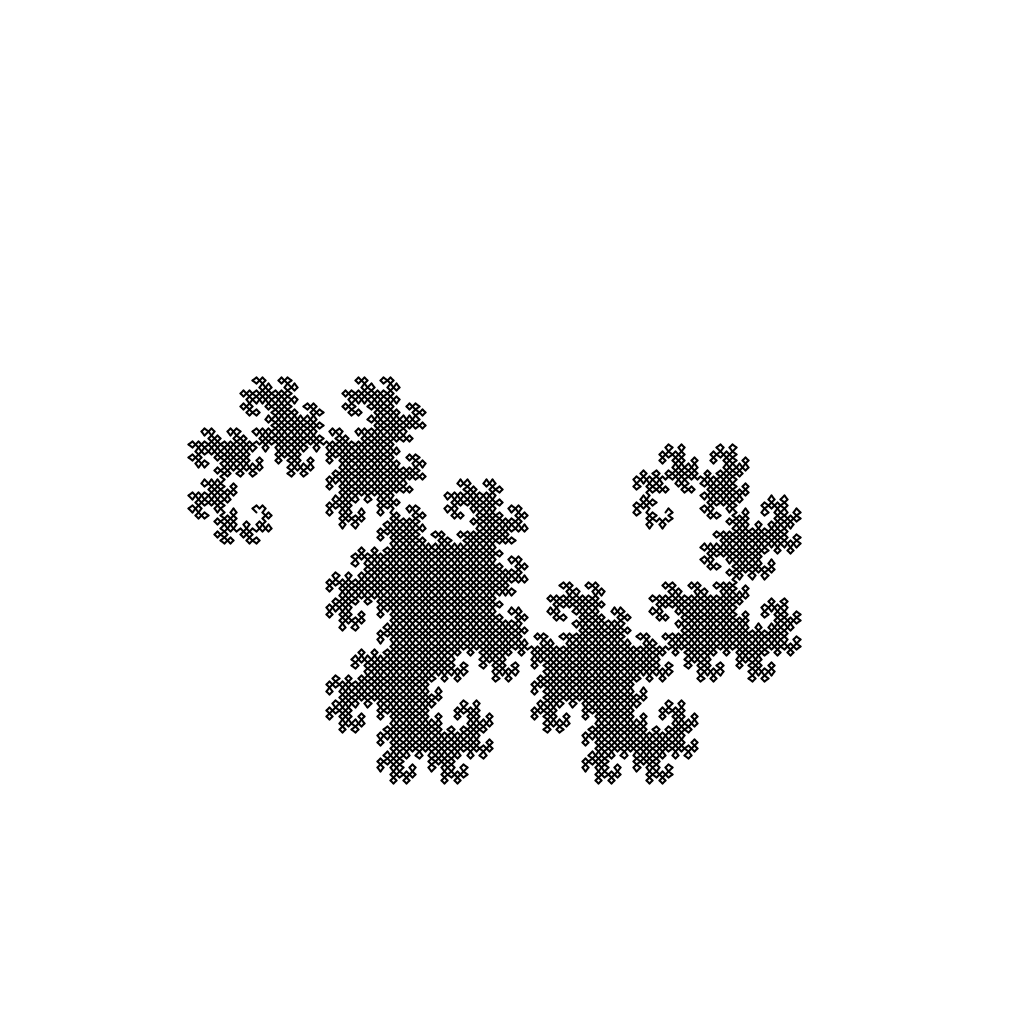
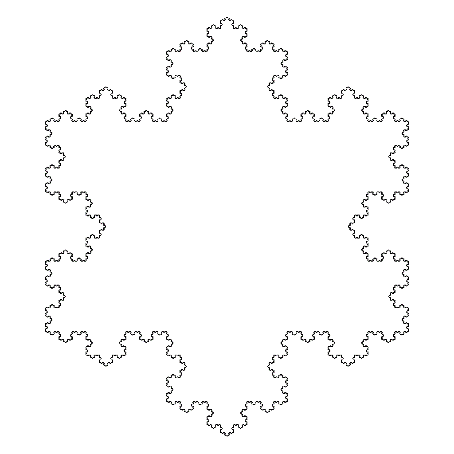
Ausserdem werden in diesem Projekt die Resultate des Programms analysiert und mit mathematischen Erkenntnissen verglichen. Bei der Drachenkurve wird in der Diskussion bewiesen, dass die unendlich lange Kurve eine endliche Fläche bedeckt.

Abgrenzung: konkrete ziele

Struktur der Arbeit

1.3 Vorschau Resultate

Die Zeichnungen des Programms sind staunend schön! Das Haupt-Programm läuft ohne Fehler und beinhaltet alle oben beschrieben Funktionalitäten und erfüllt somit dessen Ziele. Die einzelnen Algorithmen der Fraktale funktionieren ebenfalls einwandfrei bei beliebiger Stufe. Hier sind ein paar grafische Resultate der verschiedenen Fraktale:



Schneeflocke stufe 7 Drachenkurve Stufe 13 Pfeilspitze Stufe 9

Was und wie gemacht und was wird vorgsestellt, Darstellung

Welche Klasses erstellt

Das Programm besteht aus den Modulen *Schildkroete*, *Fraktal*, *Main*, *(StdDraw)*. Die abhängigkeit sieht dabei Folgendermassen aus:

INSERT\_DIAGRAM

Insgesamt hat wurden X Klassen erstellt mit einem Total von X tausend linien code. Eine gesamte Übersicht der Abhängigkeiten Klassen ist im Anhang INSERT\_HYPERLINK.

