Studienarbeit

Reverse Polish Notation Tile Calculator

Teilprüfungsleistung in WIP

Erstellt von dem Team 1 "Das Proletariat":

Tom Bockhorn
2715438
Mülheimer Straße 274
51469 Bergisch Gladbach

Hendrik Falk
2715450
An der Josefshöhe 33
53117 Bonn

Dennis Gentges
2715460
Zum Bahnert 22
50189 Elsdorf

Getuart Istogu
2715526
Gerberstr. 3
51688 Wipperfürth

Jannis Luca Keienburg
2715548
Ruthe Furth 4
51515 Kürten

Tim Jonas Meinerzhagen
2715581
Kamper Weg 1
51519 Odenthal

Khang Pham
2715614
Vereinsstr. 15
51379 Leverkusen

Tim Schwenke
2715670
Mülheimer Straße 274
51469 Bergisch Gladbach

Prüfer: Prof. Dr. Thomas Seifert

Eingereicht am: 3. Februar 2020

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Das Team mit Namen und Bild

Abbildung 1: Gruppenfoto

Abbildung 2: Tom Bockhorn

Abbildung 3: Hendrik Falk

Abbildung 4: Dennis Gentges

Abbildung 5: Getuart Istogu

Abbildung 6: Jannis Keienburg

Abbildung 7: Tim Meinerzhagen

Abbildung 8: Khang Pham

Abbildung 9: Tim Schwenke

2 Ziel des Projekts

Dies ist eine Vorlage zum Erstellen von Bachelorarbeiten an der FHDW mit dem Schriftsatzsystem LaTeX.

Die in der Vorlage verwendeten Pakete und Styles sind sehr gut dokumentiert. Bei der Beprechung der einzelnen Paket wird auf die jeweilige Dokumentation verwiesen, die standardmäßig mit den jeweiligen Paketen installiert wird.

3 Projektplanung

3.1 Beschreibung des Funktionsumfangs

3.2 Projektablaufplan

3.3 Planung der Software

3.3.1 Planung des Mockups

3.3.2 Planung der Datenstrukturen und Schnittstellen

Nutzung von Stack für Notation [Schwenke]

Der Taschenrechner soll als Eingabelogik für die Anwendung von Operationen die umgekehrte polnische Notation verwenden. Hierbei werden immer zunächst die Operanden und im Anschluss daran die darauf auszuführenden Operatoren angegeben. Dieser Ansatz ermöglicht eine stapelbasierte Abarbeitung.

Stacks werden, wie von den meisten Programmiersprachen, auch in Java in der Standardbibliothek unterstützt. Mit dabei sind Methoden wie `push` (für das Ablegen eines Objekts auf dem Stapel), `pop` (für das Entfernen und die Wiedergabe eines Objekts auf dem Stapel), `peek` (für die Wiedergabe ohne Entfernen eines Objekts auf dem Stapel) und `empty` (für das Leeren des Stapels).

Jedoch müssen hierbei die besonderen Anforderungen des Taschenrechners beachtet werden. Operanden können von gänzlich unterschiedlichem Typus sein, zum Beispiel eine einfache Dezimalzahl oder auch ein Tupel, und viele Operationen benötigen mehr als die ersten (maximal zwei) Operanden auf dem Stack. Möchte man Elemente vom Stapel entfernen, kann man `pop` mehrmals aufrufen. Aufwändiger hingegen wird es bei `peek`. Möchte man mehrere Elemente vom Stapel einsehen ohne diese zu entfernen, muss man bei der Arbeit mit dem vorhandenen Stack einen weiteren bereithalten, nur um zwischengespeicherte Elemente lagern zu können. Anders ist es nicht möglich `peek` auf mehrere Elemente gleichzeitig anzuwenden. Gerade das ist aber bei der App notwendig. Weitere Methoden, die bei der umgekehrten polnischen Notation oft benötigt werden, aber nicht implementiert sind, sind `reverse` (für die Vertauschung der ersten zwei Elemente auf dem Stack, was wichtig für nicht-kommutative Operationen ist), `rollUp` (das unterste Elemente wird an den ersten Platz geschoben, das erste Element an den zweiten Platz usw.) und `rollDown` (das unterste Elemente wird an den ersten Platz

geschoben, das erste Element an den zweiten Platz usw.).

Aufgrund dessen soll für dieses Projekt ein eigener Stapel implementiert werden. Dieser soll die zuvor genannten Funktionen mit unterschiedlichen Parametertypen unterstützen. Dabei ist darauf zu achten, dass die Programmierung generisch erfolgt und das Stack nicht nur alle Typen von Operanden unterstützt, sondern auch für gänzlich andere Klassenbäume in der App verwendet werden kann.

Ansatz der Kalkulationsorchestrierung [Schwenke]

Die App soll den Umgang mit unterschiedlichen Operanden-Typen beherrschen. Die Addition zweier Matrizen funktioniert anders als die Addition von zwei einfachen Dezimalzahlen. Java verfügt nativ weder über die entsprechenden Operanden noch über die Methoden für die Kalkulation. Auch die ausgewählte Bibliothek ist nicht ohne weiteres in der Lage Operationen auf alle Kombinationen von Operanden im folgenden Format einheitlich anzuwenden:

Listing 1: Konzept für Nutzung generischer Schnittstelle

```
Operation.mit(matrixOperand, dezimalOperand, dezimalOperand)
```

