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: 4. Februar 2020

Inhaltsverzeichnis

Abbildungsverzeichnis

Listingverzeichnis

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

Datum	Dauer	Beschreibung
03.09.2019	200 Min.	Auswahl des Projekttyps. Entscheidung für die Entwicklung einer Android-App. 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 Min.	Einigung mit dem Dozenten auf einen Ansatz für den Taschenrechner. 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 Min.	Fertigstellung des Konzeptes. Fortschritte beim Paper-Prototypen. Android-Umgebung ist bei allen Team-Mitgliedern komplett aufgesetzt und lauffähig.
	60 Min.	Besprechung des aktuellen Stands des Konzepts und des Prototypen. Erstellung eines Mid-Fidelity Prototypen.

Tabelle wird auf der nächsten Seite fortgesetzt...

...Forsetzung der Tabelle

Datum	**Dauer**	**Beschreibung**
		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?
	40 Min.	Projektplanung mithilfe von Projektstrukturplan und weiteren Methoden. Übertragen der Ergebnisse in den Teams-Planner. Grobe Verteilung der Arbeitspakete innerhalb des Teams.
17.09.2019	60 Min.	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.
17.09.2019	90 Min.	Wie sollen Elemente in der GUI dargestellt werden? Als Character 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.

Tabelle wird auf der nächsten Seite fortgesetzt...

...Forsetzung der Tabelle

Datum	**Dauer**	**Beschreibung**
		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 Min.	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.
05.01.2020	120 Min.	Aufnahme des aktuellen Projektstands. Besprechung des weiteren Vorgehens. Aufgabenabstimmung. Besprechung des geplanten Frontends. Besprechung/ Lösung von Problemen.
14.01.2019	90 Min.	Zusammenführung Frontend Backend Präsentation des Frontends durch das GUI-Team. Besprechen von MVC-Umsetzung in Android. Backend Unit-Testing Fortschritte.

Tabelle wird auf der nächsten Seite fortgesetzt...

...Forsetzung der Tabelle

Datum	**Dauer**	**Beschreibung**
		Serialisierung der Stacks zur Session-Sicherung.
24.01.2019	240 Min.	Detaillierte Ausarbeitung der Architektur im Backend. Programmieren im Team. Zusammenführen mehrere Features. Umbau der Programmstruktur.
28.01.2019	90 Min.	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).

Tabelle wird auf der nächsten Seite fortgesetzt...

...Forsetzung der Tabelle		
Datum	**Dauer**	**Beschreibung**
		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.
03.02.2019	90 Min.	Besprechen des aktuellen Stands der App. Was muss noch unbedingt umgesetzt werden? Aufteilung der noch offenen Kapitel in der Ausarbeitung. Neues Kapitel ”Einleitung” mit Motivation. Anpassung einiger Kapitelbezeichnungen an Gegebenheiten des Projekts. Koordination der Ausarbeitung.
Summe der Dauer aller Meetings beträgt 21 Stunden		

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 wird im Rahmen dieses Projekts der *Gitflow-Workflow* verwendet. Grafisch dargestellt ist dieser beispielhaft in der folgenden Grafik.