2. Hauptteil (Material und Methoden)

Eine Java-Applikation ist sehr komplex und kann durch falsche Entscheidung schnell zu Problemen führen. Geschicktes Vorgehen ist daher sehr wichtig und die Grundstrukturen müssen stimmen, bevor die Details implementiert werden können. Besonders mit wachsender Grösse des Programms ist die Architektur umso wichtiger.

Begonnen wird mit der konkreten Definition des Programms, was schon in der Einleitung gemacht wurde. Danach findet die Reduktion des Problems statt, wobei die Module und deren Beziehungen und der Grobalgorithmus skizziert werden. Anschliessend können die einzelnen Module und Algorithmen konkret implementiert werden. Möglicherweise werden dafür Bibliotheken verwendet, in diesem Fall „StdDraw (siehe Kapitel 2.2.1 StdDraw). Erst in diesem Schritt wird der Quellcode verfasst.

Damit das Programm bombenfest ist wurde es mit den erwarteten aber als auch unerwarteten Eingabewerten des Benutzers getestet. Dies ist wichtig, da es nie sicher ist, dass der Benutzer die erwarteten Werte eingibt. Zudem muss auch die Effizienz getestet werden, sodass ein langsames Programm bemerkt und optimiert werden kann.

Für die Verwendung des Programms muss zunächst eine Verbindung zwischen dem Benutzer und dem Programm hergestellt werden. Dies wird durch ein GUI (graphical user interface) gemacht, was die Interaktion erlaubt. In diesem Programm müssen lediglich ein paar Fenster zum Einlesen und zur Resultat-Ausgabe geöffnet werden. Die Bibliothek „Flanagan“ (siehe 2.2.2 Flanagan) erfüllt diese simplen Forderungen.

2.1 Programmiersprache

Wie vorgegeben, wurde das Programm ausschliesslich mit Java programmiert. Java ist eine objekt-orientierte Programmiersprache von Oracle und bietet so viele Vorteile, wie unter anderem Polymorphie (see Ullenboom 2012, p .50).

# **2.2 Bibliotheken**

### 2.2.1 StdDraw

Für das Programm wurde eine Zeichnungsbibliothek names „StdDraw“ vom *Computer Science* Departement der Universität Princeton verwendet. Die Bibliothek bietet eine Zeichenfläche und nützliche Methoden dazu an, die dazu dienen, die Zeichenfläche zu manipulieren. Die wichtigsten Methoden für dieses Projekt sind *line, setPenColor, setPenRadius, clear*. Eine weitere sehr wichtige Funktion ist *save*, mit der die Zeichenfläche als Bild-Datei gespeichert werden kann.