Einheitlichkeit ist notwendig, damit im Frontend der Applikation keine Logik vorhanden sein muss, die entscheidet wie genau (auf Basis der Operanden-Typen) eine Operation umgesetzt wird. Deswegen muss eine einfache Schnittstelle entwickelt werden, die für den Nutzer nur zwei Drehschrauben bereitstellt. Dies ist zunächst die Auswahl der gewünschten Operation. Das kann z.B. das Symbol `+` als übliches Zeichen für Addition sein. Anschließend wird eine Reihe von Operanden übergeben. Dieser Aufruf sollte schließlich das Ergebnis in Form eines Operanden zurückgeben. Im Fall der Addition einer Matrix mit einer rationalen Zahl wäre dies wiederrum eine Matrix. Die korrekte Kalkulation soll also dynamisch bestimmt werden. Wichtig zu klären ist hier auch das Verhalten im Falle eines Fehlschlags. Nicht alle Kombinationen von Operanden können unterstützt werden. Die Verwendung von *Optionals* (ein `Optional` ist ein Objekt, das man sich als Datenbehälter vorstellen kann, der entweder einen Wert enthält oder leer – aber nicht `null` sein kann) bietet sich hier zwar an, wird jedoch von Java in der verwendeten Android API-Version nicht unterstützt. Deswegen ist hier geplant sogenannte *checked Exceptions* zu verwenden. Diese müssen bei der Verwendung explizit aufgefangen und weiterverarbeitet werden. Die Abbildung einer Operanden-Kombination auf die

entsprechende konkrete Kalkulationsmethode muss dementsprechend zur Laufzeit des Programms erfolgen. Ein solches Mapping ist in Java nur mithilfe des Reflection-Pakets möglich. Reflektion ermöglicht den Einblick in ein Objekt (neben der Nutzung des Punkt-Operators) in eine Klasse. Zum Beispiel kann man eine Methode anhand einer Kombination von Parametertypen finden und aufrufen. Es ist geplant diesen Ansatz für die Orchestrierung der Kalkulationen in der App zu verwenden. Auch ist es nicht notwendig nur eine vordefinierte Anzahl an Argumente anzunehmen. So kann es sinnvoll sein, dass eine Methode zur Erstellung eines Tupels eine beliebige Anzahl an Operanden annimmt. Auch das lässt sich mit Reflektion umsetzen.

Der große Vorteil dabei ist, dass nirgendwo explizit in einer Abfrage entschieden werden muss, welche Kombination von Operanden an welche Methode weitergeleitet werden soll. Die Zuordnung erfolgt rein über die Deklaration der Parametertypen in der Methode selbst. Das macht das Ändern und Erweitern der Rechenfunktionalitäten einfach. Es muss lediglich die entsprechende Klasse herausgesucht und eine Methode im korrekten Format hinzugefügt werden.

Zu entscheiden ist ebenfalls, ob das Gros der Rechenmethoden innerhalb der jeweiligen Operanden-Klassen oder dedizierten Klassen für die Kalkulation angesiedelt sind. Die erste Option hat neben der stärkeren Objektorientierung den Vorteil, dass immer klar ist, dass eine Methode mit den übergebenen Argumenten auf dem jeweiligen Objekt ausgeführt wird. Andererseits erhöht dies die Komplexität der Operanden-Klassen deutlich. Unterstützt man wie geplant 5 bis 7 dedizierte Typen von Operanden und 10 Kalkulationsarten, muss jede Klasse potenziell dutzende Methoden für die Rechnung enthalten. Die andere, und bevorzugte Option, ist die Auslagerung der Kalkulationsmethoden in eigenständige Klassen. Dies reduziert zwar nicht die Anzahl benötigter Methoden, isoliert die Rechenlogik jedoch in Klassen. Innerhalb dieser Klassen wird prozedural programmiert. Eine typische Charakteristik von Objekten und deren Methoden ist *Mutability*. Eine Methode bekommt ein Objekt und kann dieses verändern. Dies kann Testen unter Umständen aufwändiger gestalten. Durch Isolierung der Rechnungen in eigenen Klassen kann hingegen sichergestellt werden, dass jede Methode *immutable*, also unveränderlich, ist. Das macht das Schreiben von Tests einfach. In Java kann Immutability durch die Verwendung von Annotationen sichergestellt werden.

3.3.3 Planung der Activities und Layouts

3.3.4 Planung der Navigation zwischen den Activities

3.4 Geplante Aufgabenverteilung im Team (tabellarisch)

4 Beschreibung des Projektverlaufs

4.1 Tatsächliche Aufgabenverteilung im Team (tabellarisch)

4.2 Teammeeting-Protokolle

03.09.2019 :: 200 Minuten

- Auswahl des Projekttyps. Das Team hat sich für die Entwicklung einer Android App entschieden.
- Erste Einarbeitung in die Thematik. Lesen des bereitgestellten Dokuments mit Aufgabenstellung, groben Anforderungen und weiteres.
- Konzepterarbeitung auf Papier. Vorstellung und Diskussion verschiedener Ansätze.

04.09.2019 :: 110 Minuten

- Sich mit Herr Seifert auf einen Ansatz für den Taschenrechner einigen.
- Workflow für Git, Meeting-Protokolle, Studienarbeit und Projekttagebücher festlegen.
- Ausarbeitung des Konzepts für den Taschenrechner. Hier wurden dem Auftraggeber Herr Seifert mehrere Konzepte vorgestellt und gemeinsam mit ihm genaue Anforderungen erarbeitet.

05.09.2019 :: 20 Minuten

Besprechen der Tagesziele:

- Fertigstellung des Konzeptes.
- Fortschritte beim Paper-Prototypen machen bzw. erstellt haben.
- Android-Umgebung soll bei allen Team-Mitgliedern komplett aufgesetzt und lauffähig sein.

05.09.2019 :: 60 Minuten

Besprechen aktueller Stand des Konzepts und Prototypen:

- Es wurde ein Mid-Fidelity Prototyp erstellt.
- Diskussion über Umsetzung und Workflow der App.
  - Wie sollen die einzelnen Kacheln funktionieren?
  - Wie sollen die Kacheln miteinander interagieren?
  - Wie könnte die Architektur der App aussehen?