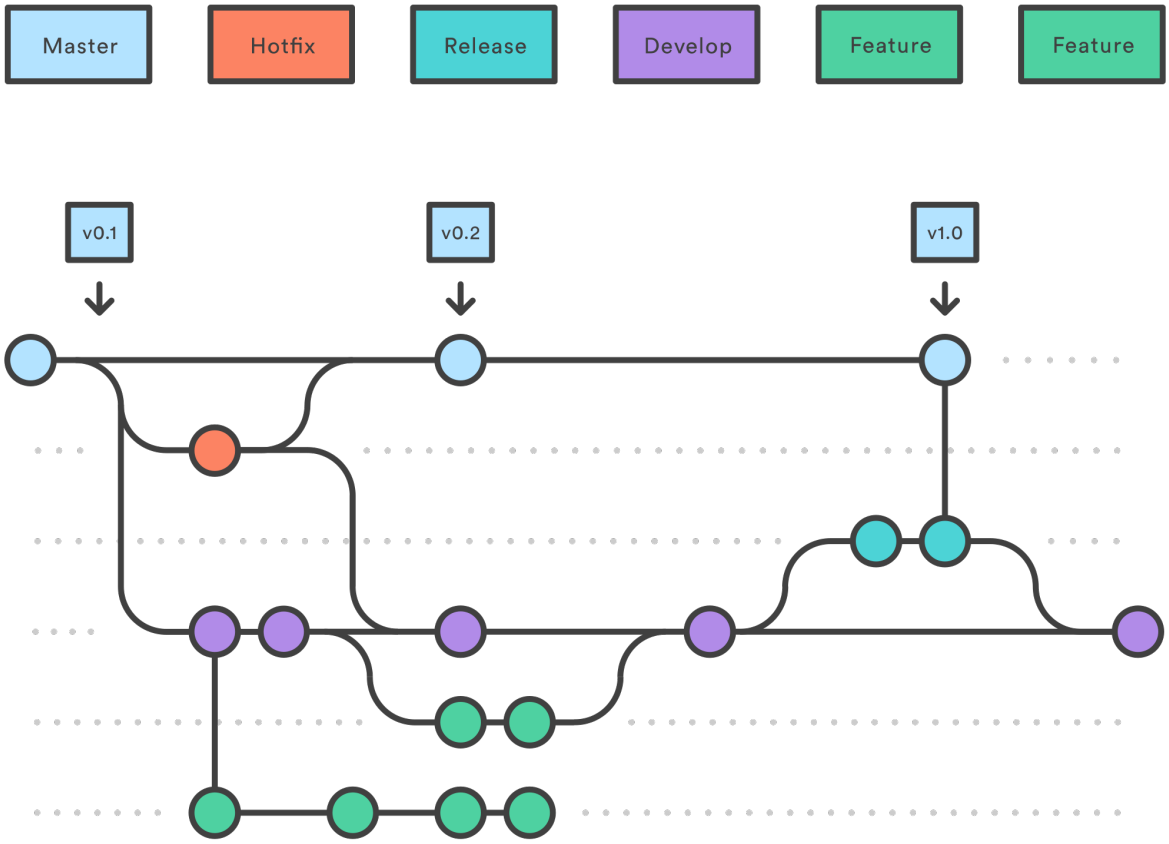

Abbildung 10: Gitflow[1]

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. Benannt werden Features hierarchisch. Im Projekt werden folgende zwei Gruppen verwendet:

`feature/backend/<konkretes-feature>`

`feature/frontend/<konkretes-feature>`

[1] Atlassian (2020)

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 worden ist, 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

Generische 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

8 Quellenverzeichnis

iju

9 Anhang - Quelltext

9.1 Model

9.1.1 Calculation

Listing 4: Action

```
XXX
```

Listing 5: ArcCosinus

```
XXX
```

Listing 6: ArcSinus

```
XXX
```

Listing 7: ArcTangens

```
XXX
oinion
```

Listing 8: CalculationException

```
XXX
```

Listing 9: Cosinus

```
XXX
```

Listing 10: Derivation

```
XXX
```

Listing 11: HighAndLowPoints

```
XXX
```

Listing 12: Integral

```
XXX
```

Listing 13: Limes

```
XXX
```

Listing 14: Logarithm

```
XXX
```

Listing 15: MatrixUtil

```
XXX
```

Listing 16: Minus

```
XXX
```

Listing 17: Modulo

```
XXX
```

Listing 18: Plus

```
XXX
```

Listing 19: Power

```
XXX
```

Listing 20: Sinus

```
XXX
```

Listing 21: Slash

```
XXX
```

Listing 22: Tangens

```
XXX
```

Listing 23: Times

```
XXX
```

Listing 24: Zeros

```
XXX
```

Listing 25: Root

```
XXX
```

9.1.2 Operands

Listing 26: DoubleComparator

```
XXX
```

Listing 27: DoubleFormatter

```
XXX
```

Listing 28: Element

```
XXX
```

Listing 29: ODouble

```
XXX
```

Listing 30: OEmpty

```
XXX
```

Listing 31: OFraction

```
XXX
```

Listing 32: OMatrix

```
XXX
```

Listing 33: OPolynom

```
XXX
```

Listing 34: OSet

```
XXX
```

Listing 35: OTuple

```
XXX
```

Listing 36: Operand

```
XXX
```

9.1.3 Settings

Listing 37: AllClear

```
XXX
```

Listing 38: ClearHistory

```
XXX
```

Listing 39: DeleteEntry

```
XXX
```

Listing 40: Dot

```
XXX
```

Listing 41: Enter

```
XXX
```

Listing 42: Inverse

```
XXX
```

Listing 43: LoadLayout

```
XXX
```

Listing 44: SaveLayout

```
XXX
```

Listing 45: Setting

```
XXX
```

Listing 46: Split

```
XXX
```

Listing 47: Swap

```
XXX
```

Listing 48: TurnAroundSign

```
XXX
```

9.1.4 Stack

Listing 49: StackInterface

```
XXX
```

Listing 50: OperandStack

```
XXX
```

9.2 View

9.2.1 Layout

Listing 51: ScreenOrientation

```
XXX
```

Listing 52: StorageLoadingException

```
XXX
```

Listing 53: TileLayout

```
XXX
```

Listing 54: TileLayoutFactory

```
XXX
```

Listing 55: TileLayoutLoader

```
XXX
```

9.2.2 Menu

Listing 56: ChooseListMenu

```
XXX
```

Listing 57: DialogMenu

```
XXX
```

Listing 58: InputDouble

```
XXX
```

Listing 59: InputFraction

```
XXX
```

Listing 60: InputMenuFactory

```
XXX
```

Listing 61: InputPolynomial

```
XXX
```

Listing 62: InputTileType

```
XXX
```

9.2.3 Schemas

Listing 63: ActionTileScheme

```
XXX
```

Listing 64: ErrorTileScheme

```
XXX
```

Listing 65: HistoryTileScheme

```
XXX
```

Listing 66: OperandTileScheme

```
XXX
```

Listing 67: SettingTileScheme

```
XXX
```

Listing 68: StackTileScheme

```
XXX
```

Listing 69: TileScheme

```
XXX
```

9.2.4 Sonstiges

Listing 70: Tile

```
XXX
```

Listing 71: TileMapping

```
XXX
```

Listing 72: TileType

```
XXX
```

Listing 73: TypeQuestionable

```
XXX
```

Listing 74: MainActivity

```
XXX
```

9.3 Presenter

Listing 75: Presenter

```
XXX
```

10 Anhang - Verwendeten Tools und Hilfsprogramme

kio

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, 4. Februar 2020

Max Mustermann