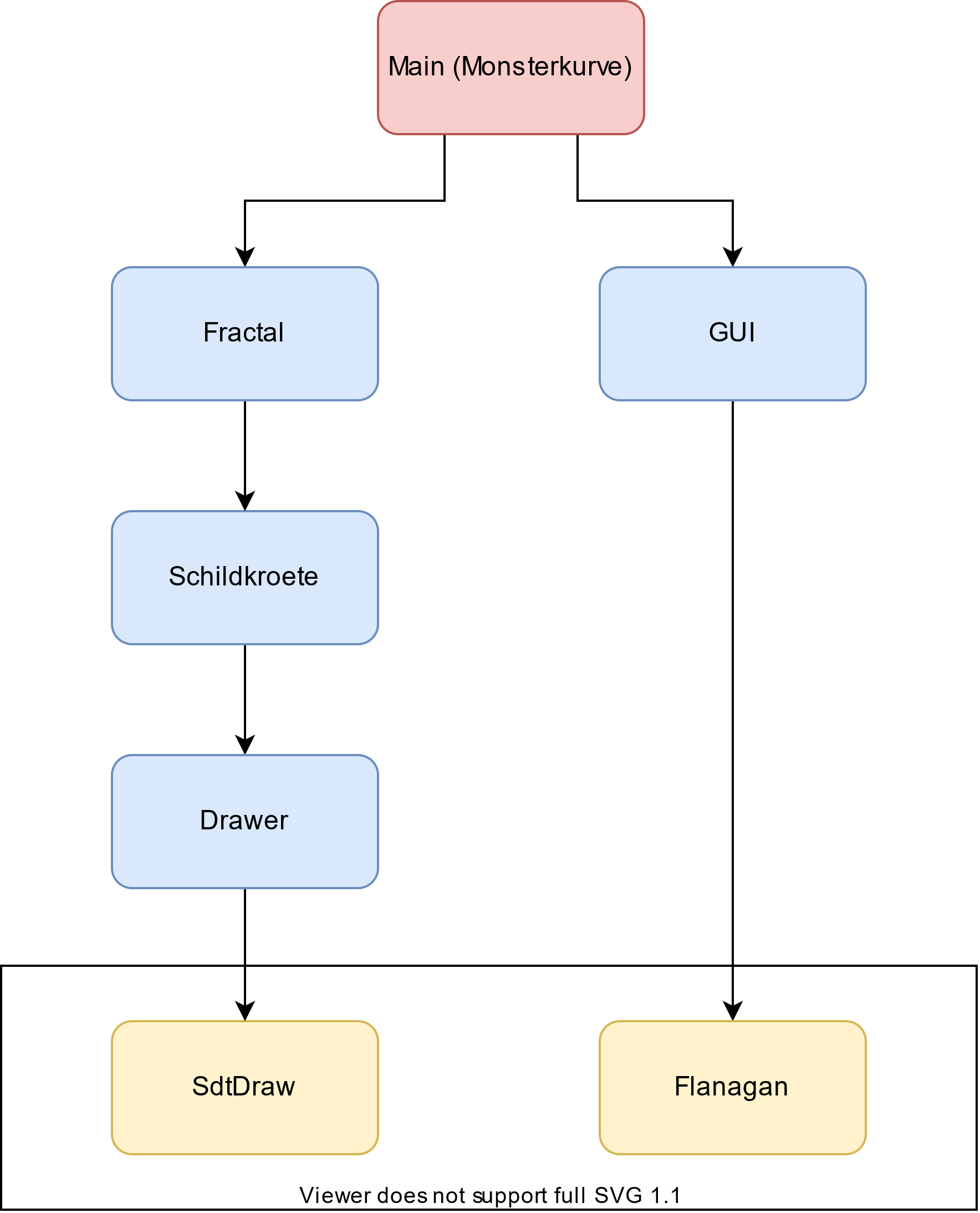
### 2.2.2 Flanagan

Für das Einlesen der gewünschten Kurve, Stufe und Dateiname des zu speichernden Bildes wurde die Bibliothek Flanagan von X implementiert. Mit ihr können einfache Fenster geöffnet werden. Die wichtigsten Methoden für dieses Projekt sind *optionBox()*,*readInt()*, *yesNo()* und *readLine()*.

## 2.3 Reduktion des Problems

Komplexe Systeme können in kleinere einfachere Aufgaben geteilt und so einfacher gelöst werden. Dieses Aufteilen heisst Modularisierung. Jede einzelne Aufgabe kann für sich gelöst werden.

Dieses Programm habe ich folgendermassen modularisiert:

*Main* erfordert ein Hilfsmodul für das GUI, welches auf die Bibliothek *Flanagan* angewiesen ist. Zudem erfordert *Main* die Fraktale (Schneeflocke, Pfeilspitze, Drachenkurve), die alle das Modul *Schildkroete* verwenden. *Schildkroete* erfordert das Modul *Drawer*, welches unter anderem die Methode *drawLine(Point from, Point to)* bietet. Zu guter Letzt verwendet *Drawer* die Bibliothek *StdDraw*.

**Grob**algorhythmus, Modularisierung

Main, Schildkroete, DrawerAbstr, StdDrawDrawer, Fraktale 1, 2, 3

Erklärung zusammenspiel der Module

2.4 Entwicklung der Module

Ablaufdiagramme, Beschreibung variablen

2.5 Entwicklung der Algorithmen

Allgemeines prinzip der Rekursion in allen 3 Algorithmen

Konkrete impl.

2.5.1 Schneeflocke

2.5.2 Pfeilspitze

2.5.3 Drachenkurve

2.6 Testing

Unit testing?

Mit ganz krummen eingangs werten

3. Resultate

3.1 Bilder

3.2 Effizienz

3.3 Features des Main-Programms? Oder zu einleitung??

4. Diskussion

Flächedrachenkurve

blabla

4.1 Optimierungsmöglichkeiten

mathematischer

4.2 Rückblick auf das Problem

Was für Schwierigkeiten?

Hätte besser angehen können?

5. Nachwort?

6 Quellenverzeichnis

7. Abbildungsverzeichnis

8. Tabellenverzeichnis

9. Quellcodeverzeichnis

10. Anhang

6.3 Glossar?

6.6 Weitere Diagramme?

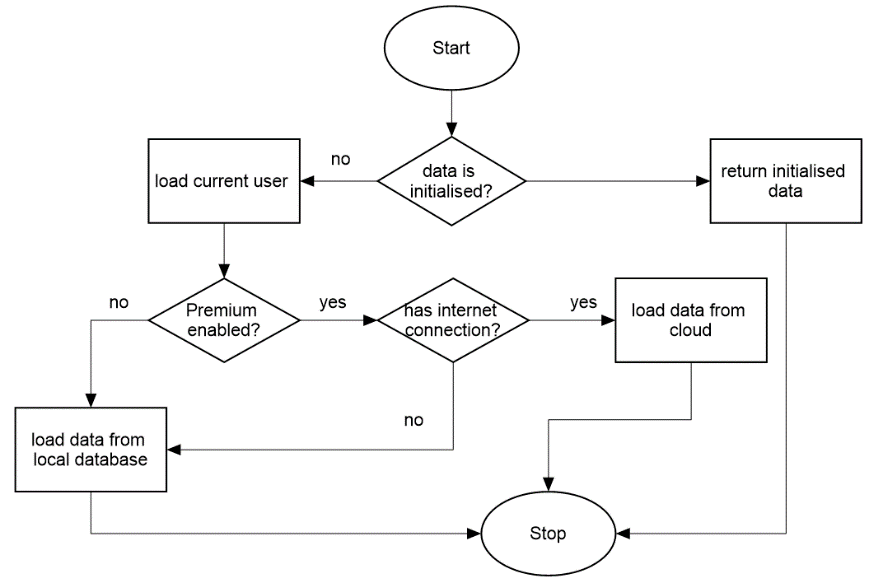
6.7 Gesamter Quellcode

### 3.4.2 Dynamic app

In this step, I implemented the reading, processing, and saving of the information. For example, all the information of a grade is saved, displayed and used to calculate the subject’s average. How the information which the user entered is saved is dealt with in chapter3.8 Connection to databases*.*

When the user switches between two *fragments*, it is very inefficient to load the data again from the storage and convert it, because this uses precious time and slows down the performance of the app. I used a library called *lifecycle-extensions*, which belongs to the *AndroidX* library which contains the class *ViewModel*. The library enables access to the data from the *activity,* as well as all *fragments* hosted within it, from the same *ViewModel* instance. A *ViewModel* instance can contain *LiveData* on which one can set listeners, which get notified when the data in the *ViewModel* changes. The big advantage of this is that and the *activity* and all its *fragments* can use the same data from one source so that they are in sync. Additionally, all the data flow and process can be handled in one place (“ViewModel” class) and the GUI handled in the *activity* or *fragments*. This separation of data and GUI makes it much easier to write and structure the source code. For example, once the data is downloaded, the school year *fragment* simply calls *getSchoolYears()* in the *ViewModel* class. This then returns the school years depending on various factors, which the school year *fragment* class does not have to worry about as shown in Figure 2.