05.09.2019 :: 40 Minuten

- Aufbau des Scrumboards mit Backlog, Doing, Review und Done.
  - Backlog: Architektur Basics, Konzept (Paper Prototype, Mid-Fidelity-PPT-Prototyp), Rechenmodule, Zeitmanagement, UI-Coding-Design.
  - Doing: Paper Mid Fidelity
  - Review:
  - Done: Git Projekt aufsetzen
- Übertragen in Teams Planner und dort weitere Ausarbeitung.
- Grobe Verteilung der Einträge im Backlog um Konzepte zu erarbeiten.

17.09.2019 :: 150 Minuten

Backend-Architektur und App-Workflow:

- Erweiterung UML-Klassendiagramm. Die Klasse Operand wird abstrakt und wird von konkreten Operanden wie Vector geerbt. Diese stellen Extensions dar die neben den eigentlichen mathematischen Werten weitere Daten und Verhalten mitbringen.
- Welche Library soll für Mathe-Funktionalitäten benutzt werden? JScience und die bereits mitgelieferte Standardbibliothek.
- Wie sollen Elemente in der GUI dargestellt werden? Als ASCII oder gerendert in LaTeX. Letzteres ist mit höherer Komplexität verbunden sieht aber auch besser aus.

- Wie soll das Layout funktionieren? Gridlayout fällt raus, weil nicht dynamisch genug? Relative-Layout ist eine Option. Hier darf aber die Anordnung beim Rotieren nicht unkontrolliert verändert werden. UI Team möchte, dass alle Komponenten gleich groß sind. In dem Fall kann man Gridlayout benutzen.
- Wie soll die Eingabe von Funktionen im Graph Operand funktionieren? Nur möglich mit bereits vorhandenen Elementen in der Oberfläche. Es öffnet sich keine Tastatur.

09.10.2019 :: 90 Minuten

Verteilung von Programmieraufgaben und Diskussion:

- Vorstellung des Backend-Entwurfs für Teammitglieder, die für das Frontend zuständig sind.
- Vorstellung des Frontend-Entwurfs für Teammitglieder, die für das Backend zuständig sind.
- Diskussion über Verbindung von Frontend und Backend. Wie abgekoppelt lässt sich der Calculator wirklich realisieren?
- Vorstellung der Hauptbibliothek die für die (aufwändigen) Rechnungen wie Nullstellenberechnung benutzt werden soll.
- Warum Apache Commons Math und nicht JScience?
- Diskussion des Programm-Workflows.
- Diskussion ob ASCII-Darstellung oder LaTeX-Rendering für Frontend benutzt werden soll

05.01.2020 :: 120 Minuten

Arbeit am Projekt:

- Aufnahme des aktuellen Projektstands.
- Besprechen des weiteren Vorgehens.
- Aufgabenabstimmung.
- Besprechung des geplanten Frontends.

- Besprechung/ Lösung von Problemen.

14.01.2019 :: 90 Minuten

Zusammenführung Frontend Backend:

- Präsentation des Frontends durch das GUI-Team.
- Besprechen von MVC-Umsetzung in Android.
- Backend Unit-Testing Fortschritte.
- Serialisierung der Stacks zur Session-Sicherung.

24.01.2019 :: 240 Minuten

- Detaillierte Ausarbeitung der Architektur im Backend.
- Programmieren im Team.
- Zusammenführen mehrere Features.
- Umbau der Programmstruktur.

28.01.2019 :: 90 Minuten

Gespräch mit Herr Prof. Dr. Thomas Seifert über den aktuellen Stand des Projekts und im Anschluss daran eine Nachbesprechung innerhalb des Teams.

Vorstellung:

- Vorstellung der bereits implementierten Grundfunktionen der App.
- Vorstellung des verwendeten Design-Patterns.
- Abgleich von Umsetzung mit den Anforderungen des Dozenten.
- Ansatz des Backends erklärt.
- Gerät ausleihen, um nicht nur mit Emulator testen zu können.
- Serialisierung der Daten (Speichern und Laden).

Ergebnis:

- Projekt ist auf einem guten Weg. Priorisiert werden sollen differenzierende Funktionen anstatt wenige Features sehr detailliert auszuarbeiten (Prototypische Arbeit).
- Ternäre, Quaternäre usw. Operationen sind gewünscht.
- Vektoren in Bestandteile lösen.
- Eingabe von Matrizen.
- Jeder Klasse muss ein Verantwortlicher zugeordnet sein.

Ideen aus der Nachbesprechung:

- "Vektor bauen" / "Vektoren auflösen" Action.
- Summe von Stack Action.
- 1x Triple Operator einfügen.
- Operanden Eingabe via einzelne Menüs.
- Ranks der Stacks anpassen.
- Format des ersten Stacks anpassen (Format nicht 0,00).