Figure - Flowchart of the loading data process with it’s factors



With this useful option, it was fairly easy to insert dynamic data. When the user updates information through the GUI in the fragments they simply pass it to the shared *ViewModel* instance. It then automatically saves the updated data in the databases and notifies any other fragments which are listening for updates to the data in other parts of the GUI. These then have a chance to update their displayed content.

For example, consider the case of selecting a subject from a list and switching to its detail screen. One member of the *ViewModel* is the *LiveData* *positionSubject*. When a user clicks on a subject in the subject list on the school year screen, the school year *fragment* updates that value accordingly to the list position of this newly clicked subject. With the two navigation libraries mentioned in chapter *3.6.1 Static App,* I then switched to the school subject screen. In the school subject *fragment,* I could display information on the correct clicked subject by requesting *positionSubject* and the other data from the same instance of the *ViewModel* class.

## 3.9 Testing

Testing is essential for building an app, especially if released in the marketplace and being used on customers’ devices. There are different types of testing including developer, regression, usability, acceptance, functional, exploratory and performance.

Of course, I tested my features as I implemented them (developer testing) and at certain points in time did regression tests.

Testing feedback from real users would have to wait till the school year starts and grades are handed out. To avoid serious bugs before officially offering my service there would have to be acceptance tests before this. There would need to be test cases that can be simulated by volunteer testers, including myself. This would involve entering grades and comparing results with expectations in a proper testing session, instead of for real, with actual subject, grades, etc. during the year.

One current plan for usability testing is to let students use my app for about one to two months. After that time, I will ask them personally about the app and to fill out a small questionnaire, which is in the appendix. Through this, small bugs are more likely to found should they exist.

I also planned on executing performance tests, which test the app’s limits and capacities. For example, one can test how much data it can handle, or how fast the server connection is, etc.

# 4. Results

## 4.4 Source code

In summary, I have a total of 155 Java and 249 XML self-written files in my project. There are more than 22’000 lines of Java source code and more than 10’700 lines of XML code.

The complete source code is on the accompanying the CD, password protected by the author.

In the project I wrote helper classes and functions which implement the core business logic of my app so that the GUI handling code is cleanly separated from the logic code. This reuse makes code more easily changeable as well as being smaller thus the app has less storage space on the device.

As an example, the following listings show how the average for a subject is calculated and shown on the subject details screen.

**public class** SubjectFragment **extends** Fragment ...{

...

@Override  
**public void** onViewCreated(@NonNull View view, @Nullable Bundle savedInstanceState) {  
 **super**.onViewCreated(view, savedInstanceState);  
 **...**  
 **viewModel** = ViewModelProviders.*of*(getActivity()).get(SharedViewModel.**class**);  
  
 **viewModel**.getYears(getContext()).observe(getViewLifecycleOwner(), **new** Observer<ArrayList<SchoolYear>>() {  
 @Override  
 **public void** onChanged(ArrayList<SchoolYear> years) {  
 setupInfoCardView();  
 ...

}  
 });

...  
}

...

**private void** setupInfoCardView() {  
TextView tv\_Info1 = getView().findViewById(R.id.***tvInfo1***);  
 ...  
 **if** (**subject**.hasValue()) {  
 **float** avr = **subject**.getAverage(subjectRounding);  
 tv\_Info1.setText(*formatFloat*(avr));  
 } **else** {  
 tv\_Info1.setText(R.string.***nothingToShow***);  
 }  
 **...**

}

}

Listing - Attaching listener to SubjectFragment and displaying the subject's average

Listing 2 shows how our listener is attached to the *ViewModel* instance. The implementation uses the *SchoolSubject*’s *getAverage()* function to implement the business logic.

**public class** SchoolSubject **extends** SchoolItem {

...

**public float** getAverage(**float** SubjectsRounding) {  
 **if** (**gradeOrGroupContainers** != **null** && **gradeOrGroupContainers**.size() > 0) {  
 **float** sumGrades = getSumGradeTimesWeight(), sumWeight = getSumWeight();  
 **float** mAverage;  
 **if** (sumWeight != 0) {  
 mAverage = sumGrades / sumWeight;  
 } **else** {  
 mAverage = 0;  
 }  
 **if** (SubjectsRounding > ***NO\_ROUNDING***) {  
 **return** Math.*round*(mAverage / SubjectsRounding) \* SubjectsRounding;  
  
 } **else** {  
 **return** mAverage;  
 }  
 } **else** {  
 **return** 0;  
 }  
}

...

}

Listing 3 - The function "getAverage()" in "Schoolsubject"

Listing 3 shows that the function *getAverage()* delegates some of the calculation to the *getSumGradeTimesWeight()* function.

**public class** SchoolSubject **extends** SchoolItem {

...

**public float** getSumGradeTimesWeight() {  
 **float** sum = 0;  
 **for** (GradeOrGroupContainer gradeOrGroupContainer : **gradeOrGroupContainers**) {  
 **if** (gradeOrGroupContainer != **null** && gradeOrGroupContainer.getSchoolGradeArrayList().size() > 0) {  
 sum += gradeOrGroupContainer.getItemValue() \* gradeOrGroupContainer.getWeight();  
 }  
 }  
 **return** sum;  
}

...

}

Listing 4 – The function “getSumGradeTimesWeight()” in “SchoolSubject”

Listing 4 shows that the function *getSumGradeTimesWeight()* uses the abstract function *getItemValue()* of the class *GradeOrGroupContainer,* which, just like *SchoolSubject* extends the abstract class *SchoolItem* as shown in Listing 5.

**public class** GradeOrGroupContainer **extends** SchoolItem {

**private** SchoolGrade **grade**;  
**private** SchoolGroup **group**;

...

@Override  
**public float** getItemValue() {  
 **if** (isGroup()) **return group**.getItemValue();  
 **return grade**.getItemValue();  
}

...

}

Listing 5 - The overridden abstract function "getItemValue()" in "GradeOrGroupContainer"

**public class** SchoolGroup **extends** SchoolItem {

...

@Override  
**public float** getItemValue() {  
 **return** getAverage();  
}

...

**public float** getAverage() {  
 **if** (**schoolGradeArrayList**.size() != 0) {  
 **float** sumGrades = getSumGradeTimesWeight(), sumWeight = getSumWeight();  
 **if** (sumWeight != 0) {  
 **return** sumGrades / sumWeight;  
 } **else** {  
 **return** 0;  
 }  
 } **else** {  
 **return** 0;  
 }  
}

...

}

Listing 6 – The implementation of “SchoolGroup.getItemValue()”

To complete the journey from the *Fragment* to the contents of a group we can see that the *SchoolGroup’*s *getItemValue()* function delegates to *getAverage()* which calculates the actual value of the school group average as shown in Listing 6.

## 4.4 Comparison to products on the market

I compared my app to some GMAs on the market to see if my app could keep up with them. Those on the market I had a look at were the Android versions (on Play Store) of the GMAs the students use according to my: *Pluspoints* (Junker 2019), *School Marks Manager* (Sunisup Studios 2017), *Delta+* (Berisha 2018), *Studool* (Jeddsan 2018), *Notenverwaltung Schweiz* (Widmer 2019), *Notenverwaltung (Schweiz)* (Gyger 2019).

The most-used app by far by participants of my survey is Pluspoints, with over 250’000 downloads on iOS (Junker 2019). Therefore it makes to focus on the comparison with that app. Here are the main points (the detailed comparison with all apps is below):

1. Pluspoints has a very similar **navigation** pattern to my app. The logic is the same, however, Pluspoints has fewer views per screen, making it clear. Both have a simple **design**, where Pluspoints uses two main colours (orange and white) and I used four (see 4.3.1 Design)
2. Pluspoints has a **feature** I do not yet support: Importing and exporting the data as a file
3. My app has several **features** which Pluspoints does not support: from chapter 4.2 Capabilities features l), m), n), p), q), and custom rearranging of grades and subjects (in detailed list of features). There are also other small features my app supports which Pluspoints does not.
4. Feature m) (online saving) cost in my app, all the other features are for free, Pluspoints is entirely for free, but does not support my premium feature.

In summary, one can say Pluspoints is easier to use and offers a useful, simple, free service, which probably is enough for many students. However, my app has a lot more capabilities to offer, which many students should find useful. My app follows a similar navigation to Pluspoints. My premium online backup support should compensate for my currently missing import and export of the data option. If I can manage to improve my design to the same quality of Pluspoints and clean up the GUI step by step while keeping the features, my app should be able to compete well with it or perhaps even be a better market product.

Not only does Pluspoints not support some of my app’s practical capabilities, but neither the other GMAs. If they included one or two of my features, they often lacked other ones as shown in Table 1 and

Table 2.

|  |  |
| --- | --- |
| **Feature** | **Feature No** |
| create years, subjects and enter grades | 1 |
| Import/Export data as file | 2 |
| calculate average of years and subjects | 3 |
| calculate PlusPoints | 4 |
| enter subgrades | 5 |
| calculate required grade to achieve target | 6 |
| choose subject rounding | 7 |
| FAQ (help for confused users) | 8 |
| sort grades by something | 9 |
| sort subjects by something | 10 |
| colored average settings | 11 |
| weighting of subject | 12 |
| notes to a grade | 13 |
| enter link of menu school menu plan | 14 |
| custom school grade system | 15 |
| colored subjects | 16 |
| online storing | 17 |
| grades on graph | 18 |
| subjects on graph | 19 |
| show trend and average over time in graph | 20 |
| images of exam to grade | 21 |
| mark years, subjects, grade groups, grades and images as favourites | 22 |
| scanning and viewing timetable | 23 |
| special stats on years | 24 |
| special stats on subjects | 25 |
|  |  |