4.3 Projekttagebücher aller Teammitglieder (tabellarisch)

4.3.1 Tom Bockhorn

4.3.2 Hendrik Falk

4.3.3 Dennis Gentges

4.3.4 Getuart Istogu

4.3.5 Jannis Keienburg

4.3.6 Tim Jonas Meinerzhagen

4.3.7 Khang Pham

4.3.8 Tim Schwenke

4.4 Beschreibung von Problemen

4.4.1 Softwareentwicklung im Team [Schwenke]

Schon kurz nach der initialen Erstellung des Git-Repositories und des Projekts in Android-Studio hat sich die Frage gestellt, wie man in einem acht Mitglieder starkem Team produktiv an einer einzelnen Code-Basis arbeiten soll. Hat man ein Quellcodeverzeichnis alleine für sich reichen zumeist um die drei aktive (also nicht *stale*) Branches aus. Das wäre zunächst der `Master`-Branche, welcher die Wurzel des Verzeichnisses darstellt und – gerade, wenn Ansätze wie CI/CD verfolgt werden – die produktiven oder zumindest lauffähigen Versionen eines Projekts enthält. Im `Development`-Branch hingegen findet die Entwicklung statt. Hier ist es üblich, dass das Projekt zum Zeitpunkt einzelner Commits Fehler enthält und nicht lauffähig ist. Sobald ein Entwickler der Meinung ist, dass der Stand in `Development` veröffentlicht werden kann, wird `Development` in `Master` vereint. Wichtig zu betonen ist hier, dass dies keine feste Regel ist, sondern eher dem allgemeinen Workflow entspricht. In einem großen Team ist ein solcher Arbeitsablauf nicht mehr möglich. So müssen mehrere Entwickler parallel an dem Projekt arbeiten. Verwendet man nun das System aus zwei Branches, wird es sehr schnell zu Merge-Konflikten kommen, die die Entwickle dazu zwingen sich mehr mit der korrekten Zusammenführung als der eigentlichen Entwicklung zu beschäftigen, sofern sie ihren lokalen Arbeitsbereich aktuell halten wollen. Die nächstliegende und ebenfalls problematische Alternative ist es nur bei Fertigstellung von Funktionen, die meist aus mehreren Commits zusammengesetzt sind, das lokale Quellcodeverzeichnis

mit dem Remote zu synchronisieren. Mit dieser Herangehensweise verpasst man unter Umständen große Fortschritte im Gesamtprojekt. Die lokale Version ist plötzlich nicht mehr lauffähig und muss aufwändig angepasst werden. Deswegen haben wir uns in diesem Projekt für den Gitflow-Workflow entschieden. Grafisch dargestellt ist dieser beispielhaft in der folgenden Grafik.

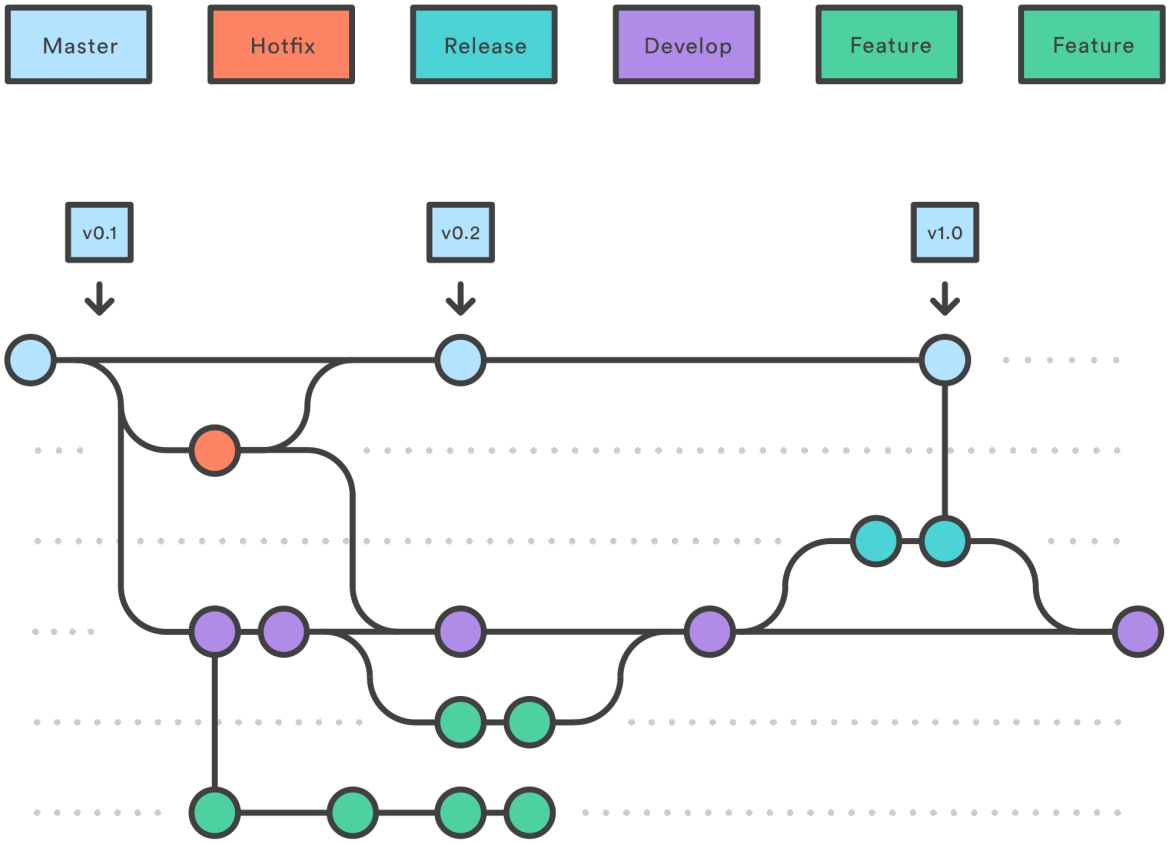


Abbildung 10: Gitflow[1]

Der Gitflow-Workflow definiert ein strenges Branching-Model und gibt jedem Typ von Branch (lediglich differenziert durch ihre Namen) eine spezifische Rolle. `Master` wird verwendet, um die Release-History festzuhalten. Hier finden sich Versionen des Projekts, die lauffähig sind und für sich alleine stehen (können). `Development` fungiert ähnlich wie `Master`, nur enthält es die gesamte Entwicklungshistorie des Projekts. Nun kommen die sogenannten `Feature`-Branches ins Spiel, die beliebig weiter untergliedert werden können. Benannt werden Features hierarchisch. In unserem Projekt haben wir folgende zwei Gruppen von Feature-Branches:

`feature/backend/<konkretes-feature>`

[1] Atlassian (2020)

`feature/frontend/<konkretes-feature>`

Jedes Feature wird einem Verantwortlichen zugeteilt und wird meist auch von diesem bearbeitet. Sobald ein Feature fertig ist, wird es in `Development` zusammengeführt. Somit werden die Abstände zwischen Zusammenführungen verringert und der Arbeitsablauf wird einfacher. Schließlich gibt es auch noch einen Hotfix-Branch, für dringende Änderungen.