Table 1 - index of features

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Feature No** | **My app** |  | Pluspoints | MarkManager | Delta+ | Studool | Noten | | Noten Rechner | Notenverwaltung | |
| 1 | free |  | free | free | free | free | free | free | | | free |
| 2 | no |  | free | no | no | no | no | | no | free | |
| 3 | free |  | free | free | free | free | free | | free | free | |
| 4 | free |  | free | free | no | no | free | | free | no | |
| 5 | free |  | free | free | free | no | no | | no | no | |
| 6 | free |  | free | free | free | no | no | | free | free | |
| 7 | free |  | free | premium | free | no | no | | free | no | |
| 8 | no |  | free | no | no | no | no | | no | no | |
| 9 | free |  | no | premium | no | no | no | | no | no | |
| 10 | free |  | no | premium | no | no | no | | no | no | |
| 11 | free |  | no | free | no | no | no | | no | no | |
| 12 | free |  | free | free | free | no | free | | free | free | |
| 13 | free |  | no | free | free | no | free | | no | free | |
| 14 | no |  | no | no | free | no | no | | no | no | |
| 15 | free |  | no | no | free | no | no | | Only german, swiss or austrian | no | |
| 16 | no |  | no | no | no | free | no | | no | no | |
| 17 | premium |  | no | premium | no | free | no | | no | no | |
| 18 | free |  | no | no | no | free | no | | no | no | |
| 19 | free |  | no | no | no | no | no | | no | no | |
| 20 | free |  | no | no | no | no | no | | no | no | |
| 21 | free |  | no | no | no | no | no | | no | free | |
| 22 | free |  | no | no | no | no | no | | no | no | |
| 23 | no |  | no | no | no | no | no | | no | no | |
| 24 | free |  | no | no | no | no | no | | no | no | |
| 25 | free |  | no | no | no | no | no | | no | no | |

Table 2 – Support of features of GMAs

# 5. Discussion

## 5.1 Further development

Overall, I am very pleased with my first app but I will try to improve it as much as possible, which can include restructuring and improving its navigation, removing or adding features, increasing the efficiency, improving the design and keeping up-to-date with the newest design models. Of course, I will continuously do this step once my app is launched, but also before. To get a better idea of what to improve, I will research and monitor the usage of my app by letting test subjects use it for two or three months and asking them about their experience. Possibly they will discover bugs I did not, or notice that there are features they never used which only took up precious space on the screen. I also have some features in mind which I will implement in the future. One is the option of importing or exporting a file containing school years, so that also none-premium users can keep their data if they, for example, buy a new phone. I will also make it available in German and perhaps French.

Currently, my app is an Android application, only available to Android and not iOS users. That is why I will also program a corresponding Apple version. This process should be much faster than what I did so far, as I simply have to copy the features and design. I even have the general source code logic.

In the near future, I will launch it on Play Store, for which I will create a video that explains all the practical things about my app. Additionally, I will try to spread my app as widely possible by writing emails to schools in which I ask them to hang up a small poster in the entrance or ask teachers to recommend this app. I will try to do this in many schools spread over Switzerland as well as other countries of which my app supports the language. Possibly I will continue in a similar fashion of the survey I did on other GMAs but this time with my app. I will promise them the premium version of my app for free if they give constructive feedback on it. The importance at the start is that I get as many users as possible instead of trying to make a lot of money. Otherwise, my app will not spread as fast and in the long run, this will decrease the success and income.

## 5.2 What could I have done differently?

My work process had a good structure and order to it which made the project easier to handle. However, I think I could have built in a few more steps and been more strategic about handling this project. For one thing, I should have let my static version of the app be tested before I continued with the dynamic data so that if something is not good about the navigation or GUI, I could have changed it before I implemented the dynamic data. There are several ways one can do this: For example, one can set up a list of tasks beforehand, which test subjects have to fulfill in the app. An example could be *Add a grade called “My grade” to the subject “Maths”. Then add an image of yourself to the grade*. While they try to complete those tasks one can watch them and if they take a long time trying to solve it, then it is obvious something must be improved. This method was described by Hoekman (Hoekman 2019, p. 104-105). In the beginning, I should have had a look at more apps with good design. This way, I would have known more ways of solving a problem elegantly, because quite often I found some excellent examples by accident which were, however, unfortunately, too late to implement. Sometimes I managed to do so but, other times I would have had to completely change the structure of my app, which at that stage was almost impossible.

Apart from the above, I think my steps were in the right order and were all useful. If I were to create another app (I have a few concrete ideas), I would follow the same process.

# 6. Afterword and acknowledgements

I did not expect this project to be so time-consuming and energy-consuming, but I also had fun while working on my app. I especially enjoyed the creative aspect, thinking of features, but also coding and designing were great. Sometimes I got lost and spent too much time on small things and testing out different ways of displaying something. It was very frustrating when the app did not work as expected and I could not find the mistake. Sometimes I spent several hours searching for the solution to an issue which only played a minor role. Also, when I knew exactly how to solve something (for example, when I had the program code in my head) and it only had to be done, it was quite boring, because it was simply executing a certain task. I reached this situation several times while writing the program code. However, apart from those tasks, I generally really enjoyed working on this project and I learned a lot from it. It was a great experience where I quickly realized when I had done something wrong. My understanding of applications, programming and databases has grown enormously. But I also learned general things like how much effort, seemingly effortless things can be, and that a good plan from the beginning is worth a lot.

Of course, without the help of others, this would not have been possible. First of all, I would like to thank my supervisor Victor Yakhontov for guiding me through this whole phase as well as explaining things about Java. Aside from the knowledge and experience he shared with me, he always took his time, so that sometimes when we said goodbye we would then somehow talk on for another hour. Also, great thanks to my co-supervisor Thomas Strub, who also helped with the planning and gave me ideas for my app.

To Tim Isler, Alejandro Sanchez, and my sister Sinéad Sebright: thank you for sharing your thoughts, ideas, and constructive feedback about the application. You made me see things I never hesitated to tell me if you thought something had to be improved.

To my father Simon Sebright: thank you for helping out when I got stuck, especially when the app had a bug and I had spent a lot of time trying to find the solution so that we found it together faster than when the two of us had done it separately. He also recommended an excellent book for app development called “Designing the Obvious”.

Last but not least, rather most of all, thank you to all these students and others who filled in my questionnaire so seriously and with thought. Without you, I could not have created an application like it is now.