Im Laufe der Entwicklung haben sich die Vorteile dieser Herangehensweise für das Team deutlich gezeigt. Unterschiedliche Features konnten, nachdem eine grundlegende Programmarchitektur umgesetzt wurde, meist ohne Probleme zusammengeführt werden.

5 Dokumentation der Software

5.1 Dokumentation der Paketstruktur des Android-Projektes

5.2 Überblick über die Activities der App bzw. der Funktionen

5.3 Dokumentation der Navigation zwischen Activities

5.4 Dokumentation der Activity-übergreifenden, persistenten Datenhaltung

5.5 Dokumentation der programmatischen Beiträge der Teammitglieder

5.5.1 Tom Bockhorn

5.5.2 Hendrik Falk

5.5.3 Dennis Gentges

5.5.4 Getuart Istogu

5.5.5 Jannis Keienburg

5.5.6 Tim Jonas Meinerzhagen

5.5.7 Khang Pham

5.5.8 Tim Schwenke

Umsetzung der generischen Kalkulationsorchestrierung

Wie schon im Kapitel zur Projektplanung erwähnt, stellt die Strukturierung und Architektur der Rechnungsumsetzung im Backend eine der vielen Anforderungen des Projekts dar. Identifiziert wurden zwei Hauptarten von Kalkulationen.

Die erste Art ist generisch und wird in einer großen Anzahl benötigt. Ein Beispiel dafür ist die Addition. Da der Taschenrechner viele unterschiedliche Operanden-Typen unterstützt (Matrizen, Brüche, Mengen, usw.) sind enorm viele Methoden notwendig, um alle Möglichkeiten der Addition abdecken zu können. Auch muss irgendwo vom Programm entschieden werden, welche Methode genau aufgerufen werden soll. Statt dies mit komplexen If-Else-Bedingungen zu lösen, wurde in der Planungsphase entschieden Reflektion zu nutzen. Somit kann man in sich geschlossene kleine Methoden programmieren, die - sofern die Schnittstellenanforderungen erfüllt sind - automatisch erkannt und von der Reflektionsmethode aufgerufen werden können. Der Nutzer im Frontend muss lediglich entscheiden, was für eine Art von generischer Kalkulation er ausführen möchte. Zum Beispiel Addition oder Multiplikation.

Die zweite Art von Kalkulationen sind sehr spezifisch, z.B. ein bestimmter Algorithmus zum Lösen von kubischen Gleichungen. Hier sind keine/kaum Kombinationen möglich und können somit direkt aufgerufen werden, ohne Reflektion zu verwenden.

Implementiert ist die Reflektion in der abstrakten Klasse `Action`. Die Klassenvariable `scopedAction` zeigt zur Laufzeit auf eine konkrete Implementierung einer `Action`, also z.B. `Plus`. Auf `scopedAction` wird die Reflektion ausgeführt. Letztere ist in der Methode `with()` umgesetzt. Diese stellt die Schnittstelle zu den generischen Kalkulationen erster Art dar. Hier ist der Methodenkopf zu sehen:

Listing 2: Methodenkopf der generischen Schnittstelle

```
@Contract(pure = true) public @NotNull
Operand with(@NotNull Operand... operands)
throws CalculationException
```

Die erste Zeile definiert einige Eigenschaften der Methode. `@Contract` sagt aus, dass die Funktion *pure* ist. Sie gibt für Tupel von Operanden immer das gleiche Ergebnis zurück und ist grundsätzlich ohne Nebeneffekte. Das ist hilfreich für das automatische Testen. Als Parameter wird ein beliebig großes Array von Operanden übergeben. Das Ergebnis

ist immer eine valide Instanz von `Operand`. Wird versucht eine nicht unterstützte Kalkulation auszuführen, wird `CalculationException` geworfen. Diese Ausnahme ist keine `RunTimeException` und muss deswegen explizit behandelt werden. Alternativ hätte man hier auch Optionals nutzen. Jedoch unterstützt die genutzte Version der Android API dieses Java-Feature nicht.

Listing 3: Implementierung der generischen Schnittstelle

```
Class[] operandClasses = new Class[operands.length];
Operand resultOperand;

for (int i = 0; i < operands.length; i++)
    operandClasses[i] = operands[i].getClass();

try {
    resultOperand = (Operand) scopedAction.getClass()
        .getDeclaredMethod(
            "on",
            operandClasses
        ).invoke(
            scopedAction,
            (Object[]) operands
        );
} catch (SeveralExceptions e) {
    throw new CalculationException(e.getMessage());
}

if (resultOperand != null) return resultOperand;
else throw new CalculationException();
```

Die Reflektion in Listing 3 beginnt mit der Extraktion der Klasse jedes übergebenen Operands. Das kann z.B. die Klasse `Matrix` oder `Fraction` sein, die alle von `Operand` erben. Die extrahierten Klassen werden in Array `operandClasses` gespeichert. Die hier vorliegende Sequenz liefert die Antwort auf die Frage, welche konkrete Methode aufgerufen werden soll. Die Entscheidung basiert alleine auf dieser Sequenz und der konkreten `Action` auf die `scopedAction` zeigt. Aus letzterer Variable wird die Klasse extrahiert und die Methode `getDeclaredMethod()` aufgerufen. Damit kann man eine Methode in einer Klasse auf Basis des Namens (in unserem Falle immer `on`) und eine Sequenz von Parametertypen finden. Diese wird anschließend mit `invoke()` aufgerufen, wobei die Operanden übergeben werden. Kommt es zu einem Fehler werden alle Fehlertypen in `CalculationException` zusammengefasst und weitergegeben. Ansonsten wird das Ergebnis zurückgegeben.

6 Dokumentation der sonstigen Beiträge der Teammitglieder

6.1 Tom Bockhorn

6.2 Hendrik Falk

6.3 Dennis Gentges

6.4 Getuart Istogu

6.5 Jannis Keienburg

6.6 Tim Jonas Meinerzhagen

6.7 Khang Pham

6.8 Tim Schwenke

7 Fazits aller Teammitglieder

7.1 Tom Bockhorn

7.2 Hendrik Falk

7.3 Dennis Gentges

7.4 Getuart Istogu

7.5 Jannis Keienburg

7.6 Tim Jonas Meinerzhagen

7.7 Khang Pham

7.8 Tim Schwenke

Anhang (Quelltext)

Anhangsverzeichnis

Anhang 1 Gesprächsnotizen

Anhang 1.1 Gespräch mit Werner Müller

Gespräch mit Werner Müller am 01.01.2013 zum Thema XXX:

- Über das gute Wetter gesprochen
- Die Regenwahrscheinlichkeit liegt immer bei ca. 3%
- Das Unternehmen ist total super
- Hier könnte eine wichtige Gesprächsnotiz stehen

Anhang (Verzeichnis der verwendeten Tools und Hilfsprogramme)

Anhangsverzeichnis

Anhang 2 Gesprächsnotizen

Anhang 2.1 Gespräch mit Werner Müller

Gespräch mit Werner Müller am 01.01.2013 zum Thema XXX:

- Über das gute Wetter gesprochen
- Die Regenwahrscheinlichkeit liegt immer bei ca. 3%
- Das Unternehmen ist total super
- Hier könnte eine wichtige Gesprächsnotiz stehen

Quellenverzeichnis

Monographien

Meier, Reto (2010). *Professional Android 2 Application Development*. Indianapolis, IN, USA: Wiley, S. 576.

Aufsätze in Sammelbänden und Zeitschriften

Hocking, Christopher G., Steven M. Furnell, Nathan L. Clarke und Paul L. Reynolds (2010). „A Distributed and Cooperative User Authentication Framework“. In: *Proceedings of International Conference on Information Assurance and Security (IAS)*, S. 304–310.

Internetquellen

Maslennikov, Denis (2011). *ZeuS-in-the-Mobile – Facts and Theories*. URL: `http://www.securelist.com/en/analysis/204792194/ZeuS%5C_in%5C_the%5C_Mobile%5C_Facts%5C_and%5C_Theories` (besucht am 20. Dez. 2012).

Ehrenwörtliche Erklärung

Hiermit erkläre ich, dass ich die vorliegende Studienarbeit selbständig angefertigt habe. Es wurden nur die in der Arbeit ausdrücklich benannten Quellen und Hilfsmittel benutzt. Wörtlich oder sinngemäß übernommenes Gedankengut habe ich als solches kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Bergisch Gladbach, 3. Februar 2020

_____

Max Mustermann

8 Einleitung

Dies ist eine Vorlage zum Erstellen von Bachelorarbeiten an der FHDW mit dem Schriftsatzsystem LaTeX.

Die in der Vorlage verwendeten Pakete und Styles sind sehr gut dokumentiert. Bei der Beprechung der einzelnen Paket wird auf die jeweilige Dokumentation verwiesen, die standardmäßig mit den jeweiligen Paketen installiert wird.

9 Installation

9.1 TeX-Distribution

Für die Arbeit mit LaTeXist eine aktuelle TeX-Distribution erforderlich.

9.1.1 Windows

Unter Windows ist MiKTeX die Standard-LaTeX-Distribution. Der MikTex-Installer kann unter `http://miktex.org/download` heruntergeladen werden.

9.1.2 Linux

Die Standard-LaTeX-Distribution unter Linux ist Tex Live, welche über die gängigen Software-Repositories installiert werden kann.

Unter Debian/Ubuntu kann die Installation der erforderlichen Pakete mittels der folgenden Befehlen durchgeführt werden:

```
sudo apt-get install texlive-latex-base
sudo apt-get install texlive-latex-recommended
sudo apt-get install texlive-fonts-recommended
sudo apt-get install biblatex
sudo apt-get install biber
```

9.1.3 Mac-OS

Von der Tex-User-Group wird jährlich ein komplettes aktuelles MacTeX-Paket angeboten (http://www.tug.org/mactex/index.html), in dem alle relevanten Programme und Pakete enthalten sind.

9.2 PDF-Viewer

9.2.1 Windows

Als PDF-Viewer unter Windows bietet sich der freie Sumatra PDF Viewer an: `http://blog.kowalczyk.info/software/sumatrapdf/download-free-pdf-viewer-de.html`

9.2.2 Linux und Mac-OS

Die installierten Standard-PDF-Viewer unter Linux bzw. Mac-OS können problemlos genutzt werden.

9.3 Hello World

Nach der Installation sollte ein erster Test der Vorlage versucht werden. Dazu öffnen Sie ein Kommandozeilenfenster und wechseln in das Verzeichnis, in dem sich die LaTeX-Quellen dieser Vorlage befinden. Anschließend müssen auf der Kommandozeile die Befehle