# 7. References

1. [Android Developers](https://developer.android.com/). (2019). Application Fundamentals. Available under <https://developer.android.com/guide/components/fundamentals.html> (last retrieved 11/10/2019)
2. [Android Developers](https://developer.android.com/). (2019). Fragments. Available under <https://developer.android.com/guide/components/fragments> (last retrieved 11/10/2019)
3. Berisha, Pjeter: Delta+. Notenapp für Schüler, in : Google Play, version 2.0, release date – 12/01/2018. <https://play.google.com/store/apps/details?id=com.app.notenberechnungs_app&hl=en>, downloaded on 27/09/2019
4. Collins, Charlie/Galpin, Michael/Käppeler, Mathias: Android in Practice, Shelter Island, NY: Manning Publications Co, 2012.
5. Firebase. (2019). Cloud Firestore Data Model. Available under <https://firebase.google.com/docs/firestore/data-model> (last retrieved 11/10/2019)
6. Firebase. (2019). Cloud Storage. Available under <https://firebase.google.com/docs/storage> (last retrieved 11/10/2019)
7. Firebase. (2019). Firebase Authentication. Available under <https://firebase.google.com/docs/auth> (last retrieved 11/10/2019)
8. Firebase. (2019). Firebase Crashlytics. Available under <https://firebase.google.com/docs/crashlytics> (last retrieved 11/10/2019)
9. Firebase. (2019). Get started with Cloud Firestore Security Rules. Available under <https://firebase.google.com/docs/firestore/security/get-started> (last retrieved 11/10/2019)
10. [Firebase](https://developer.android.com/). (2019). Privacy and Security in Firebase. Available under <https://firebase.google.com/support/privacy> (last retrieved 11/10/2019)
11. Firebase Open Source. (2019). Firebase Authentication Password Hashing. Available under <https://firebaseopensource.com/projects/firebase/scrypt/> (last retrieved 11/10/2019)
12. Goldberg, Kevin Howard: XML. Visual QuickStart Guide. Berkeley, CA : Peachpit Press, 2009 (2nd edition).
13. Google: gson. gson-parent-2.8.5, in: Github/Google, version 2.8.5, release date – 22/05/2018. <https://github.com/google/gson/releases/tag/gson-parent-2.8.5>, downloaded on 27/07/2019
14. Google: AndroidX, in: Android Developers, release date – 18/09/2019. <https://developer.android.com/jetpack/androidx>, downloaded on 18/09/2019.
15. [Google](https://developer.android.com/). (2019). Material Design. Build beautiful, usable products faster. Material Design is an adaptable system—backed by open-source code—that helps teams build high quality digital experiences. Available under [https://material.io](https://material.io/) (last retrieved 11/10/2019)
16. Google/JetBrains: Android Studio. 3.5, in: Android Developers, release date – 20/08/2019. <https://developer.android.com/studio>, downloaded on 20/8/2019
17. Gyger, Gian: Notenverwaltung (Schweiz), in : Google Play, version 1.3, release date – 16/08/2019. <https://play.google.com/store/apps/details?id=com.giangyger.notenverwaltungschweiz&hl=en>, downloaded on 27/09/2019
18. Hoekman, Robert. Designing the Obvious. A Common Sense Approach to Web Application Design. Berkeley, CA: New Riders, 2007.
19. Jeddsan: Studool. Grade management, in : Google Play, version 1.2.1-12, release date – 08/02/2018. <https://play.google.com/store/apps/details?id=net.jeddsan.studool&hl=en>, downloaded on 27/09/2019
20. Junker, Philip: PlusPoints. Manage your grades, know your average, in : Google Play, version 1.1.8, release date – 21/09/2019. <https://play.google.com/store/apps/details?id=ch.fidelisfactory.pluspoints&hl=en>, downloaded on 27/09/2019
21. Sunisup Studios: School Marks Manager, in : Google Play, version 1.2.4, release date – 13/06/2017. <https://play.google.com/store/apps/details?id=com.noten.johannes.sunisup&hl=en>, downloaded on 27/09/2019
22. Ullenboom, Christian: Java ist eine insel. Das umfassende Handbuch. Bonn: Rheinwerk Computing, 2012 (10th edition).
23. Widmer, Jan: Notenverwaltung Schweiz, in : Google Play, version 1.0.20, release date – 11/06/2019. <https://play.google.com/store/apps/details?id=com.janmer.notenverwaltung&hl=en><https://play.google.com/store/apps/details?id=ch.fidelisfactory.pluspoints&hl=en>, downloaded on 27/09/2019

# 8. Figure index

[Figure 1 - Task flow diagram of user editing grade and viewing subjects average 7](file:///C:\Users\Jonah\Documents\Jonah\Schule\MaturDoku\Doku%20versionen\Unique%20Grade%20Manager%20Application.docx#_Toc21780044)

[Figure 2 - Flowchart of the loading data process with it’s factors 9](file:///C:\Users\Jonah\Documents\Jonah\Schule\MaturDoku\Doku%20versionen\Unique%20Grade%20Manager%20Application.docx#_Toc21780045)

[Figure 3 - Data structure of the school content 10](file:///C:\Users\Jonah\Documents\Jonah\Schule\MaturDoku\Doku%20versionen\Unique%20Grade%20Manager%20Application.docx#_Toc21780046)

[Figure 4 - Representation of the Cloud Firestore datamodel. Firebase: Datamodel, 01/10/2019 11](file:///C:\Users\Jonah\Documents\Jonah\Schule\MaturDoku\Doku%20versionen\Unique%20Grade%20Manager%20Application.docx#_Toc21780047)