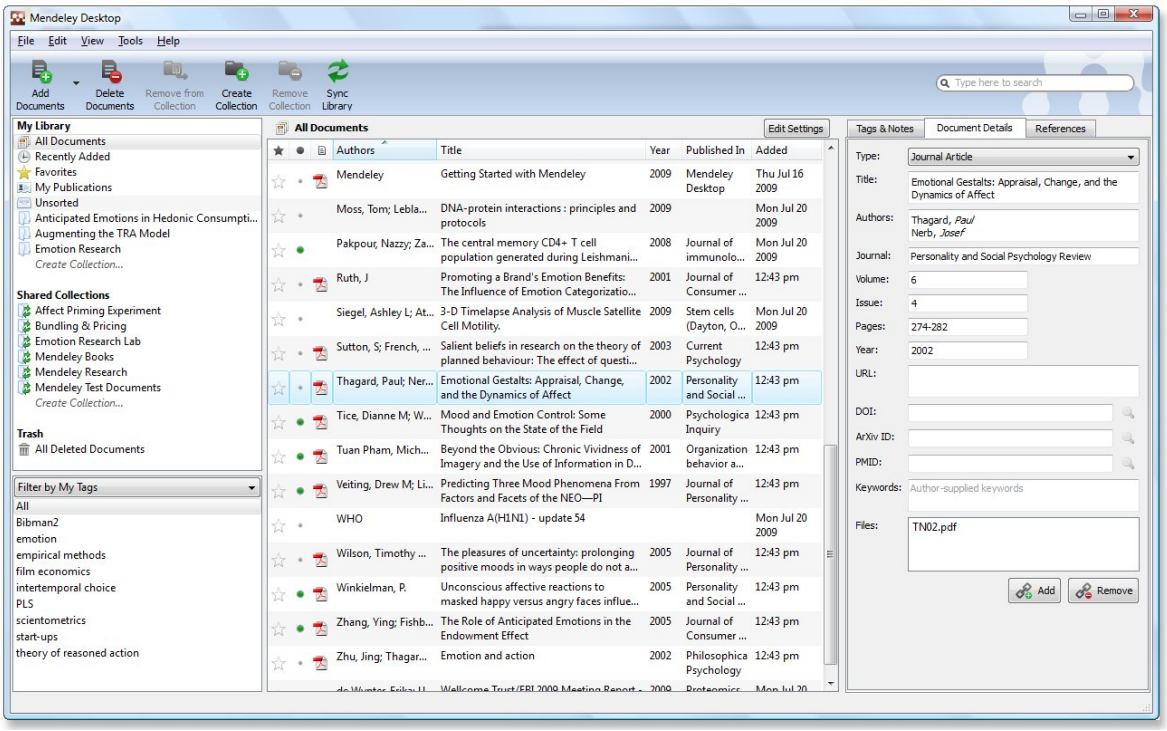
```
biber Thesis
pdflatex Thesis
```

eingegeben werden. Nun sollte eine neue Datei `Thesis.pdf` erzeugt worden sein. Falls nicht, sehen Sie bitte in den Ausgaben nach, die LaTeXerzeugt hat. Diese sind recht umfangreich, auch wenn ein PDF-Dokument erzeugt werden konnte.

9.4 Literaturverwaltung

Für die Verwaltung von Quellen eignet sich das freie, Cloud-basierte Mendely: `http://www.mendeley.com/download-mendeley-desktop/`.

Abbildung 11: Mendeley Referenzmanager



Quelle: `http://dominique-fleury.com/?p=302`

9.5 Texteditor

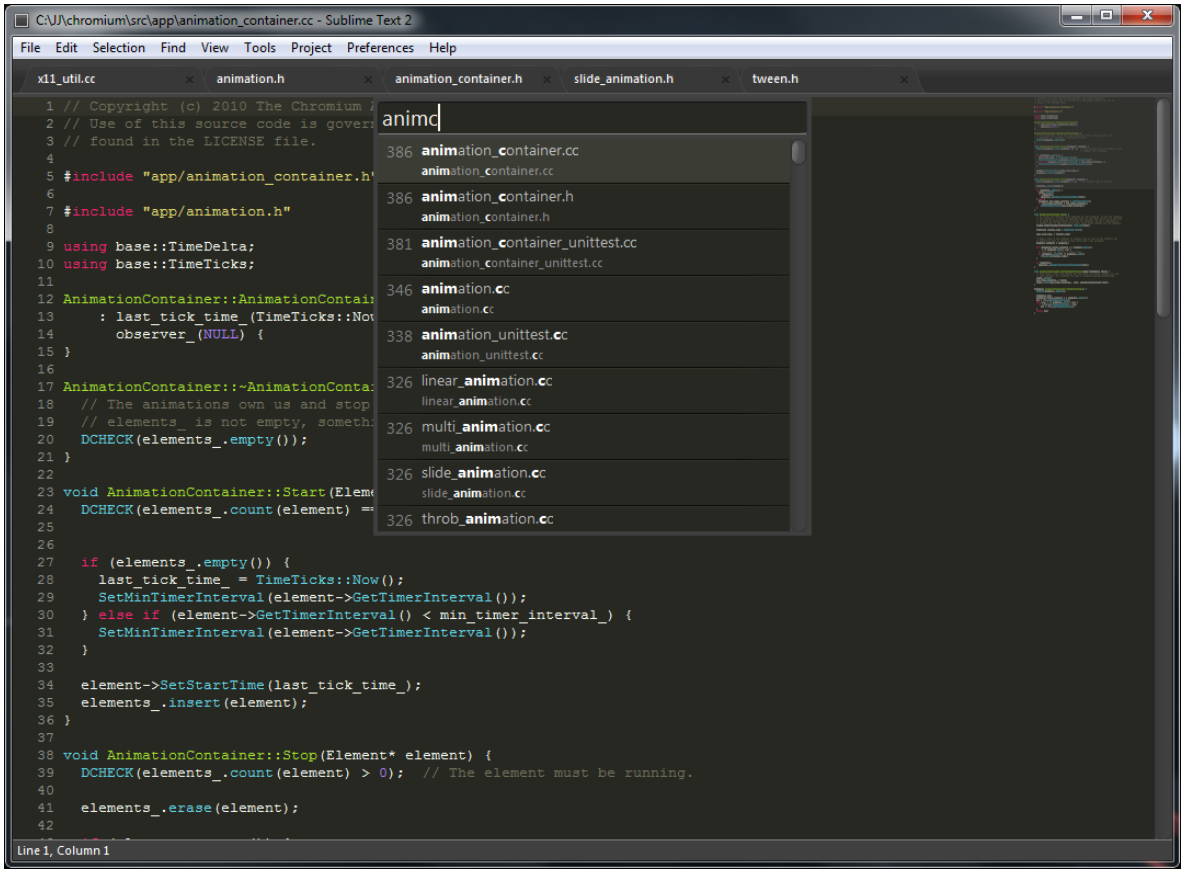
Als Texteditor für LaTeXwird Sublime Text (`http://www.sublimetext.com`) empfohlen. Zur Arbeit mit Latex ist das Plugin *LaTeXTools* erforderlich (`https://github.com/SublimeText/LaTeXTools`).

9.6 PDF-Erzeugung

Für die Erzeugung des PDF-Dokuments inklusive Referenzen, Quellenverzeichnis und Glossar sind mehrere Programmaufrufe und -durchläufe erforderlich. Der vollständige Aufruf zur PDF-Erzeugung lautet:

```
pdflatex Thesis
biber Thesis
makeindex -s Thesis.ist -t Thesis.alg -o Thesis.acr Thesis.acn
```

Abbildung 12: Sublime Texteditor



Quelle: `http://www.sublimetext.com/screenshots/alpha_goto_anything2_large.png`

```
makeglossaries Thesis

pdflatex Thesis

pdflatex Thesis
```

10 Grundlagen

10.1 Schrift

10.1.1 Schriftgrößen

Das ist sehr kleine Schrift

Das ist kleine Schrift

Das ist normale Schrift

Das ist große Schrift

Das ist größere Schrift

Das ist noch größere Schrift

Das ist riesige Schrift

Das ist noch riesigere Schrift

Das ist Script Schrift

Das ist Fußnoten Schrift

10.1.2 Schrift Typen

Das ist ein fetter Text

Das ist ein kursiver Text

Das ist ein unterstrichener Text

Das ist ein kapitälchen Text

Das ist ein serifenloser Text

`Das ist ein Schreibmaschinen Text`

Das ist ein normaler Text

10.1.3 Schrift Ausrichtung

> Quote Text (Der gesamte Text innerhalb der Umgebung wird von beiden Seiten eingerückt)

Zentrierter Text (Der gesamte Text innerhalb der Umgebung wird zentriert)

Linksbündiger Text (Der gesamte Text innerhalb der Umgebung wird linksbündig)

Rechtsbündiger Text (Der gesamte Text innerhalb der Umgebung wird rechtsbündig)

In einer Fußnote[2]

10.2 Abbildungen

In Abb. 13 sehen Sie das Logo der FHDW.

Abbildung 13: Das Logo der FHDW

Quelle: Eigene Darstellung

10.3 Tabellen

In Tabelle 1 auf Seite 40 sehen Sie die am häufigsten benutzten PINs.

10.4 Zitate

Ein Zitat im Fließtext ist zu sehen bei

Ein vergleichendes Zitat.[3]

[2]können zusätzliche Ergänzungen, Präzisierungen, Textverweise usw. eingeführt werden.
[3]vgl. Maslennikov, Denis (2011), S. 5 ff.

Tabelle 1: Die am häufigsten verwendeten PINs

Rank	PIN	Percentage	Accumulated
1	1234	4.34%	4.34%
2	0000	2.57%	6.91%
3	2580	2.32%	9.23%
4	1111	1.60%	10.83%
5	5555	0.87%	11.70%
6	5683	0.70%	12.39%
7	0852	0.60%	12.99%
8	2222	0.56%	13.55%
9	1212	0.49%	14.03%
10	1998	0.43%	14.46%

Quelle: Eigene Darstellung

Ein „wörtliches Zitat“[4]

Zitat einer Quelle mit mehreren Autoren.[5]

10.5 Abkürzungen

Bei der ersten Verwendung werden Abkürzungen ausgeschrieben: Advanced Encryption Standard (AES). Später wird dann automatisch nur noch die Kurzform benutzt: AES

10.6 Listen

Eine einfache List mit Punkten:

- Punkt 1
- Punkt 2
- Punkt 3

Eine einfache Liste mit Nummern:

1. Punkt 1

[4]Meier, Reto (2010), S. 13 f.
[5]vgl. Hocking, Christopher G. u. a. (2010), S. 32 ff.

2. Punkt 2
3. Punkt 3

Eine einfache Liste mit römischen Nummern:

I. Punkt 1
II. Punkt 2
III. Punkt 3

Eine einfache Liste mit Buchstaben:

(a) Punkt 1
(b) Punkt 2
(c) Punkt 3

10.7 Quelltext

Listing 4 auf Seite 41 zeigt einigen Quelltext.

Listing 4: Scanning for Wi-Fi Access Points on Android

```
registerReceiver(new RSSIBroadcastReceiver(),
    new IntentFilter(WifiManager.SCAN_RESULTS_AVAILABLE_ACTION));

WifiManager wifi = getSystemService(Context.WIFI_SERVICE);
wifi.startScan();

/* not thread safe */
public class RSSIBroadcastReceiver extends BroadcastReceiver {

    public void onReceive(Context context, Intent intent) {
        WifiManager wifi = getSystemService(Context.WIFI_SERVICE);
        List<ScanResult> scanResults = wifiManager.getScanResults();

        for (ScanResult scanResult : results) {
            RSSI rssi = new RSSI();
            rssi.bssi = scanResult.BSSID;
            rssi.signalLevel = scanResult.level;
        }
    }
}
```

11 Zusammenfassung

Dieses Dokument ist eine Hilfe, um die Formalien für eine Bachelor-Thesis an der FHDW bei der Verwendung von LaTeX zu erfüllen und dabei möglichst viele Automatismen von LaTeX zu nutzen. Eine Absprache mit dem betreuenden Professor ist dennoch ratsam.