[Figure 5 - Dataflow between the app and local storage with help of “gson” and “SharedPreferences” 12](file:///C:\Users\Jonah\Documents\Jonah\Schule\MaturDoku\Doku%20versionen\Unique%20Grade%20Manager%20Application.docx#_Toc21780048)

[Figure 6 - Year details screen and home screen 16](file:///C:\Users\Jonah\Documents\Jonah\Schule\MaturDoku\Doku%20versionen\Unique%20Grade%20Manager%20Application.docx#_Toc21780049)

[Figure 7 - New grade screen 16](file:///C:\Users\Jonah\Documents\Jonah\Schule\MaturDoku\Doku%20versionen\Unique%20Grade%20Manager%20Application.docx#_Toc21780050)

[Figure 8 - Year details screen with grades 17](file:///C:\Users\Jonah\Documents\Jonah\Schule\MaturDoku\Doku%20versionen\Unique%20Grade%20Manager%20Application.docx#_Toc21780051)

[Figure 9 - School year stats/analytics screen (top only) 17](file:///C:\Users\Jonah\Documents\Jonah\Schule\MaturDoku\Doku%20versionen\Unique%20Grade%20Manager%20Application.docx#_Toc21780052)

[Figure 10 - Subject stats screen with graph of grades, the average over time and the trend of grades (bottom only) 18](file:///C:\Users\Jonah\Documents\Jonah\Schule\MaturDoku\Doku%20versionen\Unique%20Grade%20Manager%20Application.docx#_Toc21780053)

[Figure 11 - Subject details screen with grade group (expanded) 18](file:///C:\Users\Jonah\Documents\Jonah\Schule\MaturDoku\Doku%20versionen\Unique%20Grade%20Manager%20Application.docx#_Toc21780054)

[Figure 12 - Grade details screen with an added image 19](file:///C:\Users\Jonah\Documents\Jonah\Schule\MaturDoku\Doku%20versionen\Unique%20Grade%20Manager%20Application.docx#_Toc21780055)

[Figure 13 - The classes for the complete data structure 31](#_Toc21780056)

# 9. Listing index

[Listing 1 - The function "isGroup()" in "GradeOrGroupContainer" 10](file:///C:\Users\Jonah\Documents\Jonah\Schule\MaturDoku\Doku%20versionen\Unique%20Grade%20Manager%20Application.docx#_Toc21772380)

[Listing 2 - Attaching listener to SubjectFragment and displaying the subject's average 20](file:///C:\Users\Jonah\Documents\Jonah\Schule\MaturDoku\Doku%20versionen\Unique%20Grade%20Manager%20Application.docx#_Toc21772381)

[Listing 3 - The function "getAverage()" in "Schoolsubject" 21](#_Toc21772382)

[Listing 4 – The function “getSumGradeTimesWeight()” in “SchoolSubject” 21](#_Toc21772383)

[Listing 5 - The overridden abstract function "getItemValue()" in "GradeOrGroupContainer" 22](#_Toc21772384)

[Listing 6 – The implementation of “SchoolGroup.getItemValue()” 22](#_Toc21772385)

# 10. Table index

[Table 1 - index of features 24](#_Toc21772386)

[Table 2 – Support of features of GMAs 25](#_Toc21772387)

# 11. Appendix

## 11.1 Personas

**Matt** is an ordinary 14-year-old schoolboy - **75%** of users are like this

He simply wants to keep track of his average in specific subjects and the overall average. He doesn't want to spend a lot of time adding the grades and when he opens the app, he wants to quickly view his averages and perhaps a few other stats like best grade or number of grades. He is not very interested in his school grades and therefore only needs few and simple features. Every now and then he forgets what the first lesson in the morning is, so he would like to open the app and quickly view his scanned timetable. He hates seeing overcrowded screens when he only wants to achieve something simple.

**Emma** is a good student at high school - **10%** of users are like this

She has good results and loves looking at old averages and comparing subjects or years. She doesn't mind learning new features as long as they make analysing more fun. She often uses a tablet to increase the size of tables or graphs, making analysing more enjoyable.

**Lukas** is not a good student who barely manages to pass the year at high school – **15%** of users are like this

He has problems passing the year and therefore needs precise data on his year and subjects, etc. He also wants to know what grades he needs in which subjects and what he should rather focus on in order to pass. He is very forgetful and just does not want to learn for exams or do his homework.

## 11.2 Questionnaire about GMAs

The questionnaire can be seen in PDF-format on the accompanying CD. It is currently available online at <https://docs.google.com/forms/d/e/1FAIpQLSd3qTP6jdte8Y4e5_3LWj8Xs60SacX_stt70o8lZSbeVGHvoQ/viewform>.

## 11.3 Questionnaire about my app

As in 5.1 Further development, my app be will tested and information gathered with the help of a questionnaire. This can be seen in PDF-format on the accompanying CD.

## 11.4 Libraries

As well as the firebase and the *AndroidX* libraries, mentioned in detail in this document, I also used the following libraries, which are explained in more detail on the accompanying CD. Here are the names as used android library system:

*Material*, *PhotoView*, *mask-eddittext* and *MPAndroidChart*.

## 12.5 Complete data structure

The complete SCR data with all it classes and members is shown below in Figure 13.



Figure 13 - The classes for the complete data structure

## 11.6 Further information on the accompanying CD

The following items can be found on the CD which accompanies this Maturaarbeit:

1. Source code (password protected by the author)
2. Further screenshots of the app
3. Detailed list of features
4. Formulae used for calculations
5. Questionnaire about GMAs
6. Questionnaire about my app for testing feedback
7. Details of the libraries used in the project